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

D3.3: Specification of the 5G-MEDIA QoS Control and Management Tools

Lead partner: UCL
Authors:
- David Griffin, Khoa Phan, Morteza Kheirkhah, Miguel Rio [UCL]
- Panagiotis Athanasoulis, Stamatia Rizou [SiLO]
- Gino Carrozzo, Francesca Moscatelli [NXW]
- David Breitgand, Avi Weit [IBM]
- Francesco Iadanza [ENG]
- Gordana Macher, Madeleine Keltsch [IRT]
- Alberto Flórez Pagés [TID]

Delivery date (DoA): 31 August 2018
Actual delivery date: 31 August 2018
Dissemination level: Public
Version number: 1.0 Status: Final

Grant Agreement N°: 761699
Project Acronym: 5G-MEDIA
Project Title: Programmable edge-to-cloud virtualisation fabric for the 5G Media industry
Instrument: IA
Call identifier: H2020-ICT-2016-2
Topic: ICT-08-2017, 5G PPP Convergent Technologies, Strand 2: Flexible network applications
Start date of the project: 1 June, 2017
Duration: 30 months

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>ToC</td>
<td>15/5/18</td>
<td>UCL</td>
<td>Initial document outline</td>
</tr>
<tr>
<td>0.0</td>
<td>21/6/18</td>
<td>UCL</td>
<td>Template and section assignment</td>
</tr>
<tr>
<td>0.1</td>
<td>19/7/18</td>
<td>UCL</td>
<td>First version with initial contributions</td>
</tr>
<tr>
<td>0.2</td>
<td>24/7/18</td>
<td>UCL</td>
<td>Further contributions incorporated</td>
</tr>
<tr>
<td>0.3</td>
<td>26/7/18</td>
<td>UCL</td>
<td>Minor edits and revisions</td>
</tr>
<tr>
<td>0.4</td>
<td>27/7/18</td>
<td>UCL</td>
<td>Further sections integrated</td>
</tr>
<tr>
<td>0.5</td>
<td>22/8/18</td>
<td>UCL</td>
<td>Comments from interim reviews incorporated</td>
</tr>
<tr>
<td>0.6</td>
<td>24/8/18</td>
<td>UCL</td>
<td>Complete version for review</td>
</tr>
<tr>
<td>0.7</td>
<td>29/8/18</td>
<td>UCL</td>
<td>Final draft</td>
</tr>
<tr>
<td>1.0</td>
<td>31/08/2018</td>
<td>ENG</td>
<td>Final version and submission</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>Interim review</td>
<td>3/8/18</td>
<td>Panagiotis Athanasoulis, Stamatia Rizou [SILO]</td>
<td>Comments and suggestions for improvement.</td>
</tr>
<tr>
<td>Interim review</td>
<td>21/8/18</td>
<td>David Breitgand [IBM]</td>
<td>Comments and suggestions for improvement.</td>
</tr>
<tr>
<td>Review</td>
<td>28/08/18</td>
<td>David Breitgand [IBM]</td>
<td>Comments and suggestions for improvement.</td>
</tr>
<tr>
<td>Review</td>
<td>29/8/18</td>
<td>Gordana Macher, Madeleine Keltsch [IRT]</td>
<td>Comments and suggestions for improvement.</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 TILEPIKIONION 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 POLITÉCNICA 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** .................................................................................................................. 11

1. **QOS CONTROL AND MANAGEMENT IN 5G-MEDIA** ......................................................... 12

2. **MAPE ARCHITECTURE** ............................................................................................................. 13
   2.1. Messaging architecture .............................................................................................................. 16

3. **MONITORING** ............................................................................................................................ 19
   3.1. NFVI MONITORING ................................................................................................................ 19
       3.1.1. OpenStack ......................................................................................................................... 20
       3.1.2. OpenNebula .................................................................................................................... 24
       3.1.3. Telegraf and Logstash integrated with Elasticsearch and Kibana .................................. 27
       3.1.4. Unikernels ....................................................................................................................... 30
       3.1.5. FaaS/Kubernetes ............................................................................................................ 32
   3.2. NETWORK MONITORING ........................................................................................................ 32
       3.2.1. Active network monitoring probes and metrics ............................................................... 32
       3.2.2. Provided functionalities and utilisation examples ............................................................. 33

4. **TRANSLATION** ........................................................................................................................... 36

5. **PRE-PROCESS & ANALYSIS** .................................................................................................... 38
   5.1. MACHINE LEARNING APPROACHES FOR FORECASTING DEMAND AND RESOURCE CONSUMPTION ................................................................. 39
       5.1.1. Feed forward neural networks ........................................................................................ 39
       5.1.2. Recurrent Neural Network ............................................................................................ 40
       5.1.3. Long Short-Term Memory ............................................................................................. 40
   5.2. PRE-PROCESS & ANALYSIS ALGORITHMS IN 5G-MEDIA .................................................. 41
       5.2.1. ML algorithms for initial implementation of the pre-process & analysis service ......... 42
   5.3. ONGOING WORK ON ML ALGORITHMS FOR THE PRE-PROCESS & ANALYSIS SERVICE ................................................................. 47

6. **PLANNING** .................................................................................................................................. 49
   6.1. OPTIMISATION ALGORITHMS FOR THE PLACEMENT AND CONFIGURATION OF VNFFGs IN MEDIA SERVICES .................................................... 49
       6.1.1. VNFFG structures ............................................................................................................ 50
   6.2. PLANNING ALGORITHMS FOR THE 5G-MEDIA USE CASES ........................................... 51
       6.2.1. Use Case 1: Immersive Media ....................................................................................... 51
       6.2.2. Use Case 2: Mobile Contribution, Remote and Smart Production in Broadcasting .... 52
       6.2.3. Use Case 3: Ultra-high Definition (UHD) over Content Distribution Networks (CDN) . 53

7. **EXECUTION** ................................................................................................................................ 55

8. **CONCLUSIONS** ............................................................................................................................. 56
9. APPENDIX A: PLANNING ALGORITHM FOR OPTIMISING THE DEPLOYMENT OF NSS AND VNFFGS

9.1. OPTIMISATION ALGORITHM ........................................................................................................57
  9.1.1. Chain structure .......................................................................................................................57
  9.1.2. Tree or parallel structure ........................................................................................................60
  9.1.3. Cycle or loop structure ..........................................................................................................61
9.2. EVOLUTIONARY ALGORITHM FOR MANAGING VNFFGS ....................................................61
  9.2.1. Genetic algorithm .................................................................................................................62
  9.2.2. Order-based genetic algorithm ............................................................................................63
9.3. EVALUATION ...........................................................................................................................64
  9.3.1. Pareto graph of utility vs. cost ..............................................................................................65
  9.3.2. Minimum cost vs. maximum utility vs. GA algorithms ..........................................................66
List of Figures

Figure 1: High level architecture of the Media Service MAPE ................................................. 12
Figure 2. MAPE system architecture .................................................................................. 13
Figure 3. Microservice-based components of the Media Service MAPE. ......................... 16
Figure 4. Model of Kafka bus serving message exchange in the MAPE loop .................... 17
Figure 5. Visualisation of monitoring metrics per Network Service ................................... 19
Figure 6. Data collection in the OpenStack NFVI ............................................................... 20
Figure 7. A message produced by the ceilometer in OpenStack NFVI ............................... 22
Figure 8. Data collection in the OpenNebula NFVI ........................................................... 25
Figure 9. Indicative output of the OpenNebula monitoring system for a set of VMs ......... 27
Figure 10. Elastic Stack functional architecture ............................................................... 29
Figure 11. Telegraf integrated with ELK ........................................................................... 29
Figure 12. Network statistics collected via Telegraf on the vCache VM .............................. 30
Figure 13. General diagram of active monitoring on deployed Network Services ............. 33
Figure 14. Example message produced by the translation service of MAPE .................... 37
Figure 15. Sample of feed-forward neural network ......................................................... 39
Figure 16. Recurrent neural network (RNN) ................................................................... 40
Figure 17. Long short term memory (LSTM) ................................................................. 40
Figure 18. Normal working state of the UHD CDN in UC3 ............................................. 41
Figure 19. Scaling out of the UHD CDN NS to deal with forecasted network congestion .... 41
Figure 20. Trace of background + video traffic .............................................................. 43
Figure 21. Trace of anomaly traffic. ................................................................................ 43
Figure 22. Aggregate traffic trace .................................................................................... 44
Figure 23. Neural network used in CNO ........................................................................ 45
Figure 24. ReLU activation function ............................................................................... 46
Figure 25. Aggregate traffic trace for testing ML model ................................................... 46
Figure 26. Samples of training data with absolute values ............................................... 47
Figure 27. Samples of training data with delta values ..................................................... 47
Figure 28. VNFFG chain structure (use case 3) ............................................................... 50
Figure 29. VNFFG tree or parallel structure (use case 2) ............................................... 50
Figure 30. VNFFG loop structure ................................................................................... 50
Figure 31. Auxiliary graph for the chain structure. ......................................................57
Figure 32. Summary of key notations ........................................................................58
Figure 33. MILP for chain structure .........................................................................58
Figure 34. MILP for tree structure ............................................................................60
Figure 35. MILP for cycle structure ..........................................................................61
Figure 36. Example of chain structure ......................................................................62
Figure 37. Genetic algorithm flow chart .....................................................................62
Figure 38. Pareto graph of utility vs. cost ..................................................................65
Figure 39. Cost of min cost vs. max utility vs. GA algorithm .....................................66
Figure 40. Utility of min cost vs. max utility vs. GA algorithm ....................................66
List of Tables

Table 1. Topics defined in MAPE brokering mechanism ................................................................. 18
Table 2. Metrics collected by ceilometer in OpenStack ................................................................. 23
Table 3. Host information and metrics collected by OpenNebula ................................................... 25
Table 4. VM monitoring metrics collected by OpenNebula .............................................................. 26
Table 5. List of VM states ................................................................................................................ 26
Table 6. Attributes and Information of messages produced by the translation service ........... 36
# Definitions and acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</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>Full Form</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 deliverable reports on the specification and design of the 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. In this deliverable we detail the five services that constitute the MAPE component.

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 two broad techniques deployed to date are demand forecasting and anomaly detection. We report in this deliverable on the initial design of an anomaly detection algorithm using artificial neural networks in the context of the 5G-MEDIA ultra-high definition content distribution networks.

The planning service executes algorithms to configure and adapt the deployment of the Network Services (NS) and the infrastructures upon which they run. The output of the optimisation algorithms are implemented by the execution service to configure the MANO components and the Virtual Network Functions (VNF) forming the NS. An initial algorithm for optimising NS placement and VNF forwarding graphs is evaluated and an outline of future enhancements to the planning algorithms in all three use case scenarios is presented.

The initial designs of each of the MAPE services as presented in this report, are currently being implemented in the project’s testbeds. The final design, algorithm specification and evaluation results will be documented in deliverable D3.4 due at the end of the project.
1. **QoS Control and Management in 5G-MEDIA**

QoS Control and Management in 5G-MEDIA is undertaken by the Monitoring, Analysis, Planning and Execution (MAPE) component of the Service Virtualisation Platform (SVP). In deliverable D3.1 this was presented at a high level as shown in Figure 1.

![Figure 1: High level architecture of the Media Service MAPE](image)

In this deliverable we refine the specification further and provide detail on each of the services of the MAPE component. The two main sub functions of the Media Service MAPE are Quality of Service (QoS)/Quality of Experience (QoE) Monitoring and the Cognitive Network Optimiser (CNO). These are decomposed further in this document according to the MAPE architecture presented in the following section.
2. **MAPE architecture**

The logical flow of Media Service MAPE component is shown in Figure 2. The main objectives of the 5G-MEDIA Media Service MAPE component are:

- To collect and store metrics about the status of resources in the integrated NFVIs, the network conditions and the performance and behaviour of the instantiated NSs and media applications.
- To organise and harmonise the collected metrics under a common data model and make them available to all interested services and users of the 5G-MEDIA SVP and Application SDK (e.g. accounting services, visualisation tools, NS developers, platform administrators, etc.).
- To drive Machine Learning (ML) and resource planning algorithms aiming to predict future conditions in user demand and optimise utilisation of Network Functions Virtualisation Infrastructure (NFVI) resources, respectively.
- To apply timely recommendations and planning policies through Open Source MANO (OSM) mechanisms that could optimise resource management, network performance and secure QoS/QoE and Service Level Agreement (SLA) guarantees.
- To align with the functionality and interfaces provided by the OSM monitoring module (as per release 4), although this is still at an experimental stage.

