

5G EVE

5G European Validation platform for Extensive trials

Deliverable D5.8

Testing and validation methodologies final report with the final version of testing and validation suite

Project Details

<i>Call</i>	H2020-ICT-17-2018
<i>Type of Action</i>	RIA
<i>Project start date</i>	01/07/2018
<i>Duration</i>	36 months
<i>GA No</i>	815074

Deliverable Details

<i>Deliverable WP:</i>	WP5
<i>Deliverable Task:</i>	Tasks T5.3, T5.4
<i>Deliverable Identifier:</i>	5GEVE_D5.8
<i>Deliverable Title:</i>	Testing and validation methodologies final report with the final version of testing and validation suite
<i>Editor(s):</i>	Christos Ntogkas, Evangelos Kosmatos (WINGS)
<i>Author(s):</i>	Christos Ntogkas, Evangelos Kosmatos, Ioannis Chondroulis, Antonia Pelekanou (WINGS), Elian Kraja, Giada Landi (NXW), Luis Miguel Contreras (TID), Frédéric Faucheux (NOKIA-FR), Emmanouil Paraskevakis (NOKIA-GR), Claudio Casetti (CNIT), Jaime Garcia-Reinoso, Luis Félix González Blázquez, Winnie Nakimuli (UC3M), Vassilis Laskaridis, Vera Stavroulaki, Nelly Giannopoulou, Ioannis Belikaidis, Vassilis Foteinos, Kostas Tsagkaris (WINGS)
<i>Reviewer(s):</i>	Georgios Sextos (NOKIA-GR), Enrique Garcia (ASTI), Rodolphe Legouable (ORA-FR)
<i>Contractual Date of Delivery:</i>	31/05/2021
<i>Submission Date:</i>	31/05/2021
<i>Dissemination Level:</i>	PU
<i>Status:</i>	Final
<i>Version:</i>	V1.0
<i>File Name:</i>	5GEVE_D5.8

Disclaimer

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

Deliverable History

Version	Date	Modification	Modified by
<i>0.1</i>	<i>17/02/2021</i>	<i>ToC proposal</i>	<i>Evangelos Kosmatos (WINGS)</i>
<i>0.2</i>	<i>26/02/2021</i>	<i>ToC and partners assignment finalized</i>	<i>Christos Ntogkas (WINGS)</i>
<i>0.3</i>	<i>18/03/2021</i>	<i>First contributions from partners</i>	<i>WINGS, NXW</i>
<i>0.4</i>	<i>29/03/2021</i>	<i>Final contributions from partners</i>	<i>TID, NOKIA-GR, NOKIA-FR, UC3M, CNIT, NXW, WINGS</i>
<i>0.5</i>	<i>31/03/2021</i>	<i>Consolidated version</i>	<i>Evangelos Kosmatos (WINGS)</i>
<i>0.6</i>	<i>06/04/2021</i>	<i>Internal review</i>	<i>Georgios Sextos (NOKIA-GR)</i>
<i>0.63</i>	<i>09/04/2021</i>	<i>Comments from internal review addressed</i>	<i>NXW, WINGS, TID, NOKIA-GR, NOKIA-FR, UC3M, CNIT</i>
<i>0.7</i>	<i>16/04/2021</i>	<i>External review</i>	<i>Enrique Garcia (ASTI), Rodolphe Legouable (ORA-FR)</i>
<i>0.72</i>	<i>19/04/2021</i>	<i>Comments from internal review addressed</i>	<i>WINGS, NXW, TID, NOKIA-GR, NOKIA-FR, UC3M, CNIT</i>
<i>0.8</i>	<i>20/04/2021</i>	<i>Final version</i>	<i>Evangelos Kosmatos (WINGS)</i>
<i>0.9</i>	<i>23/04/2021</i>	<i>QA check</i>	<i>Kostas Trichias (WINGS)</i>
<i>0.91</i>	<i>16/05/2021</i>	<i>Comments from TM</i>	<i>Giada Landi (NXW), Manuel Lorenzo (ERI-ES)</i>
<i>0.92</i>	<i>20/05/2021</i>	<i>Comments from TM review addressed</i>	<i>Evangelos Kosmatos (WINGS)</i>
<i>1.0</i>	<i>31/05/2021</i>	<i>Submitted to EC</i>	

Table of Contents

LIST OF ACRONYMS AND ABBREVIATIONS	V
LIST OF FIGURES	VII
LIST OF TABLES	VIII
EXECUTIVE SUMMARY	9
1 INTRODUCTION	10
1.1 5G EVE TESTING AND VALIDATION METHODOLOGY	10
1.2 DOCUMENT STRUCTURE	11
2 5G EVE TESTING AND VALIDATION FRAMEWORK	13
2.1 ARCHITECTURE	13
2.2 PROCEDURES	14
2.2.1 KPI collection and monitoring	15
2.2.2 KPI validation	16
2.2.3 Reporting of validation results	16
3 EXPERIMENT EXECUTION MANAGER	18
3.1 HIGH LEVEL ARCHITECTURE AND WORKFLOW	18
3.2 INTERNAL ARCHITECTURE	20
3.3 IMPLEMENTATION DETAILS	23
4 RESULT ANALYSIS AND VALIDATION	25
4.1 INTERNAL ARCHITECTURE	25
4.1.1 RAV – EEM interface	27
4.1.2 RAV – Performance Diagnosis Module interface	28
4.2 IMPLEMENTATION DETAILS	28
5 PERFORMANCE DIAGNOSIS.....	31
5.1 INTERNAL ARCHITECTURE	31
5.1.1 PD - RAV interface	33
5.2 IMPLEMENTATION DETAILS	33
5.2.1 Self-Organizing Map	34
5.2.2 Root Cause Analysis	34
5.3 PERFORMANCE DIAGNOSIS ALGORITHMS EVALUATION	35
5.3.1 SOM Algorithm Evaluation	36
5.3.2 RCA Algorithm Evaluation	37
5.3.3 Evaluation of the SOM and RCA algorithms on the 5G EVE platform	38
6 CONCLUSION	42
REFERENCES	43

List of Acronyms and Abbreviations

<i>Acronym</i>	<i>Meaning</i>
5G-PPP	5G Infrastructure Public Private Partnership
API	Application Programming Interface
BMU	Best Matching Unit
CtxB	Context Blueprint
DL	Downlink
DN	Data Network
E2E	End-to- End
EEM	Experiment Execution Manager
ELM	Experiment Lifecycle Manager
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core
ExpB	Experiment Blueprint
FSM	Finite State Machine
gNB	Next-generation NodeB
GUI	Graphical User Interface
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ITU	International Telecommunications Union
IWF	Interworking Framework
IWL	Interworking Layer
KPI	Key Performance Indicator
MAC	Medium Access Control
MEF	Metrics Extractor Function
mMTC	Massive Machine Type Communication
ML	Machine Learning
NG-RAN	Next Generation Radio Access Network
OTT	One-Trip Time
PD	Performance Diagnosis
QoS	Quality of Service
RAN	Radio Access Network
RAV	Results Analysis and Validation
RC	Runtime Configuration
RCA	Root Cause Analysis
REST	Representational State Transfer

<i>RTT</i>	Round-Trip Time
<i>SLA</i>	Service Level Agreement
<i>SOM</i>	Self-Organizing Maps
<i>TCB</i>	TestCase Blueprint
<i>UC</i>	Use Case
<i>UE</i>	User Equipment
<i>UL</i>	Uplink
<i>URLLC</i>	Ultra-Reliable Low Latency Communications
<i>VNF</i>	Virtual Network Function
<i>VR</i>	Virtual Reality
<i>VSB</i>	Vertical Service Blueprint

List of Figures

Figure 1: 5G EVE testing and validation approach.....	10
Figure 2: 5G EVE testing and validation framework architecture	13
Figure 3: Monitoring metrics architecture.....	15
Figure 4: Interaction between Experiment Execution Manager and 5G EVE components	18
Figure 5: EEM REST API.....	21
Figure 6: EEM Finite state Machine for an experiment execution.....	24
Figure 7: RAV placement in the 5G EVE platform	26
Figure 8: RAV API.....	26
Figure 9: Node status timeline.....	29
Figure 10: Experiment topology and metrics weights graphs	29
Figure 11: Service KPI profiles	30
Figure 12: Performance Diagnosis information flow	31
Figure 13: Performance Diagnosis workflow.....	32
Figure 14: Performance Diagnosis module API.....	33
Figure 15: Workflow of the RCA algorithm	35
Figure 16: Abnormal node behaviour metrics.....	36
Figure 17: Normal node behaviour metrics.....	36
Figure 18: (a) Relation between f1 score and the fault detection threshold and (b) SOM algorithm accuracy	37
Figure 19: Topology used for the RCA algorithm's evaluation.....	37
Figure 20: RCA algorithm's evaluation results.....	38
Figure 21: AirWINGS application deployed as a Network Service.....	38
Figure 22: Accuracy of the deployed SOM algorithm	39
Figure 23: General experiment information and KPI statistics	39
Figure 24: Node status timeline.....	40
Figure 25: Experiment topology and metrics weights graphs	40
Figure 26: Service KPI profiles.....	41

List of Tables

Table 1: Workflow step descriptions.....	19
Table 2: ExperimentExecutionRequest data type.....	20
Table 3: EEM data model: ConfigureExperimentRequest	22
Table 4: EEM data model: InfrastructureMetricsConfigRequest.....	22
Table 5: EEM data model: InfrastructureMetricsRemoveRequest.....	22
Table 6: EEM data model: ConfigureRAVRequest	23
Table 7: EEM data model: ValidateKPIs	23
Table 8: RAV configuration information	27
Table 9: RAV Testcase configuration object information.....	27
Table 10: RAV metric information object.....	27
Table 11: RAV KPI information object	27

Executive Summary

In this deliverable D5.8 the final version of the 5G EVE testing and validation platform is presented, focusing on the components of the platform which are related with the execution of the experiment, the analysis of the collected metrics, the validation of the generated KPIs and finally the generation of the final reports provided to the experimenters for the use case data analysis.

The key 5G EVE components which are related with the experiment execution, the analysis and validation of the results and the diagnostic functionalities and they were defined and developed in the project are the “Experiment Execution Manager”, the “Results Analysis and Validation” and the “Performance Diagnosis”. Together with these three components, also a “Tests Scripts Inventory” has been included to store the different scripts associated with different Test Cases. In this deliverable D5.8 the final versions of the aforementioned components are reported. In addition, the development updates and extensions are compared to the previous versions reported in previous deliverables (D5.2[2], D5.3[3], D5.5[4]).

Initially, in the document the overall final 5G EVE testing and validation platform is presented, as well as the relation of testing/validation/diagnosis components with the rest of the 5G EVE components. Then, the final versions of WP5 components are presented and the new features and extensions are presented.

- The Experiment Execution Manager (EEM) component is responsible to drive the execution of all the tests run on top of the 5G EVE infrastructure. Its activity starts after the experiment has been fully deployed, upon trigger from the Experiment Lifecycle Manager (ELM).
- The Results Analysis and Validation (RAV) component is responsible to realise the appropriate analysis on the collected metrics, to validate the generated KPIs and also to generate the experiments reports that will become available to the experimenters.
- The Performance Diagnosis (PD) component has the responsibilities to provide insights on the execution and validation of the tests, to provide additional information about the health status of the network and service elements and to execute diagnostics and root cause analysis mechanisms in case of performance degradations.

In the document, for each component, the internal architecture, the interactions, the related procedures and the implementation details are presented and explained in detail.

1 Introduction

This deliverable D5.8 presents the final 5G EVE testing and validation methodology and developed framework. In the current section, the 5G EVE testing and validation methodology is summarised, while the structure of the document is explained.