As shown in Figure 2, the Media Service MAPE component introduces five internal services with distinct corresponding roles and responsibilities. Specifically:

- the **monitoring service** (M) collects metrics from different computing and media application environments,
the translation service (T) transforms and enriches the collected metrics by incorporating information from the Management and Orchestration (MANO) environment,

- the pre-process & analysis services (PP and A) prepares and executes ML algorithms to forecast resource demand, future network conditions and service performance,
- the planning service (P) optimises media services deployment, Virtual Network Function (VNF) placement and network operation upon the forecast received by the analysis service, and,
- the execution service (E) is responsible to communicate concrete recommendations and configuration directives to the Service Orchestrator of the OSM (through its northbound interface).

Together, the pre-process & analysis service and the execution service implement the 5G-MEDIA Cognitive Network Optimisation (CNO) engine. The pre-process & analysis service implements the Machine Learning Engine and the planning service implements the Policy/Optimisation Engine.

All these services exchange messages carrying various types of data through a brokering mechanism based on Apache Kafka¹.

The responsibility of the monitoring service is to collect data from different types/technologies of NFVI s, as well the application domain, and assess critical KPIs about the performance of the running application services and the status of the network. This information is later used by other services of MAPE or external services of the 5G-MEDIA SDK/SVP. Specifically, monitoring service is expected to collect data from the following sources:

- OpenStack²-based cloud computing environments,
- Kubernetes³ management platforms,
- OpenNebula⁴ cloud computing platforms,
- OpenWhisk⁵ FaaS framework,
- Virtual execution environments based on unikernels (Mikelangelo⁶/Jolokia), and
- The application domain (e.g. by integrating Telegraf⁷ and Apache Traffic Server⁸).

For each integrated environment a special data adaptation module is introduced to drive metrics from the corresponding environment to the 5G-MEDIA Kafka bus acting as an open message and data exchange broker in the MAPE architecture. The specification and

¹ https://kafka.apache.org/
² https://www.openstack.org/software/
³ https://kubernetes.io/
⁴ https://opennebula.org/
⁵ https://openwhisk.apache.org/
⁶ https://www.mikelangelo-project.eu/
⁷ https://www.influxdata.com/time-series-platform/telegraf/
⁸ http://trafficserver.apache.org/
implementation of each adapter depends on the type of source and the monitoring technologies/packages used. For instance, in the case of an OpenStack-based source, a Python service (called OpenStack exporter) is deployed in the monitoring service that transfers metrics from Ceilometer\(^9\) and iPerf\(^{10}\) (active) telemetry tools to the 5G-MEDIA Kafka bus. Note that all collected metrics are also stored in a database that adds persistency in the lifecycle management of the data and guarantees their retrieval by any interested service/component of the SVP in the future through the appropriate authentication mechanisms.

The translation service consumes all incoming data into the 5G-MEDIA Kafka bus and maps resource-specific metrics from the various sources to those of the Network Service/VNF as registered in the OSM domain. This way, for instance, it is possible to bind the identifier (in the form of a Universal Unique Identifier (UUID)) of the occupied Virtual Deployment Units (VDUs) (either Virtual Machines (VMs) or dockerised containers) and other resources in an NFVI environment with the instantiated VNFs and NSs that make use of them. Apart from this, the translation service is also responsible to guarantee a uniform data model over all collected metrics from every type of integrated source/environment. The translation service also can filter the incoming data considering the origin environment just in the case that the origin environment does not support it. It is highlighted that 5G-MEDIA is continuously monitoring the modifications on the OSM information model, trying to be compliant with the latest version, although OSM information model is still not in a mature state.

Pre-process & analysis services are responsible to prepare data and execute ML algorithms, respectively, towards addressing specific problems over the overall network topology, e.g. predict resource availability, network congestion or traffic/demand changes. For instance, depending on the case, a pre-processing service may aim to address aspects such as the completion of missing values, removal of outliers, parameter creation, etc. On the other side, analysis aims to extract feature vectors and carry out the core processing stages imposed by the deployed ML algorithms. Typically, the Media Service MAPE should be able to support in parallel several ML algorithms of different types, requirements and objectives. Thus, tailored, and potentially multiple, pre-process services may be onboarded per each different supported ML algorithm.

On top of the ML analysis output, the planning service orchestrates one or more optimisation algorithms to improve resource planning and assignment and produce recommendations how to improve Virtual Network Function Forwarding Graphs (VNFFGs) topologies and network configuration policies (e.g. scaling). The execution service calls the appropriate endpoints of the Service Orchestrator interface to send these recommendations to the OSM.

Figure 3 depicts in a detailed way the microservice-based components (running as docker containers) that constitute the Media Service MAPE and their interactions.

---

\(^9\) [https://docs.openstack.org/ceilometer/](https://docs.openstack.org/ceilometer/)

\(^{10}\) [https://iperf.fr/](https://iperf.fr/)
2.1. Messaging architecture

The Media Service MAPE component deploys a Kafka bus to enable fast, reliable and scalable by design data sharing and message exchange among its internal services. The bus has been installed in a separate VM in the central cloud at the project’s testbed at ENG (Engineering – Ingegneria Informatica SpA) accessible to each entity/service which has access to the private network of the core cloud. This also includes services hosted at the edge clouds.

Figure 4 shows the high-level configuration of the 5G-MEDIA Kafka bus. A Kafka cluster comprises multiple brokers/servers, each one hosting one or more partitions of the defined topics, i.e. categories where messages are published, so multiple producers and consumers can publish and retrieve messages at the same time. Brokers, producers and consumers use Zookeeper\(^{11}\) to manage state of servers and share a consistent view of messages, files and directories. Note also that the Kafka bus is configured to temporarily store all exchanged messages through the deployed brokers according to a configurable retention policy.

---

\(^{11}\) [https://zookeeper.apache.org/]
Each integrated NFVI or application-driven telemetry software acts as producer that pushes messages (e.g. about the status of resources) to a data aggregation topic, i.e. a specific topic bound to that source in one-to-one way. Thus, the number of data aggregation topics equals to the number of different sources that have been integrated to the 5G-MEDIA SVP, including both NFVI-type environments and application-based software telemetry technologies (such as telegraf). The specifications of message exchange processes within each such topic depend on the specific configuration of each producer and the implemented data adaptation procedures per case and they are thoroughly discussed in the subsections of section 3.1. Note that all these topics are also consumed by the database of the MAPE loop to permanently store collected metrics. The uniqueness of the NS-related topics is ensured by the design of the OSM since OSM database keeps unique record (UUID) per NS.

All messages entering any of the data aggregation topics are consumed by the translation service, which apart from merging information from OSM environment into each message it also homogenises them under a common data model of reference (as described in section 4). After filtering a message, the translation service pushes it to a topic that is tailored to the NS identified by the message. In other words, the architectural design is to define a separate, exclusive partition or topic per each instantiated NS, where the messages that refer to that NS are forwarded by translation service. A main advantage of this policy is that it facilitates and simplifies the metrology management for NSs which are instantiated in multiple NFVIs/PoPs sources (e.g. different subsets of VNFs are deployed in different NFVIs). In such a case, the translation service simply filters out messages retrieved from different “data aggregation” topics and routes the processed output (that combines metrics from all of them) to a common, unique destination-topic.

As mentioned before, apart from the translation service, the Kafka bus is also used by the pre-process & analysis services, the planning service and the execution service of the MAPE
loop to exchange messages carrying either metrics, metadata and/or control actions. Specifically, the messages which are produced by the pre-process service(s) are consumed by the analysis service(s), those produced by the analysis service(s) are consumed by the planning service and those produced by the planning service are consumed by the execution service. As for the translation service, exclusive topics are established for each instantiated NS (different that those used by/from the translation service) to guarantee full separation of concerns among the corresponding message exchanges processes. In Table 1, the topics in the Kafka bus are summarised. The name of the data aggregation topics is composed by the fixed prefix “nfvi” followed by the unique identifier of the specific source and its type. For instance, in the case of an OpenStack-based NFVI with unique identifier “ced93ba9-cf62-42d9-b0a6-70f211ff038”, the name of the topic is “nfvi.ced93ba9-cf62-42d9-b0a6-70f211ff038.openstack”. The unique identifier is a kind of configuration per environment. In the case of topics used/accessed by MAPE services, naming results by concatenating the unique identifier of the corresponding NS and the publishing service of MAPE as postfix (i.e. the one that produces data to the topic). Note that non-existing topics are created automatically by the Kafka bus when data is first published to it.

Table 1. Topics defined in MAPE brokering mechanism

<table>
<thead>
<tr>
<th>#</th>
<th>Topic Name</th>
<th>Producer</th>
<th>Consumer</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One Topic per each integrated environment</td>
<td>nfvi.&lt;origin_id&gt;.&lt;type&gt;</td>
<td>OpenStack environment</td>
<td>Translation service</td>
</tr>
<tr>
<td>3</td>
<td>One Topic per each instantiated NS</td>
<td>ns.&lt;ns_id&gt;.trans</td>
<td>Translation service</td>
<td>Pre-processing &amp; Analysis services</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>ns.&lt;ns_id&gt;.prep</td>
<td>Pre-processing service</td>
<td>Analysis service</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>ns.&lt;ns_id&gt;.analysis</td>
<td>Analysis service</td>
<td>Planning service</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>ns.&lt;ns_id&gt;.plan</td>
<td>Planning service</td>
<td>Execution service</td>
</tr>
</tbody>
</table>
3. Monitoring

The primary objective of the monitoring service in the Media Service MAPE is to setup a metering framework to guarantee reliable and timely metric collection from the NFVIs and the application domain. It is also responsible for the supply of data to the rest of the services of the Media Service MAPE and any other internal service of the 5G-MEDIA SVP (e.g. accounting and billing services) and the 5G-MEDIA SDK that make use of them.

As previously mentioned, the monitoring service takes advantage of the most efficient and up-to-date telemetry tools for the various NFVIs considered by the 5G-MEDIA project. It leverages a Kafka brokering mechanism, as well as specialised data adaptation and transformation services to organise a metering framework and support of services not only for the Media Service MAPE but also for those of the 5G-MEDIA SVP and SDK that leverage upon telemetry.

3.1. NFVI monitoring

The monitoring service sets up a metering framework to collect data about resources and running NSs and media applications. This data is later used by the other MAPE services, the visualisation tools of 5G-MEDIA SDK and several other internal microservices of the SVP. The implemented framework facilitates collection of measurements from various sources including numerous cloud platforms and virtualisation technologies, as well media application software monitoring tools. Clearly, each integrated source has its own technical specifications, limitations, telemetry capabilities and level of maturity which impact differently the software design of metrics collection and data adaptation processes in the scope of the metering framework. In the following subsections, the metering specifications are described for each NFVI or application domain data source.

Figure 5 depicts an indicative dashboard view that visualises Central Processing Unit (CPU), memory and network related monitoring metrics per Network Service (NS) coming from an OpenStack-based NFVI (already registered in the OSM).

![Figure 5. Visualisation of monitoring metrics per Network Service](image-url)
3.1.1. **OpenStack**

The logical flow of data collection in the OpenStack environment is shown in Figure 6. OpenStack telemetry leverages on the Ceilometer\(^{12}\) software package to collect resource status and utilisation data. In addition to the ceilometer outputs, two separate data flows are introduced to drive metrics in OSM and the MAPE loop, respectively. Specifically, on the one side, the metrics collected by the ceilometer are transferred to OSM through Gnocchi and the special OpenStack plugin implemented by its monitoring module\(^{13}\). Gnocchi\(^{14}\) is a time-series database (as a service) able to store and index metrics at a large scale as time series records, and can guarantee fast access to the data, which is desirable for NS orchestration and control in OSM.

![Figure 6. Data collection in the OpenStack NFVI](image)

Thus, in the case of OpenStack, a second data flow is implemented on top of ceilometer outputs to transparently drive metrics to the MAPE loop. Specifically, a special script exporter (implemented in Python) is deployed in OpenStack (called OpenStack Publisher) with the responsibility to receive metrics from the ceilometer (encapsulated in UDP messages) and publish them to the appropriate data aggregation topic of the 5G-MEDIA Kafka bus (the one

\(^{12}\) [https://wiki.openstack.org/wiki/Telemetry](https://wiki.openstack.org/wiki/Telemetry)


\(^{14}\) [https://wiki.openstack.org/wiki/Gnocchi](https://wiki.openstack.org/wiki/Gnocchi)
which is bound to the specific OpenStack environment). At the configuration level the interaction between ceilometer and OpenStack Publisher is achieved by the pipeline mechanism provided by the ceilometer software that couples the two entities as a source-sink pair\textsuperscript{15}. All the metrics that are aggregated by the ceilometer are transferred to the OpenStack Publisher in JSON format every a (configurable) fixed time interval (that has been set equal to 10 secs). In Figure 7, the structure of these JSON objects (that eventually are pushed in the 5G-MEDIA Kafka bus) are shown. The first attribute key (named as resource metadata) contains information about the profile of the tracked resource e.g. its flavour, status etc., while the attributes with “counter” as prefix carry the measured values (e.g. CPU time for the specific example).

\textsuperscript{15} https://docs.openstack.org/ceilometer/latest/admin/telemetry-data-pipelines.html
A complete list of metrics collected by ceilometer for OpenStack compute can be found in the Ceilometer documentation\(^\text{16}\). However, to reduce traffic burden and load of exchanged messages, the ceilometer in 5G-MEDIA is configured to collect and transfer to the Kafka bus the subset of metrics shown in Table 2. Clearly, this subset can be updated according to the

\(^{16}\) [Link to documentation](https://docs.openstack.org/ceilometer/pike/admin/telemetry-measurements.html)
requirements imposed by the deployed pre-processing & analysis services in the Media Service MAPE loop.

Table 2. Metrics collected by ceilometer in OpenStack

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Name</th>
<th>Type</th>
<th>Unit</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>memory</td>
<td>Gauge</td>
<td>MB</td>
<td>Volume of RAM allocated to the instance</td>
</tr>
<tr>
<td>2</td>
<td>memory.usage</td>
<td>Gauge</td>
<td>MB</td>
<td>Volume of RAM used by the instance from the amount of its allocated memory</td>
</tr>
<tr>
<td>3</td>
<td>memory.resident</td>
<td>Gauge</td>
<td>MB</td>
<td>Volume of RAM used by the instance on the physical machine</td>
</tr>
<tr>
<td>4</td>
<td>cpu</td>
<td>Cumulative</td>
<td>nsec</td>
<td>CPU time used</td>
</tr>
<tr>
<td>5</td>
<td>cpu_util</td>
<td>Gauge</td>
<td>%</td>
<td>Average CPU utilisation</td>
</tr>
<tr>
<td>6</td>
<td>vcpus</td>
<td>Gauge</td>
<td>vcpu</td>
<td>Number of virtual CPUs allocated to the instance</td>
</tr>
<tr>
<td>7</td>
<td>disk.read.requests.rate</td>
<td>Gauge</td>
<td>Request/sec</td>
<td>Average rate of read requests</td>
</tr>
<tr>
<td>8</td>
<td>disk.write.requests.rate</td>
<td>Gauge</td>
<td>Request/sec</td>
<td>Average rate of write requests</td>
</tr>
<tr>
<td>9</td>
<td>disk.read.bytes.rate</td>
<td>Gauge</td>
<td>B/sec</td>
<td>Average rate of reads</td>
</tr>
<tr>
<td>10</td>
<td>disk.write.bytes.rate</td>
<td>Gauge</td>
<td>B/sec</td>
<td>Average rate of writes</td>
</tr>
<tr>
<td>11</td>
<td>disk.latency</td>
<td>Gauge</td>
<td>msec</td>
<td>Average disk latency</td>
</tr>
<tr>
<td>12</td>
<td>disk.capacity</td>
<td>Gauge</td>
<td>B</td>
<td>The amount of disk that the instance can see</td>
</tr>
<tr>
<td>13</td>
<td>disk.allocation</td>
<td>Gauge</td>
<td>B</td>
<td>The amount of disk occupied by the instance on the host machine</td>
</tr>
<tr>
<td>14</td>
<td>disk.usage</td>
<td>Gauge</td>
<td>B</td>
<td>The physical size in bytes of the image container on the host</td>
</tr>
<tr>
<td>15</td>
<td>network.incoming.bytes.rate</td>
<td>Gauge</td>
<td>B/sec</td>
<td>Average rate of incoming bytes</td>
</tr>
<tr>
<td>16</td>
<td>network.outgoing.bytes.rate</td>
<td>Gauge</td>
<td>B/sec</td>
<td>Average rate of outgoing bytes</td>
</tr>
</tbody>
</table>
### 3.1.2. OpenNebula

OpenNebula provides an inherent monitoring subsystem (consisting of a set of scripts) 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. This information is collected by executing these scripts as static probes. The output of these probes is sent to OpenNebula using a push mechanism. The monitoring data can be either actively queried by OpenNebula or sent periodically by an agent running in the hosts to the publisher which pushes the data to the appropriate data aggregation topic of the 5G-MEDIA Kafka bus.
Table 3 depicts the monitoring metrics that OpenNebula provides for the Host\(^\text{17}\).

Table 3. Host information and metrics collected by OpenNebula

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Name</th>
<th>Unit</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TOTALCPU</td>
<td>-</td>
<td>Number of CPUs multiplied by 100 (for a 16 cores machine, the value will be 1600)</td>
</tr>
<tr>
<td>2</td>
<td>CPUSPEED</td>
<td>MHz</td>
<td>Speed of the CPUs</td>
</tr>
<tr>
<td>3</td>
<td>FREECPU</td>
<td>%</td>
<td>Percentage of idling CPU multiplied by the number of cores. (if 50% of the CPU is idling in a 4-core machine, the value will be 200)</td>
</tr>
<tr>
<td>4</td>
<td>USEDCPU</td>
<td>%</td>
<td>Percentage of used CPU multiplied by the number of the cores</td>
</tr>
<tr>
<td>5</td>
<td>TOTALMEMORY</td>
<td>KB</td>
<td>Maximum memory that could be used for VMs</td>
</tr>
<tr>
<td>6</td>
<td>USEDMEMORY</td>
<td>KB</td>
<td>Memory used</td>
</tr>
<tr>
<td>7</td>
<td>FREEMEMORY</td>
<td>KB</td>
<td>Available memory for VMs at the moment</td>
</tr>
<tr>
<td>8</td>
<td>NETRX</td>
<td>bytes</td>
<td>Received bytes from the network</td>
</tr>
<tr>
<td>9</td>
<td>NETTX</td>
<td>Bytes</td>
<td>Transferred bytes from the network</td>
</tr>
</tbody>
</table>

\(^{17}\) [https://docs.opennebula.org/5.4/integration/infrastructure_integration/devel-im.html](https://docs.opennebula.org/5.4/integration/infrastructure_integration/devel-im.html)
OpenNebula includes scripts that provide information about the VMs running in the host. The information about the VMs include the OpenNebula VM identifier, the hypervisor name or identifier of the VM (DEPLOY_ID), the human readable VM name (VM_NAME), the state and a set of the monitoring metrics of the VM. Table 4 depicts the set of monitoring metrics per VM that will pushed in the 5G-MEDIA Kafka bus while Table 5 presents the potential state of each VM.

### Table 4. VM monitoring metrics collected by OpenNebula

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Name</th>
<th>Unit</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU</td>
<td>%</td>
<td>Percentage of a single CPU consumed (e.g. for two fully consumed CPUs, the value is 200)</td>
</tr>
<tr>
<td>2</td>
<td>MEMORY</td>
<td>KB</td>
<td>Memory consumption</td>
</tr>
<tr>
<td>3</td>
<td>NETRX</td>
<td>B</td>
<td>Received bytes from the network</td>
</tr>
<tr>
<td>4</td>
<td>NETTX</td>
<td>B</td>
<td>Sent bytes to the network</td>
</tr>
<tr>
<td>5</td>
<td>DISKRDBYTES</td>
<td>B</td>
<td>Read bytes from all disks since last VM start</td>
</tr>
<tr>
<td>6</td>
<td>DISKWWRBYTES</td>
<td>B</td>
<td>Written bytes from all disks since last VM start</td>
</tr>
<tr>
<td>7</td>
<td>DISKRDIOPS</td>
<td>-</td>
<td>Read IO operations from all disks since last VM start</td>
</tr>
<tr>
<td>8</td>
<td>DISKWRIOOPS</td>
<td>-</td>
<td>Written IO operations all disks since last VM start</td>
</tr>
<tr>
<td>9</td>
<td>DISK_SIZE</td>
<td>MB</td>
<td>Vector attribute two sub-attributes: ID: identifier of the disk, and SIZE: real size of the disk</td>
</tr>
<tr>
<td>10</td>
<td>SNAPSHOT_SIZE</td>
<td>MB</td>
<td>Vector attribute with two sub-attributes: ID: identifier of the snapshot, DISK_ID: identifier of the disk, and SIZE: real size of the snapshot</td>
</tr>
</tbody>
</table>

### Table 5. List of VM states

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Detecting error state</td>
<td>The monitoring could be done, but it returned an unexpected output.</td>
</tr>
<tr>
<td>a</td>
<td>Active</td>
<td>The VM alive (running, blocked, booting...). The VM will be set to RUNNING</td>
</tr>
<tr>
<td>p</td>
<td>Paused</td>
<td>The VM will be set to SUSPENDED</td>
</tr>
<tr>
<td>e</td>
<td>Error</td>
<td>The VM crashed or somehow its deployment failed. The VM will be set to UNKNOWN</td>
</tr>
</tbody>
</table>

---

18 [https://docs.opennebula.org/5.4/integration/infrastructure_integration/devel-vmm.html#devel-vmm](https://docs.opennebula.org/5.4/integration/infrastructure_integration/devel-vmm.html#devel-vmm)
d | Disappeared | VM not known by the hypervisor anymore. The VM will be set to POWEROFF

Figure 9 depicts an indicative output of the monitoring subsystem that OpenNebula supports for a set of VMs in the same host.

```
VM $FOLL$-Y25
VM=[
  ID=115,
  DEPLOY_ID=one-115,
  VM_NAME=one-115,
  IMPORT_TEMPLATE="TxFRTJ3vGhUfN/cqXwJhXeKo13fJbNwQ0iEFNSyNfY5wJrX0",
  POLL="STATE=ON CPU=100.0 MEMORY=75232 NETTX=1518 NETRX=0 DISK_SIZE( ID=0, SIZE=24)
SNAPSHOT_SIZE=[ ID=1, DISK_ID=0, SIZE=24]"
]
VM=[
  ID=116,
  DEPLOY_ID=one-116,
  VM_NAME=one-116,
  IMPORT_TEMPLATE="TxFRTJ3vGhUfN/cqXwJhXeKo13fJbNwQ0iEFNSyNfY5wJrX0",
  POLL="STATE=ON CPU=100.5 MEMORY=77224 NETTX=1392 NETRX=0 DISK_SIZE( ID=0, SIZE=24)
DISK_SIZE=[ ID=1, SIZE=0]"
]
```

Figure 9. Indicative output of the OpenNebula monitoring system for a set of VMs

### 3.1.3. Telegraf and Logstash integrated with Elasticsearch and Kibana

In addition to OpenStack’s infrastructure monitoring mechanisms, the media applications’ monitoring for standard VNFs (i.e. non FaaS) can be realised in 5G-MEDIA through the use of monitoring agents like Influxdata Telegraf\(^\text{19}\) and the Elastic Logstash\(^\text{20}\), activated on the VM to collect various statistics of the application. For example, in the vCache VNF\(^\text{21}\) based on Apache Traffic Server\(^\text{22}\) our choice has been to integrate two different monitoring data collectors in the VNF to serve different type of monitoring data sources and scopes. Logstash is used to collect Apache Traffic Server application logs and events, natively integrated with the caching software release; and Telegraf is used to collecting more generic compute and network metrics of the VM together with more specific cache performance metrics (e.g. cache hits, throughput, etc.) exposed by ATS via REST APIs. The chosen solution allows the implementation of a flexible environment for monitoring the vCache VNF which can easily be adapted to a wider range of metrics/statistics to be offered to the 5G-MEDIA MAPE for further analysis and decisions on service optimisation.

Both Influxdata Telegraf and Elastic Logstash are plugin-driven monitoring agents capable of collecting, processing and reporting different kinds of metrics. They provide several input

---

\(^{19}\) https://www.influxdata.com/time-series-platform/telegraf/

\(^{20}\) https://www.elastic.co/products/logstash

\(^{21}\) See 5G-MEDIA deliverable D4.1 for a detailed specification of the vCache VNF.

\(^{22}\) http://trafficserver.apache.org/
plugins that work in parallel for retrieving data, for instance, from databases (e.g. CouchDB, Elasticsearch, InfluxDB etc.), web servers (e.g. NGNIX, Apache HTTP Server), queues (e.g. Kafka, RabbitMQ) or via standardised protocols (e.g. HTTP, SNMP). In the same way, they provide different output plugins that allow to dispatch the collected metrics to different entities, such as databases, queues and tools.