1.1 5G EVE testing and validation methodology

The 5G EVE testing and validation methodology describes the steps that should be taken starting from collecting the vertical objectives and requirements and collecting information related to:

- the services or applications the vertical will provide on top of the network infrastructure;
- the network topology and deployment;
- the test scenarios that will be useful for the vertical to be evaluated;
- the list of KPIs together with their minimum requirements.

Then, the methodology translates all the aforementioned information into a set of technical specific test cases that are executed in order to evaluate the selected KPIs against the preferable defined criteria.

An initial version of 5G EVE testing and validation methodology was presented in D5.1 [1], while this methodology was further extended leading to their final version that was presented in detail in D5.2 [2]. In addition, the low-level procedures included in the methodology are presented in detail in D5.3 [3]. This final version has remained unchanged until now, therefore it is summarised below for the shake of the completeness of the document.

The main target of the evaluation methodology is to provide the appropriate guidelines to all involved stakeholders in order to design the test cases, to prepare the tests, to execute and finally evaluate the service test performance results. Independently of low-level conditions, requirements, etc., the high-level workflow for the evaluation process consists of the four phases presented in Figure 1.

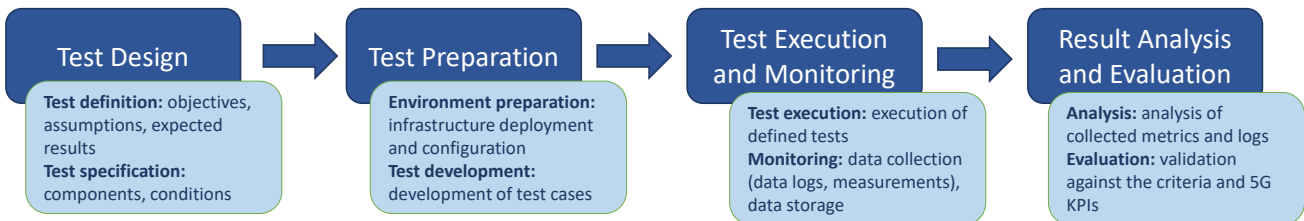


Figure 1: 5G EVE testing and validation approach

Test Design

The main goal is to permit 5G EVE Experiment Developers to understand and define the objectives of the different experiments, how they can be executed and what KPIs are critical for the experimenters. This phase is key, as the whole interaction between Vertical and Facility Provider will be influenced by how well they are able to align different perspectives from an early stage.

The main external input to this phase is the “Vertical test plan template” which is used by the Verticals in order to convey “in their own language” what are their intentions from the evaluation process. The Vertical test plan template is presented in detail in deliverable D5.1 [1]. The generated Vertical test plans are used by the Experiment Developers (technical experts) of the sites in order to generate the “Low-level test plan templates”. The low-level templates translate the Vertical requirements into specific service blueprints, test blueprints and test scripts, which are defined and described in deliverable D5.2 [3]. In 5G EVE different type of Blueprints are

defined, including Blueprints for Vertical Service/Context/Test Case/Experiment aspects. In addition, other components are defined including test scripts, required VNFs/PNFs, etc.

The “Vertical test plan” includes a set of test cases. These test cases define the experiments at the Experimenter-level and must be translated into a more technical and execution-oriented language. By the term “Experimenter”, we define the person who will actually execute the tests. In other words, a meaningful test plan is to be generated, including not only the inputs from the experimenters, but also the testing procedures, the test configurations, the measurable KPIs (vertical- and network-related), the required supporting infrastructure, etc.

In order to accomplish this translation, and in order to generate an appropriate set of pilot site tests, the Experiment Developers will use the evaluation procedures described in detail in D5.3 [3]. The main target of this translation is to create a set of low-level test procedures in order to evaluate the KPIs defined in the high-level pilot test plans. Finally, it is also expected that VNF providers upload their packages into the pilot sites during this phase.

Test Preparation

The objective of this phase is to get everything ready to execute the desired test. All tests have some initial conditions that must be set up by Experiment Developers, and most of them have some variables and threshold values for validation that Verticals may want to customize. This customization enables the generation of the required descriptors from the blueprints. From there, it should be ensured that the testing environment is ready to host the various tests.

Test Execution and Monitoring

Once the entire infrastructure is ready, the virtual environment (e.g., virtual machines, containers, VNFs) to run the tests is built and configured. This should be done automatically, and when ready, the Experimenter can start executing the different test cases. Both application and infrastructure metrics are gathered and made available both for monitoring (visualization) and for validation purposes.

Result Analysis and Evaluation

Results are collected and analysed, and the final evaluation report is generated. There are many levels of analysis, depending on the expectations of the Verticals and their interests for specific KPIs. A set of network KPIs are measured by default by the 5G EVE platform, in order to validate the results against the 5G PPP KPIs and as a way to ensure that the obtained results are valid (i.e., obtained under correct network conditions). The list of 5G EVE network KPIs are presented and explained in detail in D5.2 [2].

Finally, and based on the gathered results, it is checked that the values of the KPIs are fulfilling the success criteria; if so, the final test report will indicate that the test has been passed; otherwise, it will have failed.

In the evaluation methodology, among the four phases, the most critical and cumbersome phase is the initial “Test Design” phase. The “Test Design” phase requires a high level of interaction with the Vertical Customers and high effort in order the Vertical Customers’ requirements and envisage scenarios to be mapped to a set of clearly defined test cases. In addition, after being filled by the Vertical Customers, this template is further translated to a set of “Low level test plans” which are the detailed technical descriptions of the test cases that should be executed during the “Test execution and monitoring” phase. Based on the evaluation methodology, one “Low level test plan” may include one or several evaluation procedures.

1.2 Document structure

D5.8 is structured as follows:

- Section 2 presents the 5G EVE testing and validation architecture. The high-level architecture of the framework is presented, focusing on the components that are designed and developed in WP5. In addition, their position and the communication with the other 5G EVE components are explained and presented. Finally, the main processes adopted in the components for the KPI validation, monitoring and reporting are presented.

- In Sections 3, 4 and 5 the main components of 5G EVE testing and validation architecture are presented. Section 3 presents the architecture, design aspects and implementation details of Experiment Execution Manager (EEM). In Section 4, the Results Analysis and Validation (RAV) module is explained in detail. The RAV architecture, both internal and its interaction with the external components are presented, the design principles are explained, while the implementation details are reported. Section 5 presents the Performance Diagnosis module including its internal architecture, interaction with other components and implementation aspects. In addition, the algorithms developed inside the component are also presented.
- Finally, Section 6 summarises the document including the key findings.

2 5G EVE testing and validation framework

In this section, the designed and developed 5G EVE testing and validation framework is briefly presented. The high-level architecture of the framework is presented, the components developed in WP5, namely “Experiment Execution Manager”, “Results Analysis and Validation” and “Performance Diagnosis” modules are explained, while their position in the whole 5G EVE developed framework is presented. In addition, the main processes adopted for the KPI validation, monitoring and reporting are presented.

2.1 Architecture

The 5G EVE testing and validation framework architecture is illustrated in Figure 2. In this figure, the overall architecture is presented including components from all the 5G EVE technical WPs (WP2/WP3/WP4/WP5) as well as the relation of WP5 modules with the rest of the 5G EVE components. In detail, WP3 modules appear in yellow colour, while those of WP4 appear in green and those of WP5 in red. The colour of the different arrows shows the allocation of the module that starts the communication.

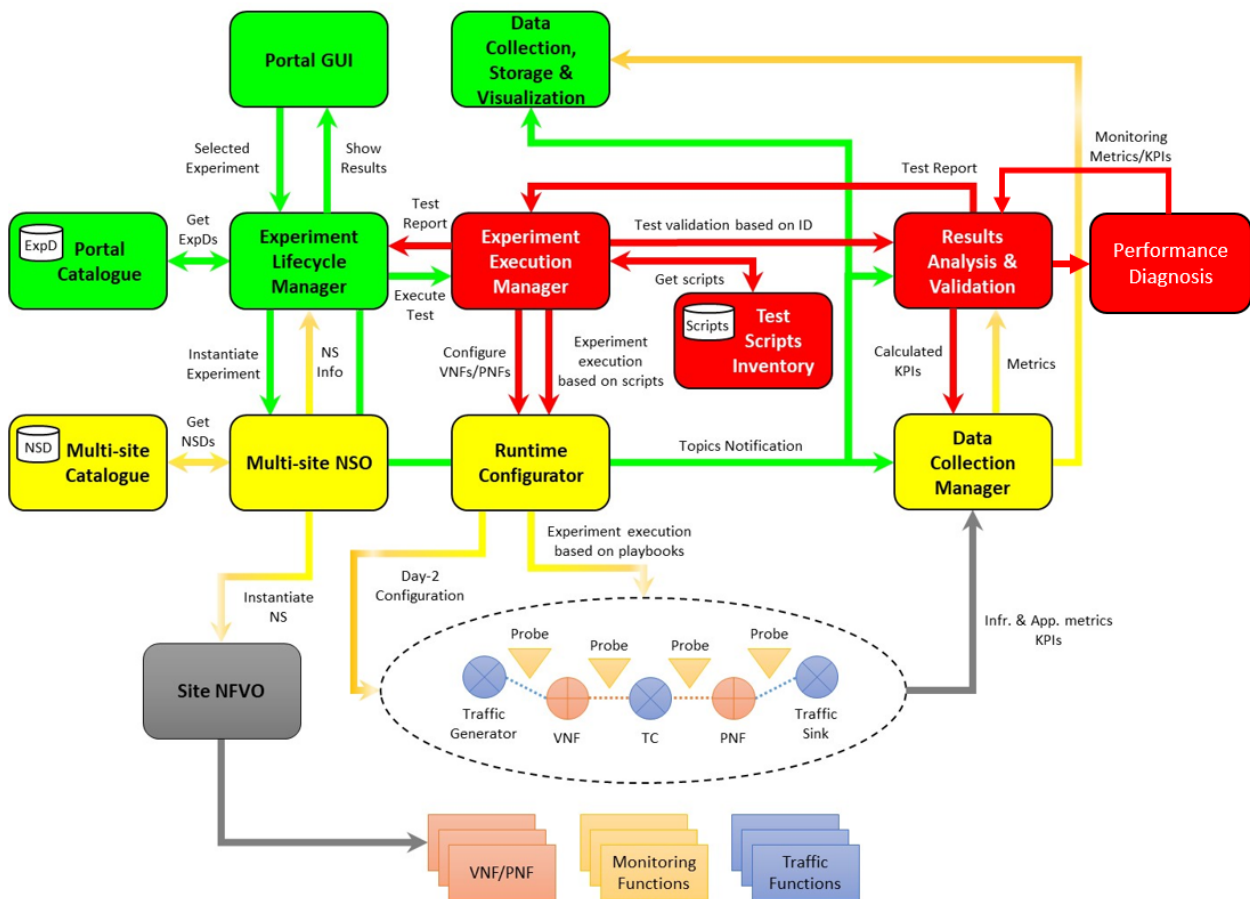


Figure 2: 5G EVE testing and validation framework architecture

As depicted in Figure 2, three are the key modules that WP5 has defined as part of the 5G EVE Test and Validation framework. These are the “Experiment Execution Manager”, the “Results Analysis and Validation” and the “Performance Diagnosis” components. Together with these three components, also a “Tests Scripts Inventory” has been included to store the different scripts associated with different Test Cases.

The main role of the Experiment Execution Manager (EEM) is to drive the execution of all the tests that will run on top of the 5G EVE infrastructure. Its activity starts after the experiment has been fully deployed, and after it gets a request from the Experiment Lifecycle Manager (ELM). The EEM has four main functionalities:

- to trigger the day-2 configuration through the Runtime Configurator;
- to trigger the execution of test cases leaning on the Runtime Configurator for the interaction with the site components;
- to trigger the validation of test cases through the Results Analysis and Validation module;
- to return to initial conditions after every test case execution.