Telegraf works with several data formats, input plugins are able to parse InfluxDB Line Protocol, JSON, Graphite, Value, Nagios, Collectd and Dropwizard dataset, while output data can be structured as JSON, Graphite and InfluxDB Line Protocol entries. As mentioned before, Logstash is able to parse logs, and more in general files, according to specified line formats to be defined through proper patterns depending on the application’s log structure, but it’s also capable of collecting metrics through application plugins. In Logstash, filter plugins, such as Grok, JSON and CVS plugins, are responsible for defining proper patterns for extracting values. Furthermore, input and output data representation can be encoded using codec plugins. Logstash offers several codecs including JSON and Graphite.

Elasticsearch is a distributed time series data base, which provides RESTful APIs for easily integrating clients based on different programming languages and offering fast searching based on indices. Kibana is a web graphical user interface that allows to visualise, aggregate and analyse data stored in Elasticsearch through the creation of personalised dashboards. Kibana’s dashboards can include histograms, line graphs, pie charts, sunbursts etc. for shaping data according to the user’s needs. Logstash, Elasticsearch and Kibana are components of the

---

23 https://www.elastic.co/products/elasticsearch
Elastic Stack architecture, which together realised a complete monitoring stack, as depicted in Figure 10:

In the Elasticsearch, Logstash and Kibana\(^{24}\) (ELK) stack we have integrated also the Telegraf agent as an alternative to Logstash (see Figure 11). Indeed, Telegraf provides an output plugin that targets the Elasticsearch data base, allowing to push data to an index that can be created runtime on Telegraf’s bootstrap.

The kinds of metrics collected through the presented application monitoring stack are dependent on the application itself and, more specifically, on what the application is able to

\(^{24}\) [https://www.elastic.co/products/kibana](https://www.elastic.co/products/kibana)
expose through logs, APIs, command line tools etc. In 5G-MEDIA an early integration of the application monitoring stack was actuated on top of vCache VNF developed in the context of use case 3. The vCache implementation, as already mentioned, is based on the Apache Traffic Server and both Telegraf and Logstash were integrated as monitoring agents. Telegraf is used for collecting both net statistics at the VM level and application’s metrics such as the throughput and the cache’s hits, while Logstash is collecting logs relevant events (e.g. errors). The following Figure 12 presents an example of metrics collected through Telegraf on a vCache instance.

```
# NET STATS
{
  "fields": {
    "bytes_recv": 183387273,
    "bytes_sent": 304161205,
    "drop_in": 0,
    "drop_out": 0,
    "err_in": 0,
    "err_out": 0,
    "packets_recv": 1108140,
    "packets_sent": 910064
  },
  "name": "net",
  "tags": {
    "host": "uhdocdn-100851",
    "interface": "eth0"
  },
  "timestamp": 1527165350
}
```

*Figure 12. Network statistics collected via Telegraf on the vCache VM*

### 3.1.4. Unikernels

For VNFs implemented using unikernel VDUs, monitoring can be achieved using the results of EU project Mikelangelo. This provides an API to expose a large quantity of information on the current unikernel VM status, including information on its threads, files, memory and network configuration. The data model is described in the “REST API” section of the Universal Unikernel OSv documentation. Depending on the actual monitoring needs, for example, a simple check could periodically monitor Operating System memory total and free, the dmesg content, the uptime, etc.

---

25 See 5G-MEDIA deliverable D6.1 for a detailed description of use case 3: UHD CDN.
26 [https://www.mikelangelo-project.eu/](https://www.mikelangelo-project.eu/)
27 [https://www.mikelangelo-project.eu/technology/universal-unikernel-osv/](https://www.mikelangelo-project.eu/technology/universal-unikernel-osv/)
28 [http://www.linfo.org/dmesg.html](http://www.linfo.org/dmesg.html)
On top of such unikernels VDUs, the application logic can be deployed with some constraints (given that due to the unikernel concept, only one process but multiple threads model can be executed) and, with them, the possible inclusion of a monitoring component that is application-dependent and allows to publish the relevant data model for each VNF.

For this second monitoring level, there is a strong dependency on the application language used and the logic itself, so, apart from the single process constraint and the possible compilation of libraries needed for the monitoring, this process is exactly the same as for a plain non-unikernel packaging.

From the perspective of Java development though, Mikelangelo provides JDKs that already incorporate Jolokia\(^{29}\) technology that supports pub/sub and plain data retrieval of the general purpose JMX\(^{30}\) data model. For the other programming languages, nothing similar is provided, so from the perspective of a developer, some specific libraries should be added to the unikernel VDU to get similar features.

For Java development, JMX is the standard de facto for the monitoring of application layer data. By simply using standard annotations (such as “@MBean”) on top of a Java Bean class file, its data is automatically published in JMX console and, with Jolokia, offered through a REST API on HTTP/HTTPS.

Together with the specific MBeans\(^{31}\) annotated by the developers, JMX is also used by the JDK itself to publish data about its internal status that could be used for monitoring as well.

For example, a typical use case is to keep track of the memory heap size to avoid the “java.lang.OutOfMemory” error; such data is published by the attribute “HeapMemoryUsage” of the “java.lang.Memory” MBean and is available on Jolokia and the relative URL “/jolokia/read/java.lang:type=Memory/HeapMemoryUsage”. The complete data model is described in the dedicated Jolokia web page\(^{32}\).

Additionally, a 5G-MEDIA developer can use the SDK to automatically setup the unikernel project, so that he/she can use JMX technology just like in plain Java development environment and have its data model published as REST API in a complete transparent way.

The data provided so far is published as REST API and retrieved by the 5G-MEDIA monitoring component. While the part related to unikernel VDU does not adopt any standard protocol, Jolokia is instead a well-known monitoring protocol, so many options are available to simplify the extraction and loading for further elaboration, such as Telegraf. While it usually adopts an agent-based approach that is installed on the application to be monitored for local extraction, in case of unikernels and due to their limitation in terms of single processing, an external Telegraf agent could be used to extract data from Jolokia\(^{33}\), but the process would remain

\(^{29}\) https://jolokia.org/
\(^{30}\) http://www.oracle.com/technetwork/articles/java/javamanagement-140525.html
\(^{31}\) https://docs.oracle.com/javase/tutorial/jmx/mbeans/standard.html
\(^{33}\) https://github.com/influxdata/telegraf/tree/master/plugins/inputs/jolokia2
exactly the same. For the data extraction and processing, please refer to the dedicated section in this document.

3.1.5. **FaaS/Kubernetes**

The monitoring architecture for Apache OpenWhisk running functions on top of Kubernetes is described in detail in Section 3.7 of 5G-MEDIA deliverable D3.2.

3.2. **Network monitoring**

Passive network monitoring is a process that collects information with respect to network performance metrics by utilising applications that do not actively intervene with the network traffic. This type of monitoring helps to determine packet flows and allows the study of traffic patterns. In this perspective, passive monitoring is helpful when network administrator needs to know details such as network topology, running services, ports used frequently, etc. However, there are plenty of cases where active network monitoring is required. This type of monitoring demands probes and transmits packets between two endpoints. This can interfere with the regular traffic and while this might be a problem in some contexts, active network monitoring allows full control over additional packets that are required to be sent over the network, as they can be sent whenever required by any specific monitoring application. Hence, in some cases active monitoring can be a flexible tool to provide useful insights on the network characteristics.

Passive network monitoring involves accessing the underlying network equipment, via the northbound interface of the network operator’s SDN controller or other network management system. The project plans to investigate the options and mechanisms for achieving this in the project’s testbeds in the second year of the project during the next iteration of the three use case scenarios. An update to the project’s MAPE architecture which includes interfaces to the SDN controller and/or network management system of the network provider will be provided in deliverable D3.4 at the end of the project. In the meantime, the active monitoring techniques deployed by the project are described in the following subsections.

3.2.1. **Active network monitoring probes and metrics**

5G-MEDIA active monitoring probes have been designed according to a client/server paradigm. The implemented probes provide three different measurements, namely Round-Trip-Time (RTT), Jitter and Bandwidth.

Active network monitoring service consists of the following components and functionalities:

- **Simple RESTful API**
  The service provides a set of simple API methods, utilising the Flask-RESTful library\(^{34}\) which is an extension for Python Flask that adds support for quickly building REST APIs. It is a lightweight abstraction that works with several libraries and encourages best

\(^{34}\) [https://flask-restful.readthedocs.io/](https://flask-restful.readthedocs.io/)
practices with minimal setup. This API is used to initiate the tests, but can be further extended to cover any future demand of the service during the lifetime of 5G-MEDIA project.

- **ICMP Ping3 tool**
  For the realisation of ping measurements, we utilised Ping3\(^{35}\), which is a pure Python3 version of the Internet Control Message Protocol (ICMP) ping implementation using raw socket. The tool is written in Python to be easily integrated with the rest of the tools.

- **IPerf3 tool**
  The IPerf\(^{36}\) is a series of tools that perform active measurements to determine the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, protocols, and buffers. For each test it reports the measured throughput, loss, and other parameters. The version iperf3 is a redesign of an original version with the goal of a smaller, simpler code base, and a library version of the functionality that can be used in other programs.

- **Gnocchi client**
  As the implementation of 5G-MEDIA is utilising Gnocchi, being also part of the OSM monitoring implementation, there is a need for the installation and integration of the Gnocchi client. This client provides the Python bindings and command line tool to communicate with the Gnocchi API.

### 3.2.2. Provided functionalities and utilisation examples

A general diagram of the active monitoring process includes one or more VNFs. Each of the VNFs include one port for the incoming traffic, one port for the outgoing traffic and one management port.

![Diagram of active monitoring on deployed Network Services](https://pypi.org/project/ping3/)

![Diagram of active monitoring on deployed Network Services](https://iperf.fr/)

---

35 [https://pypi.org/project/ping3/](https://pypi.org/project/ping3/)
36 [https://iperf.fr/](https://iperf.fr/)
3.2.2.1. Measure RTT from VNF1 to VNF2

The current implementation provides the utilisation of a method, through the management port, to measure RTT between two VNFs. The result returns information on the RTT measurement as shown below, where relevant information must be inserted as the IP address and the number of packets.

```
curl http://<vnf1_management_ip>:5002/activeMon/rtt/pkts/2/dest/<vnf2_public_ip>
```

```
{
    "avg": "2.086",
    "max": "2.890",
    "mdev": "0.804",
    "min": "1.283",
    "num_pkts": 2,
    "status": "OK"
}
```

3.2.2.2. Measure bandwidth, packet loss and jitter between VMs

Additionally, the service can instantiate the command to execute the test for the measurement of bandwidth, packet loss and jitter, as shown below.

```
curl http://<vnf1_management_ip>:5002/activeMon/bandwith/duration/1/dest/<vnf2_public_ip>/port/6969
```

```
{
    "bandwidth": 13392.2,
    "duration": 1,
    "jitter": 1.04346,
    "pkt_loss": 88.7767,
    "seconds": 1.0002,
    "status": "OK"
}
```

3.2.2.3. Retrieve data from Gnocchi

By utilising the Gnocchi client, the user can retrieve data as demonstrated in the following example commands.

```
gnocchi resource list -c id -c type
```

```
+----------------------------------------+------------------+
| id                                    | type             |
+----------------------------------------+------------------+
| 88916e95-4a4a-49a4-8472-80bb7c78ecb1   | generic          |
| 75c44741-cc60-4033-804e-2d3098c7d2e9    | active_network_monitoring |
+----------------------------------------+------------------+
```
gnocchi resource show 75c44741-cc60-4033-804e-2d3098c7d2e9

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>created_by_project_id</td>
<td>e579eb5a982c4c559a777886754da9e</td>
</tr>
<tr>
<td>created_by_user_id</td>
<td>1c51d7a519b94345845287b8d7d6f6</td>
</tr>
<tr>
<td>ended_at</td>
<td>None</td>
</tr>
<tr>
<td>id</td>
<td>75c44741-cc60-4033-804e-2d3098c7d2e9</td>
</tr>
<tr>
<td>metrics</td>
<td></td>
</tr>
<tr>
<td>bandwidth</td>
<td>c2e4770c-c6c7-4d68-b360-177f47249a22</td>
</tr>
<tr>
<td>jitter</td>
<td>ce39ec56-3f0e-42fe-bd67-ff5959b6e49b</td>
</tr>
<tr>
<td>started_at</td>
<td>2018-03-02T09:35:20.033313+00:00</td>
</tr>
<tr>
<td>type</td>
<td>active_network_monitoring</td>
</tr>
<tr>
<td>user_id</td>
<td>6526f183-b745-401c-a0bd-b5a22ed1b14-172.24.4.2</td>
</tr>
</tbody>
</table>

Metric values can be retrieved with the following commands.