The Experiment Execution Manager (EEM) is described in D5.2 [2] (first version of EEM), while its final version is presented in detail in Section 3.

The role of the Results Analysis and Validation (RAV) module is to determine the correctness of each executed experiment. This correctness is evaluated in terms of obtained results, i.e., indicating whether a test case is passed or failed or a collected KPI falls below a specified target value. The main features of the RAV module are:

- to calculate KPIs from raw metrics;
- to validate the test results;
- to generate the test report.

The Results Analysis and Validation (RAV) module is reported in D5.2 [2] (first version of RAV), D5.5 [4] (almost final version of RAV), while its final version is presented in detail in Section 4.

The Performance Diagnosis (PD) module is the third WP5 module, designed and developed during the second year of the project. Therefore, the initial testing and validation framework architecture described in D5.2 [2] was extended to support performance diagnosis functionalities. The main objectives of the developed performance diagnosis mechanism are:

- to provide insights on the execution and validation of the tests;
- to provide information about the health status of the network and service elements;
- to execute diagnostics and root cause analysis mechanisms in case of performance degradations.

The main functionalities of 5G EVE advanced performance diagnosis mechanism including:

- to perform a deeper KPI analysis by analysing additional data;
- to execute anomaly detection on the collected metrics;
- to perform Root Cause Analysis (RCA) and
- finally generate useful insights for alleviating any performance degradation.

At the end of the second year, initial results from the application of the performance diagnosis mechanisms on emulated networks were collected and analysed. The outcomes of this process as well as the almost final version of the performance diagnosis module is reported in D5.5 [4], while the final version of PD is presented in detail in Section 5.

The Test Scripts Inventory has the only responsibility to store the required scripts for all the experiments, making them available on demand to the EEM.

2.2 Procedures

The procedures of the 5G EVE testing and validation framework that are related with the WP5 components (“Experiment Execution Manager”, “Results Analysis and Validation” and “Performance Diagnosis” components) are related with the collection of metrics/KPIs, validation of KPIs and reporting of validation results. These procedures are explained in the next paragraphs.

2.2.1 KPI collection and monitoring

The correct execution of the KPI monitoring process is subject to the data collection mechanism to obtain the proper monitoring data that will feed the monitoring platform. As a result, the KPI collection process is a critical phase that must be deeply evaluated and designed from the different stakeholders implied in the testing process.

This results in the usage of a central component in the testing system, capable of collecting and persisting all the monitoring data (being more precise: the metrics gathered from the monitored components – VNFs, PNFs and other components – and the KPIs and results inferred from these metrics) that are required to be gathered during the execution of experiments, and delivering them to specific components that demand this monitoring data, with two purposes: (i) the monitoring of the experiment and (ii) the validation of the targeted KPIs.

The main challenge to address in this process is the management of multiple and different sources of monitoring data, such as activity log files, configuration data, active/passive probes or monitoring devices, among others. Each source would provide the monitoring data in a specific format, needing the application of data processing techniques to homogenize the data format to be received by the Monitoring system. Moreover, these sources may be located in different physical locations, so a distributed system is required to ingest the monitoring data from different sources.

This problem is addressed in the 5G EVE project with the implementation of a monitoring platform capable of gathering data from multi-site 5G environments, following the standard architecture presented in Figure 3. The monitoring platform is presented in detail in deliverables D3.4 [5].

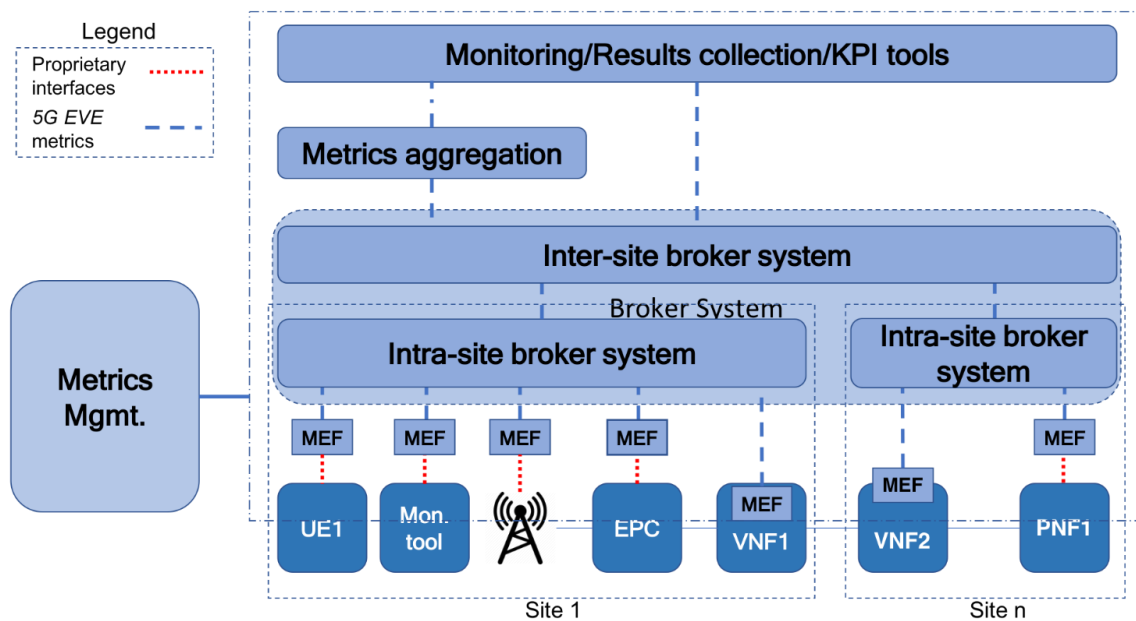


Figure 3: Monitoring metrics architecture

In Figure 3, there are different components in each site that could be monitored (user equipment, base stations, the EPC, the VNFs deployed, etc.) and the format of their logs may be different, as explained before. Consequently, a first set of functions to be applied must be related to the extraction and translation (if required) of the metrics generated by a heterogeneous set of infrastructure components, role played by the Metrics Extractor Function (MEF) entity in Figure 3. Note that this way of obtaining data from components must be transparent for verticals.

For the distribution of the monitoring data between the implied entities, a Broker System was applied, being composed by two brokering levels:

1. An Intra-site broker, deployed per site, whose role is to eventually harmonize the monitoring data format to provide it in a unified way, preserving the data privacy of each site;
2. An Inter-site broker, which interconnects all sites together to both aggregate monitoring data through the Metrics aggregation component, generating new metrics based on those provided by the MEFs, and directly provide them to a set of value-added additional components, such as KPI performance diagnosis tools, data analytics platforms, SLA enforcement mechanisms or data visualisation services, which can be fed from the monitoring data provided by the system.

2.2.2 KPI validation

For the KPI evaluation process, three major pillars are considered namely, the statistical analysis, the measurements correlation check (in the case that in addition to the target KPI complementary measurements are available) and the prediction analysis based on Machine Learning (ML) tools.

- Statistical analysis:
The analysis of the KPIs is based on the results collected in the conducted experiments. An experiment consists of one or more test cases, which contain multiple iterations of a single test. The statistical properties of a single test are calculated from the measurements collected in the test. The statistical properties of a complete test case are obtained by taking the average of the corresponding properties of the test iterations in the test case. This results in a collection of normally distributed test case results whose averages will be close to the real value of the statistical property. Furthermore, it allows to specify confidence intervals for them using the Student-T distribution.
- Correlation and causality analysis:
Correlation is a statistical association between observed variables. Correlation techniques can reveal similar, or in extreme cases identical, behaviour between KPIs or other monitored variables. A high positive correlation is the most intuitive case, where two variables exhibit the same nature of change (increase and decrease). However, if two variables show opposing change (one always increases while the other decreases), they are also similar. This is called negative correlation. In a practical example, correlation analysis allows to test how strongly two KPIs are depending on a third KPI. A low correlation score might indicate that factors other than the monitored variables should be considered.
- Prediction analysis:
The focus here is on predicting how a change in the system might influence a variable's behaviour. For example, a use case for prediction can be to estimate the required network deployment (e.g., adding or removing network elements) to guarantee acceptable KPI values. Other potential use cases include prediction of KPI degradation upon network element (re-)configuration and prediction of the effect of specific KPIs.

In 5G EVE testing and validation platform, the statistical analysis approach is used in Result Analysis and Validation module for the actual analysis of collected metrics and the validation of requested KPIs. This process is further explained in detail in Section 4.

In the Performance Diagnosis module, metric correlation and causality analysis approaches are utilised in order to: a) provide insights regarding the correlation between different metrics and KPIs; b) to realise Root Cause Analysis (RCA) in case of impairments or service degradations. In addition, predictive analytics are also utilised in the Performance Diagnosis module in order to estimate the health status of network and services components and identify any impairments of performance degradations, which will cause the trigger of RCA functionalities.

2.2.3 Reporting of validation results

The 5G EVE testing and validation framework includes a complete and well-defined reporting mechanism, which provides to experimenters, clear answers to their initial validation questions. In addition, this reporting mechanism gives a complete picture of the testing and validation process realised using the infrastructure. In addition, it includes, to passed/failed results, information about the actual testing process, the environment in which the tests are executed, the different test cases executed to provide the results, as well as high-level configuration information. All this information is provided in a way that ensures the transparency and repeatability of the entire testing and benchmarking process.

In this direction, the final report is consolidated through the composition of a set of sections created by the elements that participate in the testing and validation platform. This is important because, while the main focus of the experiments is the KPI validation, the information regarding the infrastructure, the conditions and the technologies used can be extremely useful and insightful to the experimenter as well.

Reporting the information generated from each of the various stages of the experiment definition, preparation and execution processes happen independently in the respective stages. The final validation report contains the following sections:

- General experiment information regarding the executed experiment, selected site, and time slot;
- Test case related information such as selected KPIs to validate and the validation conditions requested;
- Validation results section that includes the behavior of each of the selected KPIs for the whole duration of the experiment and the final validation result;
- Performance diagnosis section, if PD was requested by the vertical, that includes: a) analysis results of the status of the service elements during the experiment execution, b) the topology of the deployed service, c) statistics regarding the metrics most impactful to the performance of the various elements and finally d) a comparison between the performance of the monitored KPIs for each distinct deployment, resources wise, that the selected service was tested with.

3 Experiment Execution Manager

3.1 High level architecture and workflow

The Experiment Execution Manager (EEM) is the 5G EVE platform component that coordinates all the procedures for the automated execution and validation of experiments in the 5G EVE infrastructure. The high-level architecture and the software architecture introduced in D5.2 [2] are still valid.