- For RTT measurements:
  
gnocchi measures show rtt --resource-id 75c44741-cc60-4033-804e-2d3098c7d2e9

<table>
<thead>
<tr>
<th>timestamp</th>
<th>granularity</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-02T09:00:00+00:00</td>
<td>3600.0</td>
<td>2.086</td>
</tr>
<tr>
<td>2018-03-02T09:35:00+00:00</td>
<td>60.0</td>
<td>2.086</td>
</tr>
<tr>
<td>2018-03-02T09:35:24+00:00</td>
<td>1.0</td>
<td>2.086</td>
</tr>
</tbody>
</table>

- For jitter measurements:
  
gnocchi measures show jitter --resource-id 75c44741-cc60-4033-804e-2d3098c7d2e9

<table>
<thead>
<tr>
<th>timestamp</th>
<th>granularity</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-02T09:00:00+00:00</td>
<td>3600.0</td>
<td>1.04346</td>
</tr>
<tr>
<td>2018-03-02T09:36:00+00:00</td>
<td>60.0</td>
<td>1.04346</td>
</tr>
<tr>
<td>2018-03-02T09:36:57+00:00</td>
<td>1.0</td>
<td>1.04346</td>
</tr>
</tbody>
</table>

- For bandwidth measurements:
  
gnocchi measures show bandwidth --resource-id 75c44741-cc60-4033-804e-2d3098c7d2e9

<table>
<thead>
<tr>
<th>timestamp</th>
<th>granularity</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-02T09:00:00+00:00</td>
<td>3600.0</td>
<td>13392.2</td>
</tr>
<tr>
<td>2018-03-02T09:36:00+00:00</td>
<td>60.0</td>
<td>13392.2</td>
</tr>
<tr>
<td>2018-03-02T09:36:57+00:00</td>
<td>1.0</td>
<td>13392.2</td>
</tr>
</tbody>
</table>
4. **Translation**

The translation service of Media Service MAPE consumes all messages injected into any of the data aggregation topics of the 5G-MEDIA Kafka bus. Typically, these messages carry metrics about the status and performance of one or more computing and networking resources in the corresponding integrated environment (i.e. the environment which is bound to the specific topic). The role of the translation service is to enrich this information with data and metadata of NSs/VNFs orchestration, as this is retrieved by the API of the Service Orchestrator of the OSM. Specifically, upon the reception of a message, the translation service makes use of the following endpoint of OSM Service Orchestrator API to retrieve UUIDs of NSs and VNFs that make use of the resource(s) which is included in the message:

https://{osm-so-ipv4}:{port}/v1/api/operational/project/default/vnfr-catalog/vnf

Another responsibility of the translation service is to guarantee a common and unified data model for the messages it produces. This is a critical task given the different software modules, data models and data adaptation mechanisms used by the various types of integrated sources to send data to the Kafka bus. Thus, the mission of translation service is not only to preserve all useful information included in the messages entering the data aggregation topics but also to restructure and provide it back in a homogenised way. Specifically, the attributes included in the implemented data model (by the translation service) contains the information shown in Table 6.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 vim</td>
<td>Name, type and IP of the corresponding VIM</td>
</tr>
<tr>
<td>2 mano</td>
<td>VDU: profile of the resource in reference, including VDU identifier, flavour, image identifier and status VNF: identifier of VNF running on the resource in reference NS: identifier of NS running on the resource in reference</td>
</tr>
<tr>
<td>3 metric</td>
<td>Name, value, timestamp, and type of the specific metric</td>
</tr>
</tbody>
</table>

As an example, in Figure 14, the JSON object is shown which is produced by the translation service when the metric in Figure 7 is filtered out.
Figure 14. Example message produced by the translation service of MAPE
5. Pre-process & Analysis

The pre-process & analysis functionality is concerned with converting raw monitored data into useful statistics on past performance and future predictions on demand, and resource performance. The two broad techniques to be deployed are demand forecasting and anomaly detection. The results of the pre-process & analysis logic are used by the planning component to optimise the network service according to predictions on load, resource usage and NS performance.

Network traffic and resource consumption prediction using machine learning has been attracting a lot of attention recently.\textsuperscript{37, 38, 39, 40} The CogNet project\textsuperscript{41} developed a set of tools for autonomic network management based on traffic prediction using machine learning techniques. In general, several methods have been proposed in the literature for traffic forecasting which can be classified into two categories: linear prediction and non-linear prediction. Results\textsuperscript{42} show that non-linear prediction based on Neural Networks (NN) outperforms linear forecasting models. Recurrent Neural Networks (RNN) have been shown to be suited for sequence modelling tasks (e.g. traffic matrix prediction) as they have temporal memory.\textsuperscript{37} An overview of ML approaches for forecasting demand and resource consumption is provided in the following.

\begin{itemize}
\item[40] Cognet project: http://www.cognet.5g-ppp.eu
\end{itemize}
5.1. Machine Learning approaches for forecasting demand and resource consumption

5.1.1. Feed forward neural networks

Supervised learning is a Machine Learning task that consists of using a set of input-output pairs \((x_i, y_i)\) to try to determine a model \(f(x) = y\) that can accurately predict an output \(y^*\) from an input \(x^*\). There are several different models and algorithms such as Logistic Regression, Decision Trees\(^{43}\) or Support Vector Machines\(^{44}\). A particularly successful category is Artificial Neural Networks (ANN), a class of Machine Learning models that have been used in many different fields showing a very good generalisation performance. The most basic ANN are the Feed-Forward or Dense Neural Networks (DNN) which are characterised by computing an input across several layers, each layer having an activation function and a number of neurons. The first layer \(l_1\) computes \(l_1 = a_1(W_1x + b_1)\) and the output of each subsequent layer \(k\) of a DNN is given by \(l_k = a_k(W_kl_{k-1} + b_k)\), where \(W_k, b_k\) and \(a_k\) are respectively the weights, biases and activation function of layer \(k\). In practice, training an ANN involves finding the values for \(W\) and \(b\) that minimise a given loss function. A common way to train ANNs is through a gradient descent optimisation algorithm where the gradient is found through the back-propagation algorithm\(^{45}\).

---


5.1.2. Recurrent Neural Network

![Recurrent Neural Network](image)

Figure 16. Recurrent neural network (RNN)

Recurrent Neural Networks (RNN) are a type of ANN where each layer’s output depends not only on the previous layer’s values but also on weights $W$ and biases which share a dependency between parts of the input. Neurons are fed information not just from the previous layer but also from themselves from the previous pass. This makes them suited to tasks where the inputs show a time dependence/order, such as time series data, words in a text and frames of a video, which would perfectly suit to the needs of the 5G-MEDIA use cases scenarios.

5.1.3. Long Short-Term Memory

![Long Short-Term Memory](image)

Figure 17. Long short term memory (LSTM)

In Long Short-Term Memory (LSTM), each neuron has a memory cell and three gates: input, output and forget. The function of these gates is to safeguard the information by stopping or allowing the flow of it. The input gate determines how much of the information from the previous layer gets stored in the cell. The output layer takes the job on the other end and determines how much of the next layer gets to know about the state of this cell. LSTMs have been shown to be able to model complex sequences, such as written prose or music compositions.

---


5.2. Pre-process & analysis algorithms in 5G-MEDIA

This section describes the initial algorithms designed for the pre-process & analysis service in the context of use case 3\textsuperscript{48}. Figure 18 shows the normal working state of use case 3: mobile users receive video streams from the vCaches or from the OriginServer based on the availability of the content. A machine learning algorithm is used to predict the traffic conditions on the link embedded with the vNetEm VNF.

![Figure 18. Normal working state of the UHD CDN in UC3](image)

When congestion is predicted by the ML algorithms, which would cause video quality degradation for the users receiving video streams from that instance of the vCache, the pre-process & analysis service will trigger the planning service to initiate the scale-out of the instantiated NS as shown in Figure 19.

![Figure 19. Scaling out of the UHD CDN NS to deal with forecasted network congestion](image)

\textsuperscript{48} See 5G-MEDIA deliverables D2.2 and D6.1 for a comprehensive description of the UHD CDN scenario in use case 3 and its deployment plans in the project testbeds.
5.2.1. **ML algorithms for initial implementation of the pre-process & analysis service**

Machine learning algorithms can respond to a condition when it has been sufficiently trained to recognise the condition in advance. In our case we wish to identify when anomalous load conditions are likely to cause congestion on the link between the virtual cache and the users and are likely to degrade the quality of the delivered video streams. The traffic forecasting algorithm requires extensive training with traffic traces to operate optimally in a wide range of network conditions. In other words, data sets used for training phase of the machine learning algorithm should cover diverse set of network conditions; various traffic loads over variety of network settings (e.g. various link capacities, queue lengths, etc.).

As the project’s testbeds are small in scale and do not have a large number of users generating background traffic to cause congestion and quality degradations, we need to make use of traffic generation techniques. A set of traffic traces are needed to train and test the ML algorithms. We are making use of both real and artificial traffic traces, as described in the following.

5.2.1.1. **Real traffic traces**

Two sources of real traffic datasets have been selected for the initial training and testing of the ML algorithms:

- a large traffic data repository where the traffic traces are captured from two transit links maintained by the MAWI Working Group of the WIDE Project (in Japan)\(^49\)
- a traffic data repository comprising the traffic traces collected from backbone links by the Center for Applied Internet Data Analysis (CAIDA) institute.\(^50\)

The repository in the WIDE project includes daily and weekly traffic captures from two sample points. The first is a high-speed link with the data rate of 1Gbps, connecting the WIDE network to upstream Internet Service Provider (ISP); this is where traffic flows to the internet from the WIDE network. The second is the main Internet Exchange Point (IXP) of the WIDE network. Older traces since year 1999 from different sample points are also publicly available. These old traces are highly beneficial because they are collected from links with varied capacity. The traffic traces are captured via tcpdump and stored in a pcap file. Each file includes 15 minutes’ worth of traffic capture.

CAIDA maintains Internet data collection monitors at two datacentres (Chicago and San Jose). The monitor in Chicago is connected to a backbone link of a Tier1 ISP between Chicago and Seattle. The monitor in San Jose is connected to a backbone link of a Tier1 ISP between San Jose and Los Angeles. CAIDA collect a one-hour traffic trace at each of the monitors once a month. The repository includes traffic traces since 2008.

---


\(^{50}\) [http://www.caida.org/data/](http://www.caida.org/data/)
5.2.1.2. Artificial traffic traces

For controlled testing and tuning of the algorithms we are also using artificially generated traffic traces. We consider two kinds of traffic: (background + video) and anomaly traffic. For testing, the (background + video) traffic is created in which varies between 50 Mbps and 90 Mbps (Figure 20).

![Figure 20. Trace of background + video traffic.](image)

In Figure 20, we show the trace of (background + video) traffic with 10,000 data points. We assume that those data is collected by the monitoring system and stored in a history log file. The link capacity is 100 Mbps and the (background + video) traffic does not cause congestion. Thus, we introduce anomaly traffic to increase the load to force network congestion.

![Figure 21. Trace of anomaly traffic.](image)

Figure 21 shows sample of 500 data points of the anomaly traffic in which the volume is in [0, 30] Mbps. In most cases, the anomaly traffic only contributes up to 10 Mbps to the total load. However, there are some points where the anomaly traffic increases to 30 Mbps. The anomaly traffic may cause congestion when combined with the (background + video) traffic and the total load is in excess of 100 Mbps.
The monitoring system does not see different kinds of traffic. Instead, it only sees the aggregate traffic in which traffic volume is the sum of (background + video) and anomaly traffic. Given the (background + video) and the anomaly traffic in Figure 20 and Figure 21, we show in Figure 22 the aggregate traffic collected by the monitoring system.

![Figure 22. Aggregate traffic trace.](image)

In Figure 22, we can see that there are some points where the total load is greater than 100 Mbps. As an example of congestion, if there are more than 5 continuous points (this value is a parameter of the machine learning algorithm and can be set at run time) in which their values are over 100 Mbps, we consider the network is congested.

We need to design a machine learning algorithm to predict when the congestion will happen in the near future in order to trigger the scale-out of the instantiated NS. In more detail, we consider a moving window size of 10 points of the aggregate traffic. We look ahead 90 points to see whether congestion happen in the future. Again, those values (moving window size and look-ahead window) are parameters of the algorithm. The machine learning algorithm will be trained to identify traffic patterns within the moving window that characterise future congestion conditions to decide whether or not to scale-out the network service.