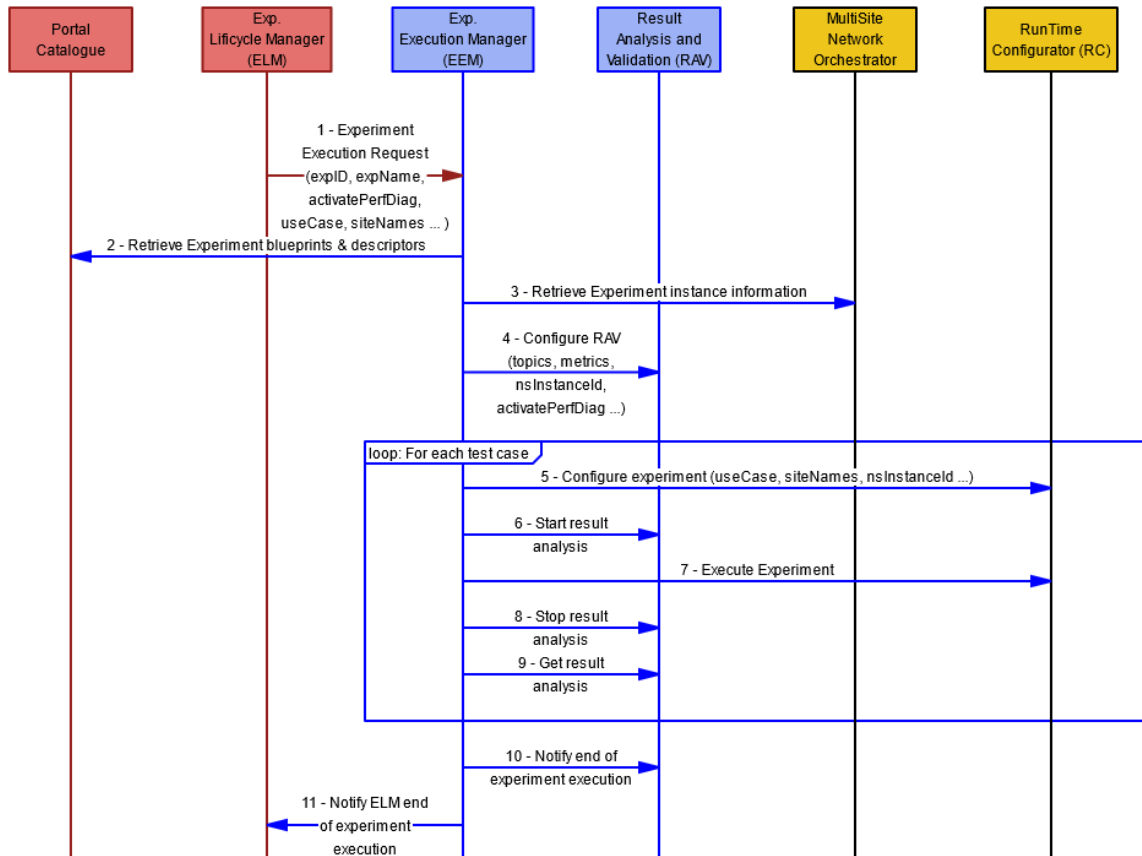


Figure 4: Interaction between Experiment Execution Manager and 5G EVE components

The workflow related to the experiment execution, depicted in Figure 4, shows the interactions between the EEM and the other components of the 5G EVE platform. The workflow defined in D5.5 [4] is still valid, with some minor updates made in the interaction between the EEM and the Experiment Lifecycle Manager (ELM) and between EEM and the Result Analysis and Validation tool (RAV) to allow the execution of the Performance Diagnostic tool. The enabler of the Performance Diagnosis tool is triggered by the Portal GUI, and it is notified to the EEM through the ELM when the execution of an experiment is requested (step 1). A new parameter, named “activatePerfDiag” has been added to the list of the parameters sent from the ELM to the EEM.

When this parameter is selected, the EEM will notify the RAV that the experiment has required the activation of the Performance Diagnosis tool when the EEM requests the result analysis (step 4) with the RAV. Table 1 presents the different steps of the workflow shown above.

Table 1: Workflow step descriptions

Step	Involved entities	Description
1	ELM, EEM	Experiment execution request. In this step the ELM sends a request to the EEM for the execution of an experiment that has been deployed in the virtual environment. The request includes the details of the deployed experiment, including the ID of the corresponding Network Service, information about its blueprints and the activatePerfDiag flag that enables the diagnostic functionalities.
2	EEM, Portal Catalogue	Retrieval of experiment blueprints and descriptors. In this step the EEM reads the various blueprints and descriptors to get all the information needed to configure, execute and evaluate the experiment. In particular, the EEM retrieves information about the scripts to be launched for the configuration and the execution, their variable parameters, the list of application and infrastructure metrics and KPIs to be collected and evaluated. Guidelines for writing the configuration and execution scripts are available in D3.4 [5]
3	EEM, MSNO	Retrieval of information on the deployed experiment instance. In this step the EEM performs a query related to the Network Service instance associated to the experiment. This query allows the EEM to retrieve the VNFs' IP addresses that must be included in the configuration and execution scripts.
4	EEM, RAV	Configure RAV. In this step the EEM performs a POST to the RAV to notify a new experiment execution. The payload of the request contains the list of topics, name of the metrics, the flag to enable the performance diagnosis tool, network instance id and other useful parameters needed by the RAV.
5	EEM, RC	Experiment configuration. In this step the EEM interacts with the RC to request the Day 2 configuration of the VNFs and the configuration of the data shippers in the site facilities for the collection of infrastructure metrics. In the former case, the EEM sends the configuration script retrieved from the TCB and filled with the runtime parameters provided by the user or gathered from the MSNO. In the latter case the EEM send the list of the infrastructure metrics that must be collected, as specified in the ExpB.
6	EEM, RAV	Activation of result analysis procedure. In this step the EEM interacts with the RAV to activate the process of collecting metrics and compute KPIs, providing all the related details available in the blueprints.
7	EEM, RC	Experiment execution. In this step the EEM interacts with the RC to request the execution of the script to run the experiment. As in step 4, the EEM retrieves the script from the TCB and completes it with the runtime parameters provided by the user or gathered from the MSNO. After triggering the execution, the EEM starts a polling thread to query the RC about the execution status and to follow the progress of the experiment.
8	EEM, RAV	Termination of execution and activation of result analysis procedure. In this step the EEM notifies the RAV about the termination of the experiment execution and requests the RAV to proceed with the result analysis.
9	EEM, RAV	Query of result report. In this step the EEM sends a query to the RAV for retrieving the URL of the report generated in the result analysis procedure. This URL is reported to the ELM in the execution record, so that the results can be displayed in the GUI.

10	EEM, RAV	Notify end of experiment execution. In this step the EEM notifies the RAV that each of the test cases of the experiment have been executed.
11	EEM, ELM	Notify end of experiment execution. In this step the EEM notifies the ELM that each of the test cases of the experiment have been executed.

3.2 Internal architecture

The Experiment Execution Manager allows external components to interact with it through a REST API interface, mainly consumed by the Experiment Lifecycle Manager (ELM). The API represents the north-bound interface of the EEM and the list of methods of the REST API presented in D5.2 [2] is still valid.

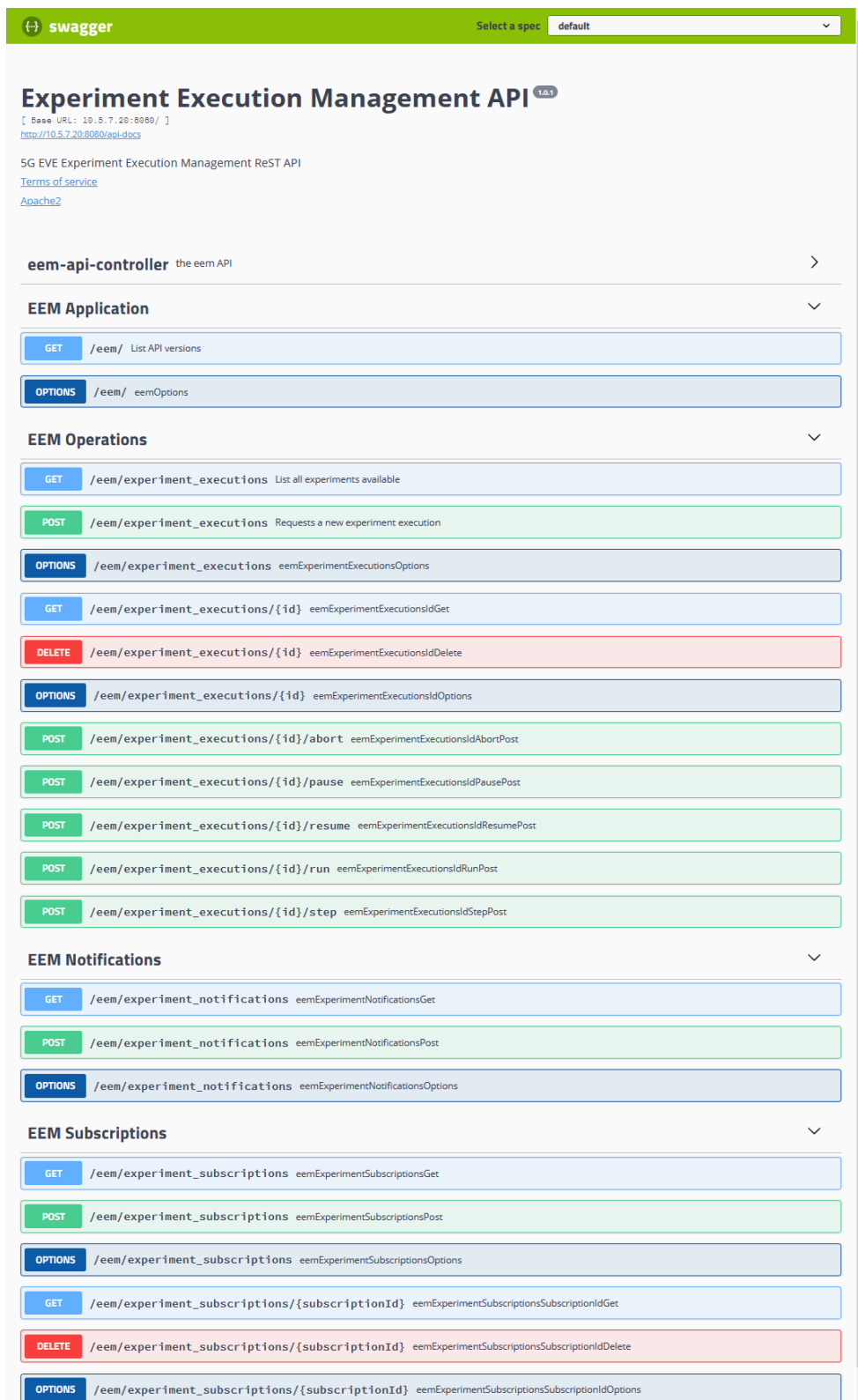
As already mentioned for the workflows of the EEM described in Section 3.1, the content of the messages exchanged on the REST API interface, depicted in Figure 5, has been slightly updated to accommodate the new requirements of the 5G EVE platform, with respect to the experiment execution. In particular, the north-bound interface has been enhanced to allow the ELM to provide to the EEM the parameters to activate the Performance Diagnosis tool during the execution of the experiment. Table 2 shows the parameters provided by the ELM to request a new experiment execution.

Table 2: ExperimentExecutionRequest data type

Name	Type
executionId*	String
experimentDescriptorId*	String
nsInstanceId*	String
testCaseDescriptorConfiguration	Map<String, Map<String, String>>
experimentName*	String
applicationMetrics	Map<String, String>
infrastructureMetrics	Map<String, String>
kpiMetrics	Map<String, String>
activatePerfDiag	boolean
siteNames	List<String>
useCase	String

The updated data model shows that the ELM sends to the EEM the application, infrastructure and KPI metrics. These parameters are not directly used by the EEM itself but are sent to the RAV and the RC when the execution of the test cases of the experiment runs. The ELM provides as well the activatePerfDiag that is used by the RAV to enable the performance diagnosis tool, and the siteNames and useCases that are used by the RC to properly identify the topics related to the experiment.

At the same time, the EEM interacts with other components to properly complete the execution of the experiments that are required by the Verticals through the 5G EVE portal. As depicted in Figure 4, the EEM interacts with the RC to start the configuration of the VNF related to the service. In this case, the EEM is acting as client, and the information sent to the RC is shown in the following Table 3. The “**executionId**” parameter identifies the running experiment. The configuration script, identified as “**configScript**”, is formatted as string, using a well-known format that is translated in ansible commands by the RC. The reset script uses the same format as the configuration script and is executed by the runtime configuration when the experiment execution terminates.



Experiment Execution Management API ^{1.0.0}

[Base URL: 10.5.7.20:8080 /]
<http://10.5.7.20:8080/api-docs>

5G EVE Experiment Execution Management ReST API
[Terms of service](#)
[Apache2](#)

eem-api-controller the eem API

EEM Application

- GET /eem/ List API versions
- OPTIONS /eem/ eemOptions

EEM Operations

- GET /eem/experiment_executions List all experiments available
- POST /eem/experiment_executions Requests a new experiment execution
- OPTIONS /eem/experiment_executions eemExperimentExecutionsOptions
- GET /eem/experiment_executions/{id} eemExperimentExecutionsIdGet
- DELETE /eem/experiment_executions/{id} eemExperimentExecutionsIdDelete
- OPTIONS /eem/experiment_executions/{id} eemExperimentExecutionsIdOptions
- POST /eem/experiment_executions/{id}/abort eemExperimentExecutionsIdAbortPost
- POST /eem/experiment_executions/{id}/pause eemExperimentExecutionsIdPausePost
- POST /eem/experiment_executions/{id}/resume eemExperimentExecutionsIdResumePost
- POST /eem/experiment_executions/{id}/run eemExperimentExecutionsIdRunPost
- POST /eem/experiment_executions/{id}/step eemExperimentExecutionsIdStepPost

EEM Notifications

- GET /eem/experiment_notifications eemExperimentNotificationsGet
- POST /eem/experiment_notifications eemExperimentNotificationsPost
- OPTIONS /eem/experiment_notifications eemExperimentNotificationsOptions

EEM Subscriptions

- GET /eem/experiment_subscriptions eemExperimentSubscriptionsGet
- POST /eem/experiment_subscriptions eemExperimentSubscriptionsPost
- OPTIONS /eem/experiment_subscriptions eemExperimentSubscriptionsOptions
- GET /eem/experiment_subscriptions/{subscriptionId} eemExperimentSubscriptionsSubscriptionIdGet
- DELETE /eem/experiment_subscriptions/{subscriptionId} eemExperimentSubscriptionsSubscriptionIdDelete
- OPTIONS /eem/experiment_subscriptions/{subscriptionId} eemExperimentSubscriptionsSubscriptionIdOptions

Figure 5: EEM REST API

Table 3: EEM data model: ConfigureExperimentRequest

Name	Type
executionId*	String
tcDescriptorId *	String
configScript *	String
resetScript	String

*** mandatory fields**

The same data model is used to send the execution commands. The “**configScript**” parameter in this case contains the translated script, in string format, that will run in the VNFs during the execution phase.

The interaction with the RC includes other steps, like the configuration of the infrastructure metrics during the experiment execution. In this case, the EEM sends the RC the data model depicted in Table 4. In this case, the EEM requests the configuration of the network probes available in the infrastructure, through the RC, for the list of infrastructure metrics selected by the Vertical.

Table 4: EEM data model: InfrastructureMetricsConfigRequest

Name	Type
executionId*	String
tcDescriptorId *	String
metrics *	List<MetricInfo>
nsInstanceId *	String

*** mandatory fields**

In addition to the metrics list, the RC must receive the network service identifier, **nsInstanceId** parameter, for the necessary configurations that may be required on the probes (i.e.: the IP address of the virtual network function that generates traffic is needed in case the probe is configured for the user data rate downlink/uplink). This information is gathered by the RC invoking the MSNO.

When the execution of the experiment ends, the RC receives the request to remove the configuration from the probes. The data model used by the EEM in this case is described in the following Table 5.

Table 5: EEM data model: InfrastructureMetricsRemoveRequest

Name	Type
executionId*	String
tcDescriptorId *	String
metrics *	List<MetricInfo>

*** mandatory fields**

The EEM interacts with the RAV component, as already mentioned, to enable to metrics gathering and the validation of the KPIs, based on the configurations provided by the Experiment Developer, when the experiment is created.

The configuration request provides to the RAV all the necessary information to reach the involved Kafka Broker, and consume the list of topics related to the experiment. The data model for the request is described in Table 6.

Table 6: EEM data model: ConfigureRAVRequest

Name	Type
vertical*	String
expID *	String
perfDiag	Boolean
nsInstanceId	String
tcID*	String
topics*	List<Topic>
publish*	List<PublishTopic>

*** mandatory fields**

The EEM will send the topics list, where the topic data model includes the broker details, the topic name and the metric name that should be consumed by the RAV. The “**vertical**” parameter identifies the vertical that is executing the experiment. The “**publish**” parameter is the list of the KPIs that are calculated by the RAV. The data model is similar to the topic. It contains the name of the KPI, the topic where to publish the calculated data, the list of the metrics used to get the KPI, and the formula used to calculate it.

In addition, the EEM can request to the RAV the activation of the performance diagnostic tool. The necessary parameters to notify this to the RAV are the “**perfDiag**” boolean, set to true, and the “**nsInstanceId**”, that allows the RAV to interact with the MSNO and get the information related to experiment.

Once the experiment ends its execution, the EEM notifies it to the RAV, and consequently requests the validation of the KPIs. The data model used in the request is defined in the following Table 7.

Table 7: EEM data model: ValidateKPIs

Name	Type
experimentId*	String
executionId*	String
tcDescriptorId*	String

3.3 Implementation details

The EEM component, already described in D5.2 [2], is able to configure and run a set of test cases for a given experiment through the API and the workflow described in Section 3.1.

The implementation described in D5.2 sec 4.1 is still valid. The EEM components, upon reception of a request for an experiment execution, contacts the Portal Catalogue to get the blueprints and descriptors related to the experiment, and the multisite network orchestrator (MSNO) to get the network service information of the instantiated experiment. When all the information is gathered, the EEM starts the translation of the test cases, and provides the translated information to the runtime configurator that is in charge of the configuration and run of the experiment.

During the configuration phase, the EEM triggers the day-2 configuration on the vertical service through the runtime configurator and it triggers the configuration of the RAV, allowing this to be ready to gather the metrics produced during the experiment execution, and validate them during the validation phase. After the configuration phase, the EEM triggers the execution invoking the runtime configurator and, at the end of the execution, the EEM sends a request to the RAV to stop the metrics gathering and start the validation phase. During the validation period, the EEM polls the RAV until the validation result is ready.

After the validation phase, the experiment is completed. The internal state machine related to an experiment execution is described in Figure 6.

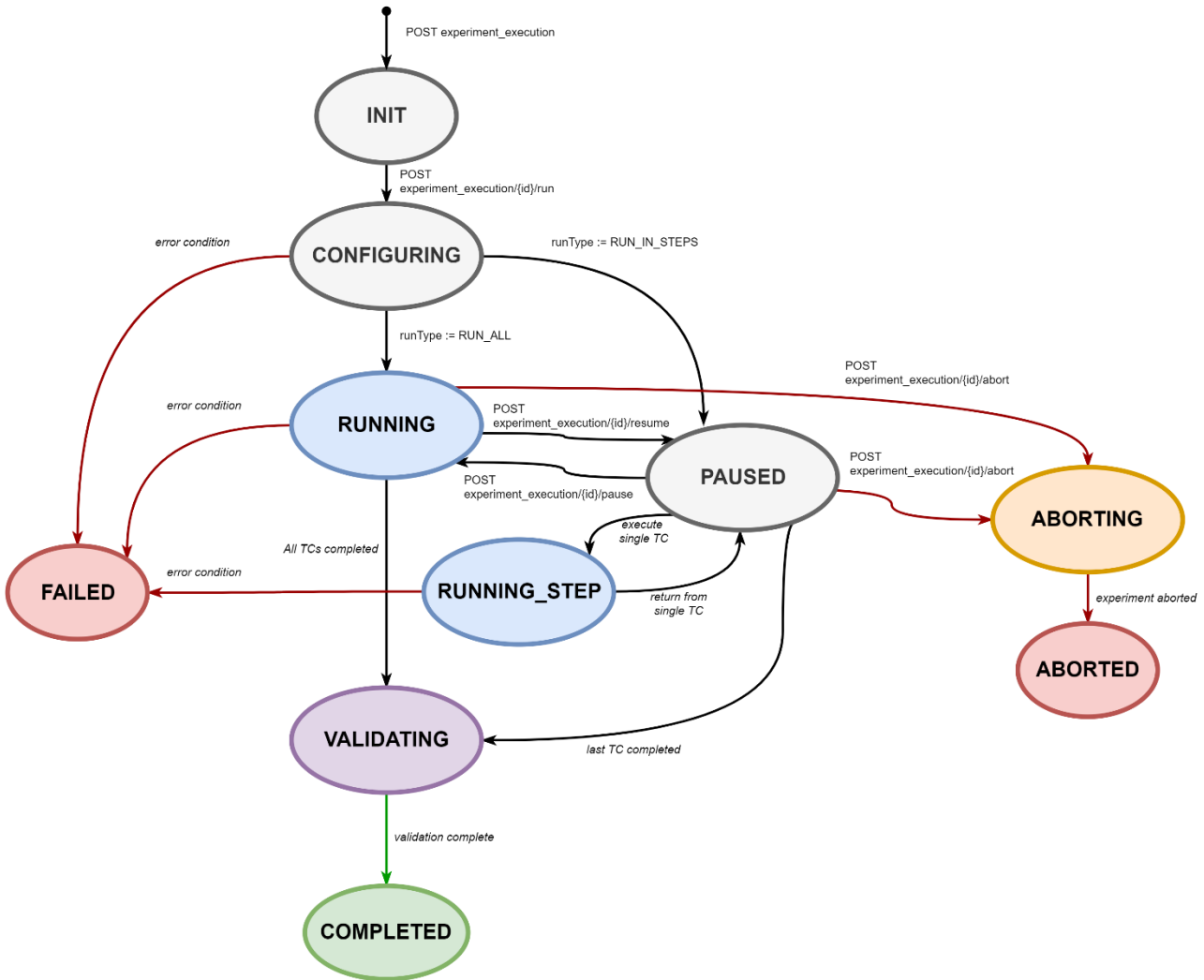


Figure 6: EEM Finite state Machine for an experiment execution

In case of errors at any state of the experiment execution, the experiment transits into FAILED state. For explicit requests to abort an experiment, the related states ABORTING and ABORTED are used to track the progress.

With respect to the EEM described in D5.2 [2], the core functionalities of the EEM have not changed. The major updates are related to the external components that the EEM interacts with, to complete the experiment execution. Different plugins have been implemented to interact with the external components, like the Runtime Configurator plugin or the Result Analysis and Validation plugin.

4 Result Analysis and Validation

The Result Analysis and Validation (RAV) module is responsible for collecting all the metrics generated during an experiment, calculating the KPIs selected by the Vertical responsible for the executed experiment and validating them using a set of acceptable parameters, also provided by the Vertical. It is the first step in the performance evaluation process followed by the performance diagnosis, if requested, after the experiment has concluded.

The initial implementation of the RAV module for the disjoint testing and validations tools delivered in D5.1 [1] was a sub-functionality of the initial testing framework design, originally built around Robot Framework and Jenkins. During the project, the design of the individual RAV module, as well as the Testing framework, changed to a standalone module in order to accommodate emergent new needs, to enhance its capabilities and enable the interfacing with other entities of the 5G EVE platform, such as the Data Collection Manager (DCM), Experiment Lifecycle Manager (ELM), Experiment Execution Manager (EEM) and so on. Another benefit of its, now, standalone status is the ability to support the monitoring of multiple experiments running in parallel.