### 5.2.1.3. Initial implementation

The pre-process & analysis service makes use of Apache Spark\(^{51}\) for large-scale data processing and the Spark library of machine learning tools, MLlib.\(^{52}\) We have developed the initial ML algorithms by integrating into this environment:

- **TensorFlow** - a dataflow-based deep-learning software package developed by Google\(^{53}\). TensorFlow has already implemented RNN.
- **Keras** - a high-level neural networks API\(^{54}\) running on top of TensorFlow to build and train the model.

---

\(^{51}\) Apache Spark. [https://spark.apache.org/](https://spark.apache.org/)

\(^{52}\) [https://spark.apache.org/mlib/](https://spark.apache.org/mlib/)

\(^{53}\) TensorFlow. [https://www.tensorflow.org/](https://www.tensorflow.org/)

\(^{54}\) Keras library. [https://keras.io](https://keras.io)
We use a fully connected neural network with one hidden layer. There are 10 nodes in the hidden layer as shown in Figure 23.

5.2.1.4. Procedure to update network weights

We use Adam (tf.train.AdamOptimizer) as an optimisation algorithm to update network weights iteratively based on training data. Adam offers several advantages over the classical stochastic gradient descent method (tf.train.GradientDescentOptimizer). For instance, it uses moving averages of the parameters to enable using a larger effective step size, and the algorithm will converge to this step size without fine tuning.

We set parameters for the Adam method as follows:

- Learning_rate = 0.001,
- Beta1 = 0.9,
- Beta2 = 0.999,
- Epsilon = 1e-08,

Where:

- Learning_rate or step size is a floating point value to determine the proportion that weights are updated.
- Beta1 is a float value or a constant float tensor. It is the exponential decay rate for the 1st moment estimates.
- Beta2 is a float value or a constant float tensor. It is the exponential decay rate for the 2nd moment estimates.
- Epsilon is a small constant for numerical stability.

5.2.1.5. Activation function

An activation function is used to convert an input of a node in a neural network to an output signal. This output signal is then used as an input for the next layer stack. For the implementation of the CNO, we use ReLU (Rectified Linear Unit) as the activation function.

---

ReLU is used by the majority of deep learning models. The idea behind ReLU is quite simple: $R(x) = \max(0, x)$ i.e. if $x < 0$, $R(x) = 0$ and if $x \geq 0$, $R(x) = x$ (Figure 24). Despite its simplicity, it has been shown that ReLU has 6 times improvement in convergence compared to the Tanh function\(^{56}\).

![ReLU activation function](image)

**Figure 24. ReLU activation function**

We train the model with 201 epochs using aggregate traffic traces, such as the one shown in Figure 22. We then validate the model using a testing traffic trace, as shown in Figure 25. We consider training the machine-learning model with absolute traffic volume values and with delta values (the difference between two consecutive points).

![Aggregate traffic trace](image)

**Figure 25. Aggregate traffic trace for testing ML model.**

\(^{56}\) [https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f](https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f)
5.2.1.6. Training with absolute traffic volume values:

```
88.0, 98.0, 98.0, 98.0, 92.0, 94.0, 56.0, 96.0, 94.0, 92.0, 0
90.0, 98.0, 98.0, 98.0, 94.0, 96.0, 96.0, 94.0, 92.0, 92.0, 0
98.0, 98.0, 92.0, 94.0, 96.0, 96.0, 94.0, 92.0, 92.0, 92.0, 0
98.0, 98.0, 92.0, 94.0, 96.0, 96.0, 94.0, 92.0, 92.0, 92.0, 0
92.0, 94.0, 96.0, 96.0, 94.0, 92.0, 92.0, 92.0, 92.0, 92.0, 1
94.0, 98.0, 98.0, 94.0, 94.0, 92.0, 92.0, 92.0, 92.0, 92.0, 1
96.0, 98.0, 94.0, 92.0, 92.0, 92.0, 92.0, 92.0, 92.0, 92.0, 1
99.0, 94.0, 92.0, 92.0, 92.0, 92.0, 92.0, 92.0, 92.0, 92.0, 0
```

Figure 26. Samples of training data with absolute values

Each row of the training data set contains 11 columns. The first ten columns are the ten consecutive data points while the last column which is “1” or “0” classifies whether the network was congested in the near future (we use look-ahead of 90 data points). However, from our initial tests we found that this type of training data contained significant noise, which prevented the ML model to give accurate predictions with a reasonable amount of training time. The test set accuracy we measured was between 75% and 79%. For this reason, we decided to use training data with delta values to remove noise and improve prediction performance as follows.

5.2.1.7. Training with delta values

```
88.0, 2.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, -2.0, -2.0, 0
90.0, 2.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, -2.0, -2.0, 0
98.0, 2.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, -2.0, -2.0, 0
98.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
92.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
94.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
96.0, 0.0, 0.0, -2.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
98.0, -2.0, -2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
```

Figure 27. Samples of training data with delta values

The first column in Figure 27 is the absolute value as a baseline and the following 9 columns give the delta from the previous data point. The last column has a binary value to determine if the network will be congested in near future or not. The results with this training data were significantly improved, giving a test set accuracy of around 93%.

5.3. Ongoing work on ML algorithms for the pre-process & analysis service

In this section we have described our initial implementation of a neural network for anomaly detection and traffic forecasting. The next steps are to complete the training and testing of the algorithms in the project testbeds. We would like to detect a wide range of traffic anomalies and the collection of real-world datasets is an important prerequisite to training the ML algorithms under a large number of conditions. The current implementation is focussed on the detection of congestion on a single link, the next iteration of the algorithms will have a wider scope taking in metrics from multiple network links as well as the computational load on VNFs forming the NSs. Inputs from a wider variety of data sources will allow ML algorithms to detect conditions that arise through a combination of load and utilisation metrics collected from a distributed set of resources. Although this will enable more general forecasts to be made, this will be at the expense of longer and more complex training phases. The use of reinforcement learning and unsupervised learning will be further explored.
for such cases. The suitability of different ML techniques for forecasting and anomaly detection in these more complex scenarios will be reported in D3.4.
6. Planning

The planning service takes input from the pre-process & analysis service about the current status and the predicted usage of network and application resources for the instantiated NSs and executes algorithms to configure and adapt the NS, the infrastructure on which it runs and the application logic. Algorithm behaviour is tailored by operator-defined policies, e.g. the service provider can define the maximum cost budget or the lowest acceptable performance level of the NS to meet SLA objectives. The output of the optimisation algorithms is formed into a set of operations forwarded on the Kafka bus which are subsequently read and implemented by the execution service.

Planning algorithms for the optimisation of media services in the three 5G-MEDIA use cases consist of four main types:

- Service placement optimisation to determine which NFVI instance/edge node should house each VNF for a NS by trading-off cost with performance of the network and computational infrastructure. This can run at various timescales, including initial NS deployment and ongoing reconfiguration to migrate existing VNFs, instantiate new VNFs, undertake service scaling as demand patterns change.
- VNFFG optimisation to determine which instances of VNFs 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/VNFFG instances.
- Infrastructure adaptation to overcome streaming difficulties, e.g. to reserve network capacity, allocate greater computational capacity for stream processing, establish expedited paths or reroute flows to avoid congested parts of the network.
- Application-specific adaptation and intelligent network-wide congestion avoidance, for example to configure the capturing or transcoding of 3D models to defined quality levels to match dynamically varying network throughput capabilities and available processing capacity along the NFVI nodes and clusters implementing the VNFFG instance.

In this section we focus on the first iteration of optimisation algorithms for the placement of the set of VNFs forming a NS. This algorithm partially addresses the first two classes of planning algorithms: service placement optimisation and VNFFG optimisation.

In subsequent work we will be defining algorithms for infrastructure and application adaptation, as discussed in section 6.2. The full set of planning algorithms is planned to be documented in deliverable D3.4.

6.1 Optimisation algorithms for the placement and configuration of VNFFGs in media services

The decision on where to locate VNFs and how to interconnect VNF instances to form the end-to-end NS depends on many factors. These include the cost and performance of potential cloud node locations and the network segments interconnecting the VNFs of the NS. Selection a placement solution that is optimal for a single VNF with regard to QoS and cost constraints
is an NP-hard problem\textsuperscript{57}, and the problem of optimising an entire VNF forwarding graph (VNFFG) across distributed cloud nodes is even more complex. In the following we identify three basic VNFFG structures.

### 6.1.1. VNFFG structures

![VNFFG chain structure](image)

One of the most common structures of VNFFG is the chain structure. In Figure 28, given an example of the use case 3, the three virtual functions vOriginServer, vTranscoder and vCache need to be connected as a chain to deliver video stream to users. For this structure, the end-to-end latency will be accumulated from each hop in the chain (including network latency and processing delay at each node).

![VNFFG tree or parallel structure](image)

In Figure 29, we use an example from use case 2 where the cognitive service could be implementing a language translation function. The video stream by vCompression and the translated audio by cognitive service will be merged at the media process engine before sending out to the viewers. The translating step can be done in parallel with video stream processing 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 parallel structure.

![VNFFG loop structure](image)

The third structure we are presenting is a cycle or a loop structure. An example could be the closed loop system shown in Figure 30. It is similar to the chain structure except there is a loop to provide feedback to make decisions in subsequent rounds. The end-to-end latency is accumulated for each hop. Examples of this include services where components send any sort of application feedback to the source.

These three basic structures can be combined to form any complex composite service. In this section, we present algorithms to maximise utility for each type of VNFFG composite structure. The algorithms can be combined to find solutions for more complex structures. We consider several constraints for the VNFFG problem:

• Fixed cost: the cost of deploying the service for the first time at a cloud node. This can be thought as the cost to transfer, install and store the software in that location. The fixed cost is incurred only once and does not vary with the number of service instances at a certain location.
• Linear cost: this cost is proportional to the resources used by the service. The more service instances are deployed the more resources are consumed and hence the cost increases with the number of instances.
• Latency: this includes both the network latency and processing time at cloud nodes. There is a trade-off, for example, between deploying services at a distant node with a higher network latency but faster processing time, or choosing a closer low-latency location with slower processing time. Our algorithms consider this trade-off in the optimisation model. In addition, some services require a higher performance connection between users and the first hop component. For example, users should connect to a low-latency rendering component in an on-line game service to reduce lag as the player moves viewpoint, while the game simulation engine itself could be located more remotely if the position of other players does 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 constraint when deploying a VNFFG.

The initial implementation of the planning service algorithms in the project testbeds for use case 3 does not require complex optimisation. The options for locating the scaled-out CDN NS are extremely limited, due to the small scale nature of the testbeds in the first phase of the project. The algorithms developed for NS placement and VNFFG optimisation have therefore been tested in a larger scale simulation environment, which presents a more challenging optimisation environment. The evaluation was done over a set of 2508 data centres distributed globally in 656 cities. The formulation of the optimisation problem and the simulation-based evaluation is presented in detail in Appendix A – the full details can be found in our journal publication58.

6.2. Planning algorithms for the 5G-MEDIA use cases

A detailed specification of the three use case scenarios and their implementation in the project testbeds is contained in 5G-MEDIA deliverables D2.2 and D6.1. In the following we give an overview of the role of the planning optimisation algorithms in each use case. A selection of these algorithms will be designed in detail and implemented in the second phase of the project. The algorithms and their evaluation will be reported in deliverable D3.4.

6.2.1. Use Case 1: Immersive Media

Placement of VNFs

• vTranscoders can be deployed as pure software components or make use of GPUs. GPU-based deployments are more powerful and are able to implement more complex compression algorithms for streaming 3D models. However they are also more

expensive. CPU-based deployments are less constrained in the sense that they are not restricted by the availability of specialised hardware in edge cloud locations. Placement optimisation will take the cost and performance of the various options for the deployment of vTranscoders in the edge and regional cloud locations, considering the network latency and bandwidth between users and the vTranscoders and the other VNFs. The optimisation algorithm will additionally consider the option of hosting multiple vTranscoders for several players in the same node versus individual instances deployed as close as possible to each player – trading off the cost and performance of the different deployment options.

- An extension of the VNF placement algorithm presented in Appendix A will be used which classifies CPU and GPU-based nodes in different groups for the deployment of vTranscoders.

Dynamic configuration of VNFFGs

- By monitoring the performance of streamed 3D data between players, the computational load on the vTranscoders the planning algorithm may decide to reconfigure the VNFFG to use alternative vTranscoders. For example, if network performance is suffering due to congestion along the path it may be necessary to compress the 3D data further to reduce the data rate and the load on the congested links. However higher compression levels may incur too high a computational load on the machine hosting the CPU-based vTranscoder VM and it may be necessary to redirect the VNFFG to use a GPU-based vTranscoder VNF in a different location.