4.1 Internal architecture

The RAV module is a component located in the Interworking Layer (IWL) of the 5G EVE platform but not directly related with the deployment and orchestration as most of the other IWL components. The functionalities provided by this component are classified in three categories, each implemented in a separate submodule. These categories are:

- Analysis and validation;
- Experiment information storage;
- Reporting.

The main submodule performs the analysis and validation operations as well as handling the communication with other components of the 5G EVE platform using appropriate interfaces. The second submodule handles the management of the experiment data, pertaining to the RAV operations, such as experiment IDs, monitored metrics, KPIs that need calculation, formulas for those KPIs and validation conditions. Finally, the third submodule handles the generation of the final report based on the produced metrics and KPIs during the execution of the experiment and the requested validation conditions set by the Vertical during the experiment creation.

As seen in Figure 7, RAV directly interfaces with the EEM and the PD module. In both cases this interface is used for configuration of the module and operational control in a chain-like way where the configuration and control commands from EEM get passed to RAV and then to the PD module. RAV also communicates, albeit one-way, with MSNO in order to retrieve information regarding the network service and the topology of the experiment. Lastly, communication between RAV and DCM includes consuming and publishing metrics and KPIs generated during the experiment execution.

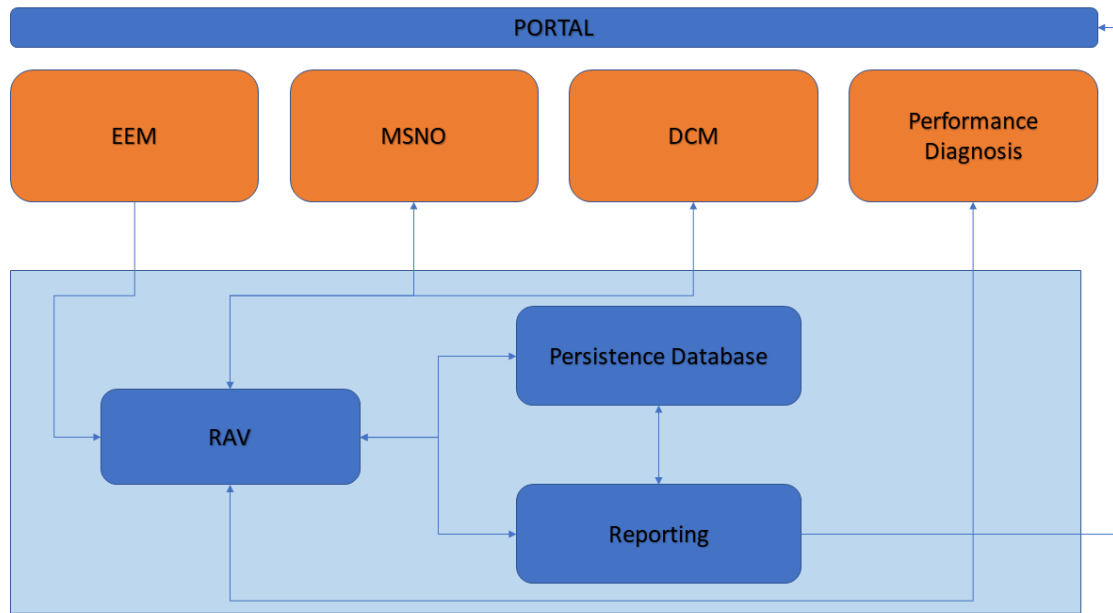


Figure 7: RAV placement in the 5G EVE platform

Interfacing the RAV module with the other components is accomplished using a REST API, seen in Figure 8, exposing all the necessary functionalities.

POST	<code>/configuration/{executeID}</code>	adds configuration for an experiment	↑
GET	<code>/configuration/{executeID}</code>	get the configuration for an experiment	↑
GET	<code>/queue</code>	dump list of active RAV instances	↑
GET	<code>/queue/{executeID}</code>	dump results for specific experiment	↑
GET	<code>/queue/{executeID}/{tcID}</code>	dump results for specific testcase	↑
GET	<code>/messages</code>	dump the consumed messages per experiment and topic	↑
GET	<code>/validationResults</code>	dump the validation results per experiment-testcase combination	↑
GET	<code>/terminate/{executeID}</code>	terminate the experiment	↑
GET	<code>/terminate/{executeID}/{tcID}</code>	terminate current testcase of experiment	↑
GET	<code>/validate/{executeID}/{tcID}</code>	start testcase validation	↑
GET	<code>/status/{executeID}/{tcID}</code>	show validation status	↑
DELETE	<code>/remove/{executeID}</code>	remove a finished experiment	↑

Figure 8: RAV API

4.1.1 RAV – EEM interface

The EEM is responsible for configuring the RAV for each experiment execution. All the required information to configure the RAV module can be found in Table 8.

Table 8: RAV configuration information

Name	Description
expID	Experiment ID string
vertical	Vertical name string
perfdiag	Enable/Disable Performance Diagnosis
identifier	Network Service instance ID
testcases	List of testcase configuration objects

The testcase configuration object contains information regarding the testcase specifics. Each testcase provides a set of metrics that will be measured during the experiment. Furthermore, a list of KPIs is provided with their corresponding formulas for calculating them and metrics to be used as inputs for these formulas. Both the metrics and the KPIs are accompanied with the information pertaining to the broker that will be used for consuming / producing these metrics/KPIs. Lastly, a set of validation conditions is provided by the vertical to validate each testcase, which is in turn passed to RAV through this configuration. The information contained in a testcase configuration is shown in Table 9.

Table 9: RAV Testcase configuration object information

Name	Description
tcID	Testcase ID string
configuration	Configuration dictionary with the metrics and KPIs

The configuration field contains two fields, topics and publish that are the lists of the metrics and KPIs that will be monitored and published during the experiment execution. For reference, the information for metric and KPI topics follows the format seen in Table 10 and Table 11 respectively.

Table 10: RAV metric information object

Name	Description
brokerAddr	Broker endpoint to reach the topic
Topic	Topic name to consume the metric messages from
metric	Metric name

Table 11: RAV KPI information object

Name	Description
brokerAddr	Broker endpoint to reach the topic
topic	Topic name to publish the KPI in
kpi	KPI name
input	List of metrics to be used as input for the KPI formula

formula	Formula string to be evaluated using the inputs
interval	Interval of the calculation and publishing of the KPI
unit	Unit of the KPI values
lowerBound	Lower acceptable value for the validation of the KPI
upperBound	Upper acceptable value for the validation of the KPI

Except configuration, the EEM is also responsible for managing the state of the RAV instances by issuing execution and termination requests for the various experiments.

4.1.2 RAV – Performance Diagnosis Module interface

The RAV module is responsible for configuring and controlling the PD module since its operation happens in parallel to the analysis and validation. The configuration information of the PD module is similar to the one used by RAV but it is enriched with additional information regarding the Network Service which is tested during the experiment. The additional information includes topology information for the elements taking part in the experiment as well as information regarding the resources used to deploy each of these elements. This information will be used in the Root Cause Analysis and the generation of a service profile for the given resources, respectively.

This information is retrieved using the Network Service instance ID and executing a request to the MSNO, for the Network Service ID, and the IWL Catalogue, for the Network Service Descriptor. The communication with these components is one-way so no additional interfaces needed.

4.2 Implementation details

The RAV component, as described in previous deliverable D5.2 [2], is capable of monitoring, analysing, and validating single or multiple test cases for a given experiment through its API. The implementation presented in D5.2, is still the same. During the functional testing of the component, the discovered bugs were fixed and performance fine-tuning was also achieved leading to its final version.

The RAV component upon receiving a registration request for an experiment, retrieves the Network Service Descriptor if the performance diagnosis is requested, and creates the experiment instance that will be used to monitor the experiment. This operation includes steps like preparing the consumers for the provided metrics and the publishers for the provided KPIs, generating the appropriate data structures for the data that will be needed to execute the validation as well as handle any signalling required for the PD module.

After the experiment execution is triggered, the metrics are consumed from the DCM component, the KPIs are calculated using the provided Vertical input and the KPIs are published back to the DCM to be used by other components like the Monitoring component. When the experiment execution has finished, the EEM signals to the RAV to start the validation stage. After the validation has finished and the results have been generated and stored to the persistence database, the RAV executes a request to the reporting component to generate the report for this experiment and returns an endpoint to the EEM, through which the report will be exposed to the Portal so that it will become available for the Vertical. At this stage, the operational cycle of the RAV for an executed experiment is complete.

While the RAV module has not changed, the Reporting submodule has received an extension to incorporate the PD results to a single final report. The version prior to the PD module implementation, as part of the RAV module has been presented in detail in D5.5 [4]. In order to present the PD results, it has been extended with an additional segment in the report. This segment contains graphs that showcase the insights generated from each of the PD algorithms as well as an additional graph that displays the performance profile for each KPI of the tested service based on the resources allocated for the deployment of the service.

As explained in Section 5.2.1 the first algorithm utilized in the PD operations cycle is the SOM. The output of the SOM algorithm is used to generate a timeline graph with the status of each node for each moment of the experiment execution. Additionally, the impact of each metric/KPI on the final status is weighed so as to identify in more detail the issue that caused that status. Such a timeline, from a service used as an example, can be seen in Figure 9. Based on the metrics and KPIs generated during the experiment execution, three states of operation are identified during the execution, healthy, spike and unhealthy. These are decided based on the training of the models during executions. In this example scenario, impairments were manually injected first on nodeB and later nodeA. Both being in the data flow of the client’s requests, it had an impact on the KPIs of the client as well. On the contrary, spikes do not seem to impact the client as much.

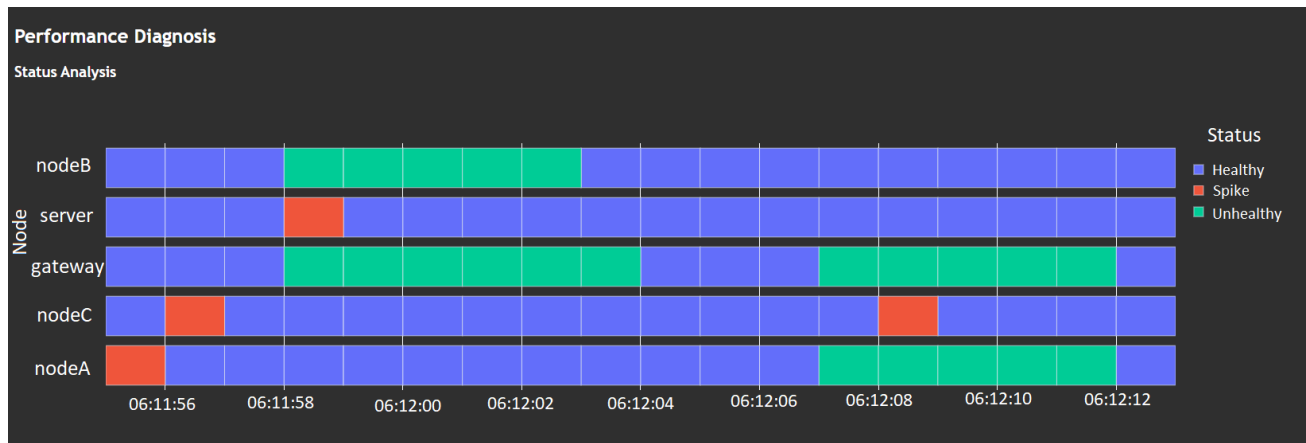


Figure 9: Node status timeline

Using this timeline, the Vertical can identify bottlenecks or anomalies in the service workflow and even correlate specific actions during the experiment with corresponding node’s status.

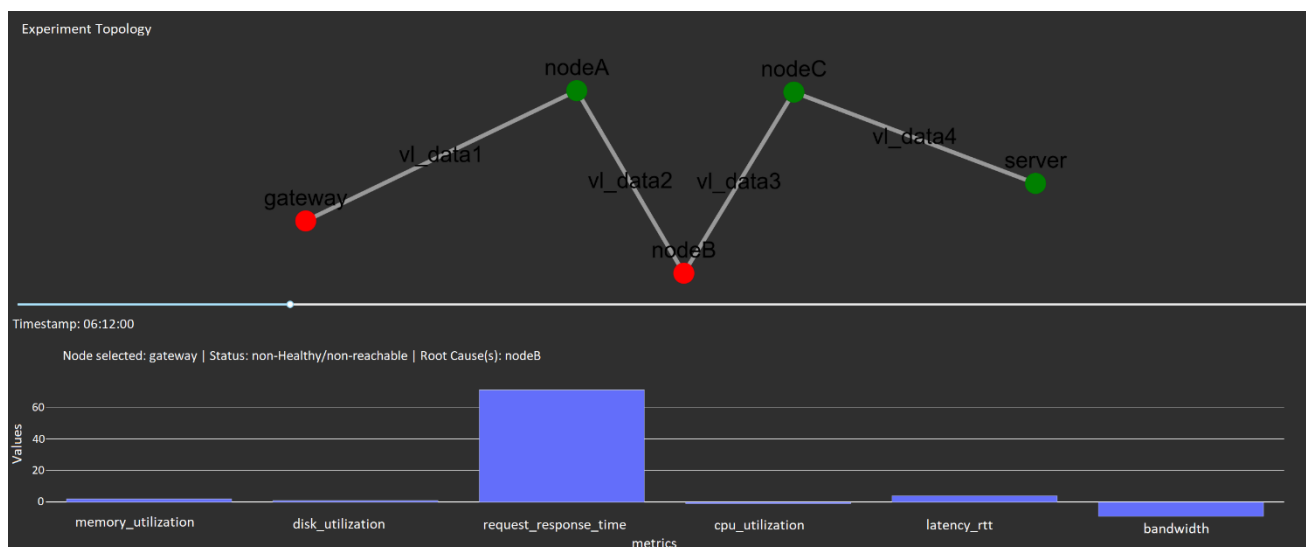


Figure 10: Experiment topology and metrics weights graphs

The second algorithm utilized in the PD operations cycle is the Root Cause Analysis using adjacency lists. This algorithm uses the output generated from the SOM algorithm in order to detect the cause of the unhealthy status of an element. Information regarding all the metrics and KPIs monitored, their effect on the status of each node, and the topology of the service data flow are displayed in detail on the graph seen in Figure 10.

Examining these graphs, the Vertical can easily identify if the cause of the performance degradation is a metric of the examined node or another node in the same data flow that is propagating it.

The last graph in the PD results segment of the Validation report, seen in Figure 11, pertains to service profiles. These profiles are the performance of each KPI monitored during the experiment in respect to the allocated resources for the various elements deployed for the service.



Figure 11: Service KPI profiles

Data from executions of the same service but with different deployment resources for the elements deployed are aggregated and displayed in this graph. This provides significant insights to the Vertical regarding the expected performance of the tested service on the specific number of resources. This information is indicative of the performance to resources ratio and can be combined with other financial information, possibly regarding the costs, in order to make decision regarding the expansion or downsizing of a deployment.

5 Performance diagnosis

The Performance Diagnosis (PD) mechanism, is responsible for collecting information during an experiment, parsing that information, analysing it, and producing insights on the performance of the various elements that comprise the experiment. It is also responsible for detecting anomalies in the behaviour of the various elements as well as identifying the root cause of possible problems or performance degradations during the experiment. As a component of the IWL, like the Data Collection Module and the Monitoring module, it is developed with the whole architecture of the platform in mind and thus is built around utilizing the previously mentioned components.

5.1 Internal architecture

The PD module’s operations can be organized in three groups:

- Information retrieval;
- Pre-processing;
- Analysis.

Information collection, related to the first stage of the performance diagnosis operations, is managed by the Data Collection Module (DCM). For the task of collecting and publishing data to the local and central broker, probes and data shippers are used, respectively. The flow of information during the PD cycle of operations can be seen in Figure 12.

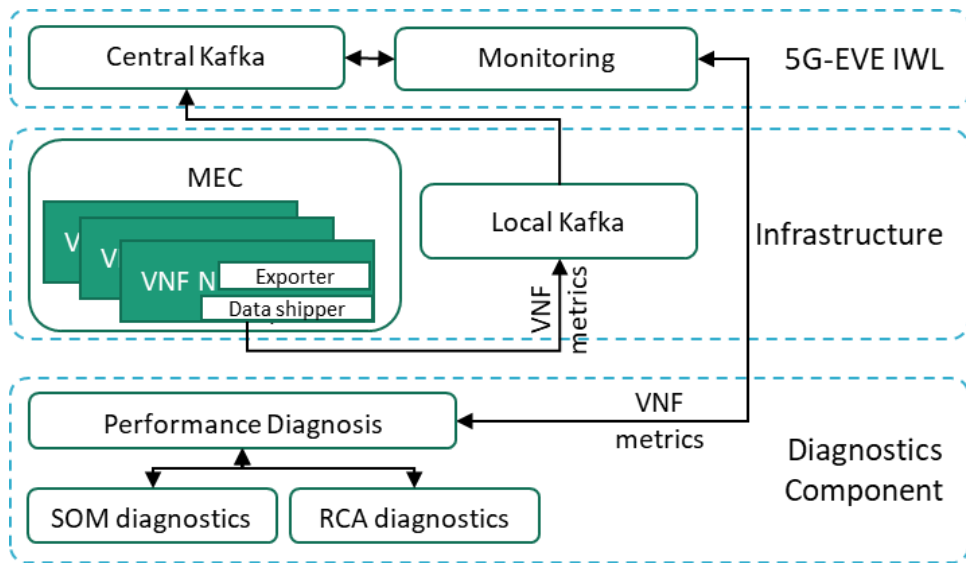


Figure 12: Performance Diagnosis information flow

To avoid functional overlap with other components like the RAV, the PD module is instructed when and where we need to retrieve the information generated during the execution. The next step is pre-processing. All the ingested information, comprising of metrics and KPIs are then parsed and transformed into appropriate structures, retaining temporal information that can be fed to the analysis algorithms to establish a behaviour timeline for the various elements. Finally, the selected algorithms, implementing SOMs and RCA, are used to perform the necessary analysis on the resulting data to detect any faults or degradation on the observed performance of the used infrastructure. The implemented analysis functionality is modular, allowing chaining different algorithms together for more in-depth analysis and conclusion. In this case, the results of the anomaly detection module are sent to the RCA module to determine the root of the detected anomaly. The workflow of the Performance Diagnosis operations, as described in D5.5, can be seen in Figure 13.

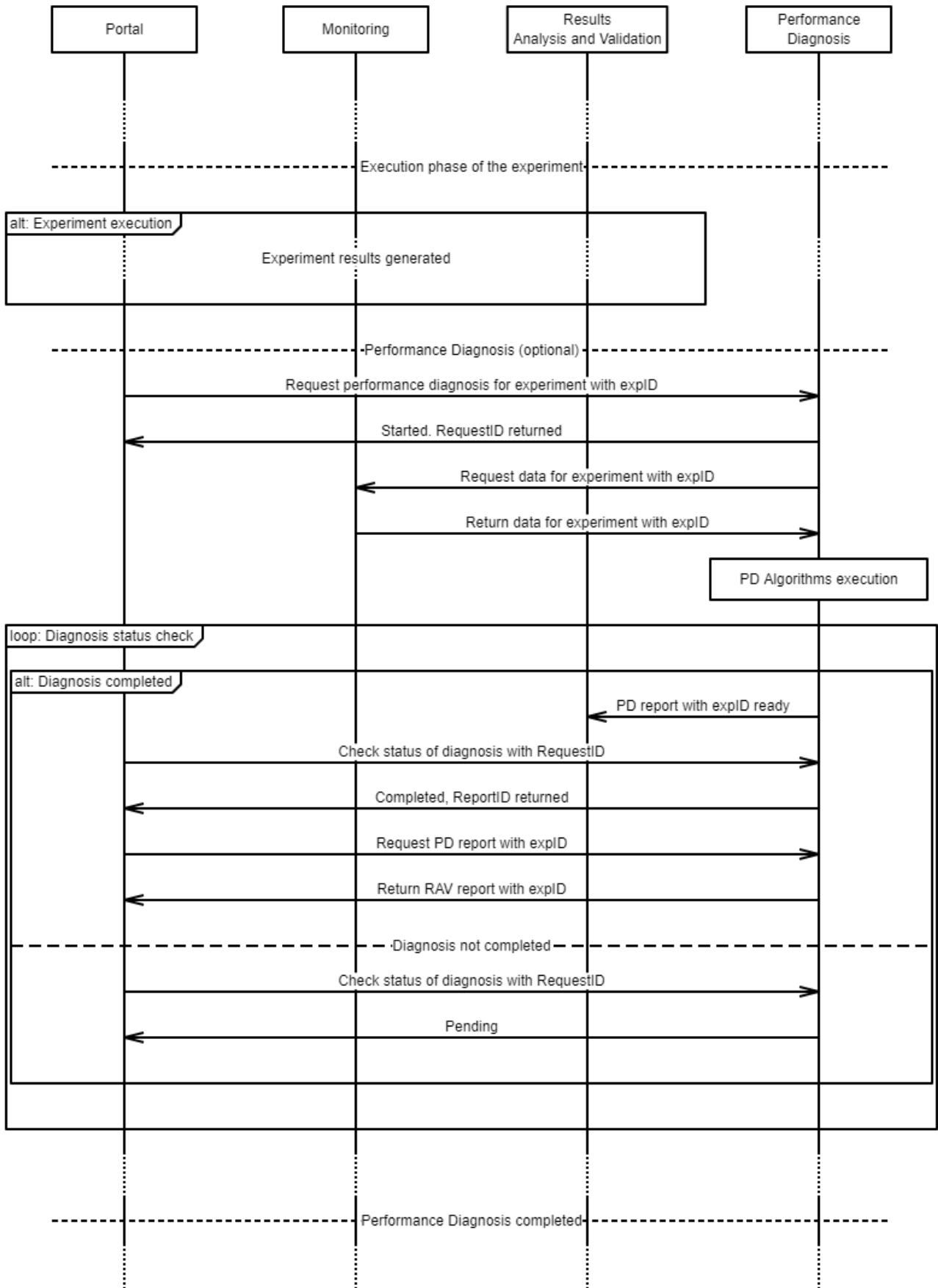


Figure 13: Performance Diagnosis workflow

The PD module follows the same modular approach as the RAV module thus being able to handle multiple experiments in parallel. Configuration and operational control of the module is handled by the corresponding RAV module instance, since their operation are closely tied together regarding the experiment information, stages, and results generation. This is accomplished using interfaces between the RAV and PD modules.

5.1.1 PD - RAV interface

As previously mentioned, this interface is used to configure and control the state of the PD module following the signalling that the RAV receives from the EEM. The API has been slightly updated from its previous version reported in D5.5. The configuration and execution requests, that were previously a single request, have now been split in separate requests allowing better synchronization of the various stages of an experiment between the components. The update API endpoints can be seen in Figure 14.