Infrastructure adaptation

- End-to-end performance of the immersive game can be reduced by the lack of network capacity between player locations or due to too high a latency caused by congestion or inefficient default paths. Although this can be compensated by adjusting the compression levels employed by the vTranscoder it may be necessary to modify the path or the priority of the underlying network. In this case the planning algorithm will request network infrastructure adjustments via the network provider’s SDN controller or other network management system. In the case of multiple spectators and a lack of network resources for the efficient unicast of the game data it may be necessary for the planning algorithm to request the configuration of a dedicated multicast group for distributing the game.

Application-level adaptation

- The compression levels of the vTranscoder can be configured dynamically by the planning algorithm according to the available capacity of the network, as monitored by the monitoring service.

6.2.2. Use Case 2: Mobile Contribution, Remote and Smart Production in Broadcasting

Placement of VNFs

- In the mobile contribution scenario, the Cognitive Services VNF used by the mobile journalist needs to be deployed in a location that offers the performance levels needed for interactive processing of the journalist’s stream. The planning algorithm will
determine the optimal placement of the VNFs in this case, considering the cost and performance trade-offs and whether existing instances can be reused for multiple mobile journalists.

Dynamic configuration of VNFFGs

- If the network performance between the remote venue and the broadcaster site is poor and it is not possible for this to be rectified by the network provider (see below), the planning algorithm may be able to select alternative instances of the vCompression Engine, or Media Process Engine to force the use of a different network path by appropriate detours at the application level.

Infrastructure adaptation

- High quality communications paths between the remote venue and the broadcaster site are critical for broadcast-quality media streams. The performance of the path in terms of available bandwidth, latency and jitter will be monitored and where problems are detected it may be necessary to modify the path or the priority of the video stream in the underlying network. In this case the planning algorithm will request network infrastructure adjustments via the network provider’s SDN controller or other network management system.

Application-level adaptation

- The compression levels of the vCompression Engine can be configured dynamically by the planning algorithm according to the available capacity of the network, as monitored by the monitoring service. There is less flexibility for compression levels than in use case 1 due to the high quality levels required by broadcast quality video streaming, hence there are more options to rectify communications path issues through infrastructure adaptation or through the application-level detour routing (see above).

6.2.3. Use Case 3: Ultra-high Definition (UHD) over Content Distribution Networks (CDN)

Placement of VNFs

- The deployment of vCaches, vTranscoders and vRepeatServers in use case 3 should be done as efficiently as possible at low cost while delivering the expected performance levels. Traffic forecasts from the pre-process & analysis service on anticipated demand are a critical input to the planning algorithms that will determine the best cloud nodes to deploy the CDN VNFs. As users move it may be necessary to deploy new vCaches to meet bandwidth and latency requirements from the new locations of the users. The planning optimisation algorithms will trade-off cost and performance.
- A variant of the initial algorithms described in Appendix A will be used for planning purposes in use case 3.

Dynamic configuration of VNFFGs

- This includes the selection of the best vCache to meet user requests, especially for content personalisation where bespoke VNFs may need to be selected per user in the I-Director scenarios of use case 3.
Infrastructure adaptation

- High quality communications paths between the vCaches and the users are critical for UHD video streams. The performance of the path in terms of available bandwidth, latency and jitter will be monitored and where problems are detected it may be necessary to modify the path or the priority of the video stream in the underlying network. In this case the planning algorithm will request network infrastructure adjustments via the network provider’s SDN controller or other network management system.

Application-level adaptation

- The compression levels of the vTranscoder can be configured dynamically by the planning algorithm according to the available capacity of the network, as monitored by the monitoring service.
- Based on forecasted demand generated by the pre-process & analysis service then planning optimisation algorithms can manage the caching policy and pre-emptive caching of content to maximise local cache hits for users, to avoid high network load to the origin server and video start up latency for the users.
7. Execution

The Execution service is responsible to communication with the Service Orchestrator and announce to it the planning directives of Media Service MAPE. According to the OSM, release 3 and 4 features, the Execution service can only support the action of scaling out a network service to one or more additional VNFs, as explained below.

Typically, an instantiated network service is composed of several VNFs that need to be scaled and operated independently but still belong to the same service and, hence, need to be completely described in the corresponding NS package. Scaling out distributes the workload of the NS across existing and new instantiated resources, thus increases the level of availability and extends the throughput capacity of the service.

5G-MEDIA execution service leverages on OSM functionalities to apply NSs scaling actions. The invoked web service of the OSM Service Orchestrator which is used by the Execution service to trigger such actions is:

```
$ curl --request POST \
--url https://{osm-ip}:{osm-port}/v1/api/config/project/default/ns-instance-config/nsr/{ns_uuid}/scaling-group/{scaling_group_id}/instance \
--header 'accept: application/JSON' \
--header 'authorisation: Basic {token}' \
--header 'cache-control: no-cache' \
--header 'content-type: application/JSON' \
--data '{"instance": [{"id": {scale_id}}]}'
```

where each variable maps to:

- `osm-ip` is the IPv4 of the installed OSM
- `osm-port` is the port that the OSM Service Orchestrator API listens to (by default is 8008)
- `ns_uuid` is the UUID of the instantiated NS
- `scaling_group_id` is the scaling group as it is defined in the descriptors
- `scale_id` is the index of the scaled instance

The set of primitives that should be followed during the scaling process is unified at NSD level. The new elements of the NS are proper deployed with the VDUs interconnected as specified in its descriptor. During the scale-out process, OSM service and resource orchestrators commit resources (e.g. VMs/containers) and deploy images, establish connectivity across all the machines and organise the new service. Once the scaling action is completed, the records of OSM are updated and the NS is orchestrated as a whole by the Service Orchestrator.
8. **Conclusions**

This deliverable has presented the QoS control and management functionality of the 5G-MEDIA project. The two main functions of QoS/QoE monitoring and the cognitive network optimiser have been expanded and positioned within the project’s MAPE architecture. Initial designs for each of the MAPE services have been described. These are currently being implemented in the project’s testbeds. The final system design, algorithm specification and evaluation results will be covered in deliverable D3.4 due at the end of the project.
9. Appendix A: Planning algorithm for optimising the deployment of NSs and VNFFGs

9.1. Optimisation algorithm

In this appendix we present the formulation and evaluation of an optimisation algorithm for the deployment of NSs composed of multiple VNFs. For further details please refer to our publication on this topic.\(^5^9\)

9.1.1. Chain structure

![Diagram](Figure 31. Auxiliary graph for the chain structure.)

To formulate the optimisation problem, we first create an auxiliary service graph as in Figure 31 for the chain structure with two VNFs. Each instance in a dashed circle represents a cloud node location or data-centre (DC), where we can deploy that type of component of the composite service. If a DC is able to deploy multiple types of components, this DC appears in several circles. When in use, we need to create a chain work-flow connecting each instance toward the user. The auxiliary graph is created as follows:

- We create a virtual source connecting to all instances in the last group. The links between the virtual source and those instances have zero latency.
- A full mesh connection is defined between each component as in Figure 31. Each link has the associated network latency and we can remove those which have latencies exceeding the maximum end-to-end or one-hop latency constraints.

Based on this auxiliary graph, we develop a Mixed Integer Linear Program (MILP) formulation to find optimal service placement solutions for the chain structure:

---

Table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_v$</td>
<td>unit cost to deploy the service at DC $v$</td>
</tr>
<tr>
<td>$D$</td>
<td>set of user requests $D = {D_i, \forall i \in I}$</td>
</tr>
<tr>
<td>$D_i$</td>
<td>required resource from user $i$</td>
</tr>
<tr>
<td>$E$</td>
<td>set of links in auxiliary graph</td>
</tr>
<tr>
<td>$F_v$</td>
<td>fixed cost to deploy the service at DC $v$</td>
</tr>
<tr>
<td>$i$</td>
<td>user $i \in I$ where $I$ is a set of users</td>
</tr>
<tr>
<td>$L_{uv}$</td>
<td>network latency for link $(u, v)$</td>
</tr>
<tr>
<td>$N(v)$</td>
<td>set of neighbors of $v$ in the auxiliary graph</td>
</tr>
<tr>
<td>$p^i_v$</td>
<td>if DC $v$ is selected by user $i$ or not</td>
</tr>
<tr>
<td>$P_v$</td>
<td>processing time at DC $v \in V$</td>
</tr>
<tr>
<td>$s$</td>
<td>virtual source</td>
</tr>
<tr>
<td>$t_i$</td>
<td>end-to-end latency perceived by user $i$</td>
</tr>
<tr>
<td>$U_i$</td>
<td>end-to-end utility for user $i$</td>
</tr>
<tr>
<td>$V$</td>
<td>set of nodes in the auxiliary graph</td>
</tr>
<tr>
<td>$x^i_{uv}$</td>
<td>link $(u, v)$ is chosen by user $i$ or not</td>
</tr>
<tr>
<td>$y_v$</td>
<td>DC $v$ is selected or not (for fixed cost)</td>
</tr>
<tr>
<td>$z_i$</td>
<td>a variable used for utility calculation of user $i$</td>
</tr>
</tbody>
</table>

**Figure 32. Summary of key notations**

\[
\max \sum_{i \in I} U_i \quad (1)
\]

s.t.

\[
\sum_{v \in N(u)} (x^i_{uv} - x^i_{vu}) = \begin{cases} 
1 & \text{if } u = s \\
-1 & \text{if } u = i \\
0 & \text{otherwise}
\end{cases} \quad (2)
\]

\[p^i_u \geq x^i_{uv} \forall (u, v) \in E, \ i \in I \quad (3)\]

\[p^i_v \geq x^i_{uv} \forall (u, v) \in E, \ i \in I \quad (4)\]

\[t_i = \sum_{(u,v) \in E} L_{uv} x^i_{uv} + \sum_{v \in V} P_v p^i_v \forall i \in I \quad (5)\]

\[y_v \geq p^i_v \forall v \in V, i \in I \quad (6)\]

\[\sum_{i \in I \ (u,v) \in E} D_i C_v x^i_{uv} + \sum_{v \in V} F_v y_v \leq \text{COST} \quad (7)\]

\[z_i \geq 0 \quad (8)\]

\[z_i \geq t_i - T^i_{\min} \quad (9)\]

\[U_i = \frac{T^i_{\max} - T^i_{\min} - z_i}{T^i_{\max} - T^i_{\min}} \quad (10)\]

\[x^i_{uv}, y_v, p^i_u \in \{0, 1\} \quad \forall i \in I, (u, v) \in E, v \in V \quad (11)\]

**Figure 33. MILP for chain structure**
We use the notations summarised in Figure 32 and the MILP is explained as follows (see Figure 33):

- Objective function (1) is to maximise total utility over all users (the utility function which was defined in our previous work\(^{60}\)).
- (2) represents flow conservation constraints, making sure that a flow from the virtual source to each of the users can be found. There will be no flow outgoing from the user i and no flow incoming to the virtual source s. For intermediate nodes, the outgoing and incoming flows should be equal.
- We use binary variable \(p_{iu}\) and \(p_{iv}\) in constraints (3 - 4) to determine if user i uses node u or node v, then equation (5) is used to compute the end-to-end latency which includes network latency and processing time.
- Constraints in (6) are used to determine if a DC is selected (by any users) or not. These will be used to compute the fixed cost.
- Constraint (7) limits the deployment cost which includes both the fixed cost and the linear cost. The cost of establishing a relationship with a DC and other one-off costs such as installing the application software is represented by the fixed cost. The cost of the computational resources consumed by the running instances of an application component is represented by the linear cost.
- Constraints (8 – 10) are used to compute the utility function.

9.1.2. Tree or parallel structure

We first create an auxiliary graph similar to the chain structure. Let A, B and C be the groups of DCs that are capable to deploy media process engine, cognitive service and vCompression services. We define the MILP model as follows:

\[ \text{objective} \quad (1) \]

\[ \text{s.t.} \]

\[ \sum_{v \in N(u)} (x_{uv}^i - x_{vu}^i) = \begin{cases} 1 & \text{if } u = s \text{ or } u \in C \\ -2 & \text{if } u = i \\ 0 & \text{otherwise} \end{cases} \quad (12) \]

\[ \sum_{v \in B, u \in C} x_{uv}^i = 1 \quad \forall i \in \mathcal{I} \quad (13) \]

\[ \sum_{v \in A, u \in C} x_{uv}^i = 1 \quad \forall i \in \mathcal{I} \quad (14) \]

\[ \sum_{u \in B} x_{ui}^i = 1 \quad \forall i \in \mathcal{I} \quad (15) \]

\[ \sum_{u \in A} x_{ui}^i = 1 \quad \forall i \in \mathcal{I} \quad (16) \]

\[ t_i \geq L_{uv} \left( \sum_{u \in C, v \in B} x_{uv}^i + \sum_{u \in B} x_{ui}^i \right) + \sum_{v \in V} P_v \cdot p_v^i \quad \forall i \in \mathcal{I} \quad (17) \]

\[ t_i \geq L_{uv} \left( \sum_{u \in C, v \in A} x_{uv}^i + \sum_{u \in A} x_{ui}^i \right) + \sum_{v \in V} P_v \cdot p_v^i \quad \forall i \in \mathcal{I} \quad (18) \]

**Constraints (3, 4, 6 – 11)**

*Figure 34. MILP for tree structure*

Where (see Figure 34):

- Constraints (12 - 16) are used to make sure that we will find a parallel structure from s to each of the users.
- Constraints (17 - 18) are used to find the maximum latency between the two branches. This maximum value will be the end-to-end latency for the composite services.
9.1.3. **Cycle or loop structure**

The auxiliary graph will be created similar to the one applied for the chain or the parallel structures. Let A, B and C be the groups of DCs that are capable to deploy “compare and adjust”, “process” and “monitor” services, respectively. The MILP model is defined as follows:

\[
\text{objective} \quad (1) \\
\text{s.t.} \\
\sum_{v \in N(u)} (x_{uv}^i - x_{vu}^i) = \begin{cases} 
1 & \text{if } u = i \\
-1 & \text{if } u \in A \\
0 & \text{otherwise}
\end{cases} \quad (19) \\
\sum_{u \in A, v \in B} x_{uv}^i = 1 \quad \forall i \in \mathcal{I} \quad (20) \\
\sum_{u \in B, v \in C} x_{uv}^i = 1 \quad \forall i \in \mathcal{I} \quad (21) \\
\sum_{u \in C, v \in A} x_{uv}^i = 1 \quad \forall i \in \mathcal{I} \quad (22) \\
\sum_{u \in A} x_{iu}^i = 1 \quad \forall i \in \mathcal{I} \quad (23) \\
t_i = \sum_{v \in N(u)} L_{uv} x_{uv}^i + \sum_{v \in V} P_v p_v^i \quad \forall i \in \mathcal{I} \quad (24) \\
\text{Constraints} \quad (3, 4, 6 - 11)
\]

*Figure 35. MILP for cycle structure*

Where (see Figure 35):

- Constraints (19 - 23) are used to make sure that we will find a cycle flow for each user.
- Constraints (24) are used to compute the end-to-end latency of the cycle structure.

9.2. **Evolutionary algorithm for managing VNFFGs**

Solving the MILP optimisation for large datasets is not feasible and hence we have developed an evolutionary algorithm to produce close to optimal results in a reasonable execution time, as discussed in the following.
9.2.1. Genetic algorithm

To illustrate the optimisation of VNFFGs, we present a genetic algorithm for the chain structure with three components, shown in Figure 36. We can apply this strategy to solve for other composite structures. Each group in Figure 36 represents a set of DCs capable of deploying one kind of component (e.g., some components can only be deployed in DCs equipped with special hardware like GPU, etc.). If a DC can deploy all three components, it will appear in all groups.

![Figure 36. Example of chain structure](image)

The genetic algorithm is based on the flow chart depicted in Figure 37. We define a fitness function $f(x)$ to evaluate each chromosome $x$ in the population. Based on the fitness score, good chromosomes are selected for the next generations.

In this genetic algorithm, the algorithm tries to maximise the following fitness function:

$$\max f(x) = \text{utility} - \alpha \times \text{cost}$$

where $\text{cost}$ is the sum of the linear and the fixed costs at DCs. We use $\alpha$ as a parameter to give a trade-off between the utility and the cost. Good chromosomes are the ones that have higher values of $f(x)$. We explain in detail the steps of the GA:
• **Start:** the initial population is a set of potential solutions to the problem. Each solution contains a list of vectors. Each vector is a list of integers representing a chain structure connecting a user to each of the three components. As an example, 

\[(1, 1, 2, 1), (2, 2, 1, 1), (3, 1, 2, 3)\] represents a possible solution for the three users: \((u_1, A_1, B_2, C_1), (u_2, A_2, B_1, C_1), \text{and} (u_3, A_1, B_2, C_3)\). We randomly choose some solutions to be in the initial population. Some of them may not be feasible, but they will be eliminated in next generations of the genetic algorithm.

• **New population:** after each round, a new population is created by the following steps until reaching the stopping criterion (maximum number of generations):
  
  o **Selection:** select parents' chromosomes from a population according to their fitness (the higher the fitness, the higher chance to be selected).
  o **Crossover:** with a crossover probability, swap part of the information between pairs of parents to form new children.
  o **Mutation:** with a mutation probability, randomly alter some genes inside a chromosome to get a new chromosome.

• **Stop:** if the end condition is reached (e.g. maximum number of generations), the algorithm stops, and returns the best solution in the current population.

### 9.2.2. Order-based genetic algorithm

By introducing the fixed cost, we try to minimise the number of DCs in use. We observe that if we pick up users one by one, and try to maximise the fitness of those users, then the order of users to be picked up is important and affects the final solution. This occurs since the flows of later users that reuse components deployed in DCs used by earlier user flows do not incur the fixed cost. In this section, we introduce an order-based algorithm, which is actually a genetic algorithm with a different representation (based on permutations) aiming to find the best sequence of user flows to consider.

• **Initial population:** random orders of users, e.g. \((u_2, u_3, u_1)\), \((u_1, u_3, u_2)\) are used to create the initial population.

• **To decode a solution,** for each user in the sequence, greedily select an assignment that minimises \(f = (\text{latency} + \alpha \times \text{cost})\). The cost here includes the fixed and linear cost. For more detail on the greedy approach, for instance, we choose the order \((u_2, u_3, u_1)\). Then starting from \(u_2\), we need to select \(A_1, A_2\) or \(A_3\) so that the value of \(f\) is minimal. Let's say we choose \(A_1\), then from \(A_1\) we need to decide to go to \(B_1, B_2\) or \(B_3\) such that the value of \(f\) is minimal and so on. After finishing for \(u_2\), we continue with \(u_3\) in a similar way. Because of the fixed cost, the order of users is important.

• **Stop:** when reaching a maximum number of generations, the algorithm returns the best order of users that maximises \(\text{fitness} = (\text{utility} - \alpha \times \text{cost})\), where \(\text{utility} = \gamma \times U_{\text{first-hop}} + U_{\text{e2e}}\).

We implemented the genetic algorithms using “inspired” - the open source framework for creating biologically-inspired computational intelligence algorithms in Python\(^{61}\). The inspyred library provides basic components such as a generator to define how solutions are created.

\(^{61}\) [https://pypi.python.org/pypi/inspyred](https://pypi.python.org/pypi/inspyred)
and an evaluator that defines how fitness values are calculated for solutions. In addition, several evolutionary operators are also available to use such as “selector”, “variator”, “replacer” (to determine parents and new population) and “terminator” (to say whether the evolution should end).

9.3. Evaluation

We solve the mixed integer linear program model using IBM’s CPLEX solver. All computations were carried out on a computer equipped with a 3 GHz CPU and 8 GB RAM. We use a dataset with 2508 data centres distributed in 656 cities all over the world. The fixed deployment cost is based on the Amazon EC2 charging model. The user demand is proportional to the population of each city. Latency between users and execution zones are computed based on Haversine distance between two points around the planet’s surface.

In the following, we use an example chain of three components, where the first component has much tighter constraints on network latency to the users than the others. For example, this could be a cloud-based rendering service for a 3D virtual reality environment or online game, where the scene, from the point-of-view of the user needs to track the user’s head movement in real time and, hence, the latency between the user and this component should be very low to avoid noticeable lag. The other components may be managing the environment state and providing background objects and textures, and although end-to-end latency to the distant component needs to be within the overall utility bounds, the constraints are not as tight as to the renderer. This pattern of component chains, with extra constraints on the positioning of the nearest component, could be applicable to many services, including virtual and augmented reality, games and video conferencing.

For the purpose of simulation, based on measurements of QoS of on-line games, we configure the utility function and other related parameters as follows:

- The first hop represents the graphic rendering component. We set $T_{\text{min}} = 20 \text{ ms}$ and $T_{\text{max}} = 50 \text{ ms}$ for first hop utility.
- Most players in impaired games can tolerate latencies of up to $150 \text{ ms}$, and so we set $T_{\text{min}} = 50 \text{ ms}$ and $T_{\text{max}} = 150 \text{ ms}$ for the end-to-end (E2E) utility, which covers the full chain of components.
- We consider latency to be the sum of network latency and processing time at DCs. This processing time is considered inversely proportional to DC's fixed cost (the intuition is the more expensive a DC is, the faster processing time it has).

We summarise the notations used in the simulations as follows:

- Improvement score of algorithm (A) vs. algorithm (B) = \( \frac{100(f(A)-f(B))}{|f(A)|} \)

---


64 [https://github.com/richardclegg/multiuservideostream](https://github.com/richardclegg/multiuservideostream)

• Fitness $f = \text{Utility} - \alpha \cdot \text{Cost}$
• Utility = $\gamma \cdot U_{\text{first-hop}} + U_{\text{e2e}}$
• GA (order): we first run the order-based genetic algorithm to find a solution, then put this solution along with random ones into the initial population.
• GA (rand.): we run the genetic algorithm in which the initial population is totally random.
• Order-based: we run the order-based generic algorithm.
• Best random: we randomly select $(2 \times 10^5)$ solutions for both the small and the large dataset, then get the one with the best fitness value.

9.3.1. Pareto graph of utility vs. cost

We first focus on the simulation results for the chain structure as the observations we found are also similar for the parallel and the cycle structures. The dataset includes 25 users, 3 groups of DCs (group A, B and C as in Figure 36), each group consists of 58, 78 and 85 DCs in which we randomly select from the full dataset (around 10% of the total 656 DCs). We set a stopping criterion for the genetic algorithm (GA) so that it will explore around 2% of the searching space ($2 \times 10^5$ solutions).

Given a placement solution, we can plot its cost and utility on a 2-D plane as in Figure 38. For the MILP, given cost as a constraint, we try to maximise the utility for that cost and find each point in the Pareto curve (in red). We identify 10 points in which the cost is varied from 0 to a maximum value ($\text{COST}_{\text{max}}$) that allows finding the best utility (note that this utility cannot be improved, even when the cost is larger than $\text{COST}_{\text{max}}$). Then, we set each value of cost in the constraint and find the optimal solution which maximises the utility. Effectively, the curve connecting these optimal points forms a Pareto front on the plane. Based on this Pareto front, service providers can trade-off between cost and utility for their services. In Figure 38, we also show the computation time of the algorithms. It took the MILP 16 hours to find the optimal curve (1.6 hours in average to find one point in the curve), while only 150 seconds for the GA algorithm to find their Pareto curves. While reducing execution time significantly, the solutions of the GA algorithm are close to optimal.
9.3.2. Minimum cost vs. maximum utility vs. GA algorithms

In this section, we made a comparison between our utility-based optimisation versus the cost-minimisation approach widely adopted in other work in the literature, as indicated by our results on minimum cost GA. In the minimum cost GA, we set \( \alpha \) to be large enough in the fitness function \( \text{fitness} = \text{utility} - \alpha \times \text{cost} \) to force the algorithm to minimise the cost. On the other hand, in the maximum utility GA we set \( \alpha = 0 \), meaning that the algorithm tries to maximise the utility. The GA (order) used in Figure 39 - Figure 40 is the GA in which we test with different values of \( \alpha \) and choose the one that has a good trade-off between utility and cost.

As shown in Figure 39 - Figure 40, the GA results in marginally higher costs than the minimum cost algorithm, while its utility is close to the maximum. It is noteworthy to mention that the maximum utility and the GA are also close to the optimal maximum utility solution. For example, when \( \gamma = 10 \) (Figure 40), the maximum total utility is \( 1834 \times 11 = 20174 \) (1834 users, each has maximum utility score \( = 10 \times U_{\text{first hop}} + U_{\text{e2e}} = 11 \)). This confirms that the GA can find good solutions even for a larger dataset. We refer the reader to our paper\(^{66}\) for more detailed evaluation results.