GET	<code>/experiment/{expID}/{executionID}/report</code>	request the report for a performance diagnosis instance	↩
GET	<code>/experiment/{expID}/{executionID}/diagnose</code>	request to start the performance diagnosis instance	↩
GET	<code>/experiment/{expID}/{executionID}/status</code>	request the status of the performance diagnosis request	↩
POST	<code>/experiment/{expID}/{executionID}/request</code>	register a performance diagnosis request	↩
DELETE	<code>/experiment/{expID}/{executionID}/remove</code>	delete the performance diagnosis report	↩

Figure 14: Performance Diagnosis module API

Management of the PD results and integration with the final report is also handled by the RAV and Reporting submodules so no additional interfaces are needed.

5.2 Implementation details

The PD component, as described in previous deliverable D5.5 [4], can detect anomalies in the behaviour of various elements of an experiment, identify the root cause of performance degradations and offer insights regarding the observed performance by applying post-process analytics on the collected KPIs. Most of the functionalities presented in D5.5 have been consolidated in a single module that requires external configuration and ingestion of the generated metrics and KPIs to perform the PD operations cycle.

The PD component upon receiving a registration request for an experiment, from RAV, initializes a new PD instance, retrieves the experiment topology from RAV, creates the necessary data structures, and prepares the consumers for the metrics and KPIs. For the efficient detection of the root cause of a possible performance degradation the PD algorithms require information in order to match the service topology retrieved from the service NSD to the metrics and KPIs generated. That is accomplished by matching the prefix of the ‘flavour_id’ key in the properties of each VNF to the prefix of the metric or KPI name. This string, by convention, is the name of the node that each VNF is implementing. Ignoring this convention does not allow the PD algorithms to match the metrics and KPIs messages to each node of the topology and thus it would be unable to analyse the topology in search for a root cause of a problem.

During the experiment execution, all the generated metrics and KPIs are collected and organized to create a behaviour timeline for all the elements of the experiment that generate them. This process results in a list of vectors that include all the available information for each element for every monitored moment of the experiment execution. These vectors are then fed to the PD algorithms to perform the actual performance diagnosis. The basic mechanisms used by this module for that purpose are Self-Organizing Maps and Root Cause Analysis using adjacency lists.

Additionally, the Reporting submodule of RAV has been extended in order to include the PD results in the Validation report, when PD is requested by the Vertical. In this way, the Vertical will receive a single detailed report with all the information and insights generated during the experiment execution.

5.2.1 Self-Organizing Map

A machine learning technique based on Self-Organizing Maps (SOM) has been developed to detect abnormal behaviour of the nodes of a deployed network service. In our case these nodes are the VNFs that compose a NS.

SOM is an alternative of traditional artificial neural networks, composed of a set of neurons, often arranged in a 2D rectangular or hexagonal grid. Each neuron is characterized by a reference vector, known as weight. The main objective of SOM is to project a set of high-dimensional input data onto a low-dimensional space, the topological map, by preserving the topological properties of the data. This projection is achieved after a learning process. SOM's learning is based on unsupervised learning technique, meaning that the learning process is not guided by predefined labelled outcomes, but through the discovery of underlying hidden patterns in the data set. More specifically, the learning of the SOM involves three main processes: (1) the Competition, (2) the Cooperation and (3) the Adaptation process. During the first process, a distance is calculated to compare the similarity between the input vector sample and the weight vector of each neuron. The neuron with the weight more similar to the input vector is declared as the winner, known as Best Matching Unit (BMU). In the second process, the BMU determines the spatial location of the cooperation neurons, which are its neighbours. These neurons share common features and activate each other to learn from the same input. Finally, through the Adaptation process, the activated neurons become more like the input samples resulting in an increased chance of these neurons to respond to similar input data in the future.

The procedure we followed to find possible abnormal behaviour of the nodes is described by the following steps.

- *Step 1 - Data collection*: The input data, which are the VNFs' performance metrics, are collected and stored as vectors.
- *Step 2 - Initialization of the topological map*: The topological map is composed of M neurons. Each neuron is represented by a weight vector. The weight vectors are initialized by randomly selected from the training data set, which consists of the performance data in normal condition.
- *Step 3 - Competition learning process*: For each input vector, the Euclidean distance between the weight vectors of the neurons is examined. The winning weight vector that is most similar the input vector is selected as BMU. This step is repeated until all input vectors are examined and assigned to their winning neurons.
- *Step 4 - Cooperation learning process*: Through this step, all the weight vectors are updated. Steps 3 and 4 are repeated until several iterations have passed.
- *Step 5 - Detect faults*: After training, the quantization error e_{QE} for each input vector is calculated. The quantization error is the distance between the input vector and the weight vector of the BMU. If the condition (5) is met, the input vector instance is considered abnormal: $e_{QE}(x_i) \geq \varepsilon$ (5), where ε is a predetermined threshold.

Step 1 to step 4 address the SOM learning process as described in the paragraph above. Finally, step 5 deals with finding faults in nodes.

SOM was chosen over other algorithms, as it is able to combine and analyse various performance metrics (such as CPU utilization, memory utilization, disk I/O, network interface I/O, request latency etc.), to learn their underlying patterns and ultimately make decisions about the health status of VNFs without prior knowledge of specific thresholds.

5.2.2 Root Cause Analysis

A low complexity Root Cause Analysis algorithm has been implemented to localize a problematic node that causes performance degradation to other nodes and the deployed service in general. The basic principle of the

algorithm is to check the reachability between the nodes (in our case, instances that belong to Virtual Network Functions), using each node’s health status that is determined by the fault detection performed in the Self Organizing Maps module, and the network topology. The workflow of the algorithm can be seen in Figure 15.

The health status of a node can be dependent on the status of another, non-healthy node in the network path. This could lead the SOM module to detect both nodes as non-healthy, based on their metrics. The RCA algorithm can provide extra information regarding the dependency described above, enabling fault localization, in terms of identifying the node that affects other nodes’ health status.

The algorithm uses the health status provided, to label the service nodes as “Up” or “Down”. An extra label “Unknown” is applied for the “Down” nodes that may be affected by other respective nodes. This is performed using an adjacency list that is obtained from the network topology. Specifically, an n-node undirected graph represented as an adjacency list is created using the virtual links of the topology. The nodes are numbered from 1 to n. The adjacency list is an array (size n) of linked lists where index i of the array contains the linked list of all the nodes directly connected by a network link to node i. Next, each “Unknown” node is examined to identify the Down nodes that may cause the node’s non-healthy state. The algorithm’s output is one list per “Unknown” node that contains the “Down” node(s) identified as the root cause for the respective node’s performance issues. The workflow described above is structured in two parts of the algorithm: Part A determines the status of individual elements in the network (“Up”, “Down”, “Unknown”), Part B creates the Root Cause lists for the “Unknown” nodes.

The developed algorithm implements a complementary tool that provides extra analysis capabilities to the Performance Diagnosis module. It exploits the advantages of the SOM tool (unsupervised deployment, heterogeneous input) and utilizes information provided by the 5G EVE platform to follow the PD module’s basic concept: perform network diagnoses of a deployed service without extensive knowledge of the service’s functionality, metrics, and overall usage. Thus, it comprises a useful tool that complements the final Performance Diagnosis report produced for a performed experiment of a network service in the 5G EVE portal.

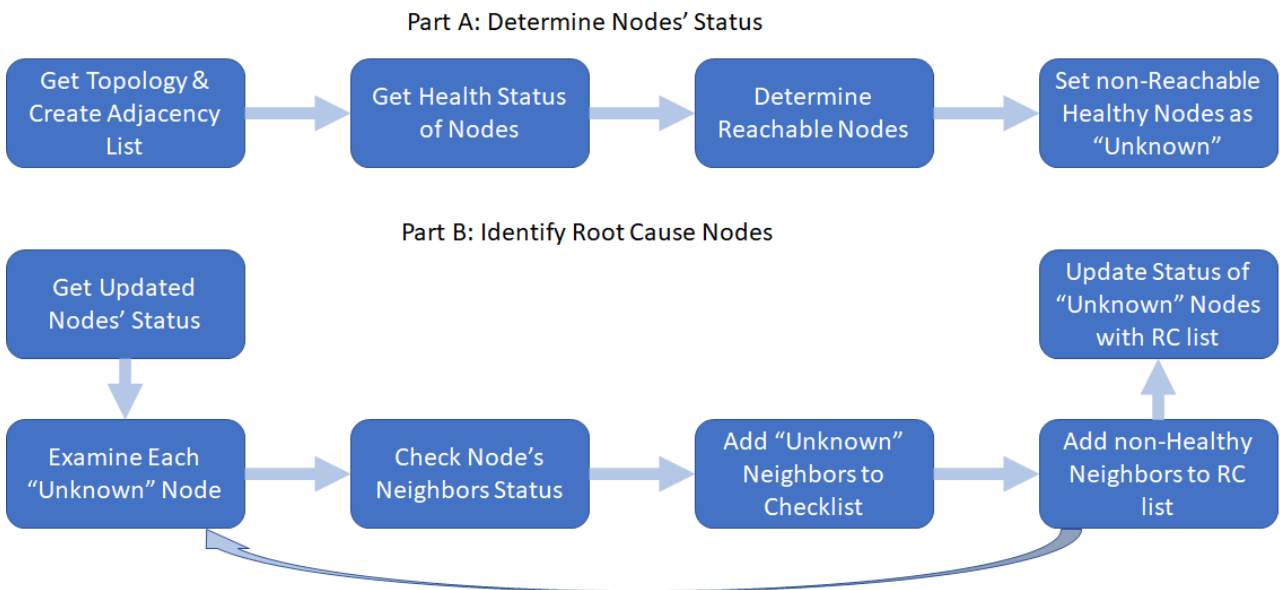


Figure 15: Workflow of the RCA algorithm

5.3 Performance Diagnosis Algorithms Evaluation

Combination of the two algorithms was evaluated and further used in the actual deployment of 5G EVE performance diagnosis component. First the individual algorithms has been evaluated in order to verify their correct operation. Each algorithm has been tested in a separate environment based on the requirements. Finally, the combination of the algorithms were tested on a setup deployed using the 5G EVE platform, in this case one of the use cases in the Greek site.

5.3.1 SOM Algorithm Evaluation

To evaluate the effectiveness of the SOM-based fault detection approach we feed the SOM algorithm with randomly generated data in vector format. These generated data simulate the VNF performance metrics, which are CPU usage, memory usage, disk usage, and request latency. The VNF is considered having an abnormal behaviour when unusually high CPU and memory utilization is generated, as shown in Figure 16.

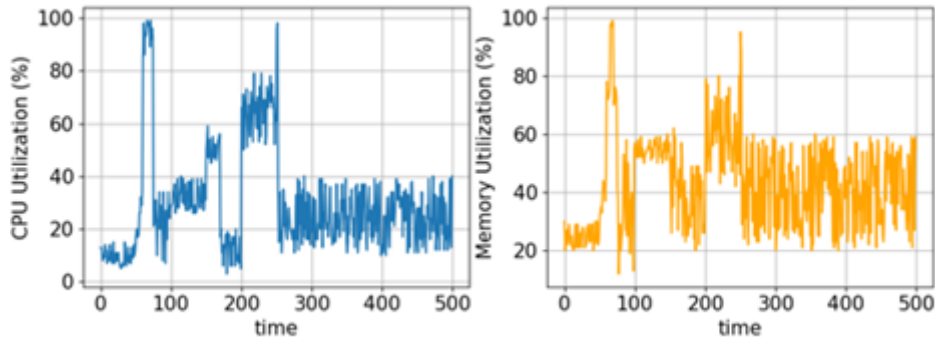


Figure 16: Abnormal node behaviour metrics

Then, the learning process of the SOM algorithm was performed using data that were composed only with metrics under normal conditions (Figure 17). Then a set of normal and abnormal input metrics were given to the trained model, which we evaluated for its capability to detect unusual node behaviour. Initially, we determined the optimal threshold parameter (ϵ) in terms of SOM fault detection accuracy.

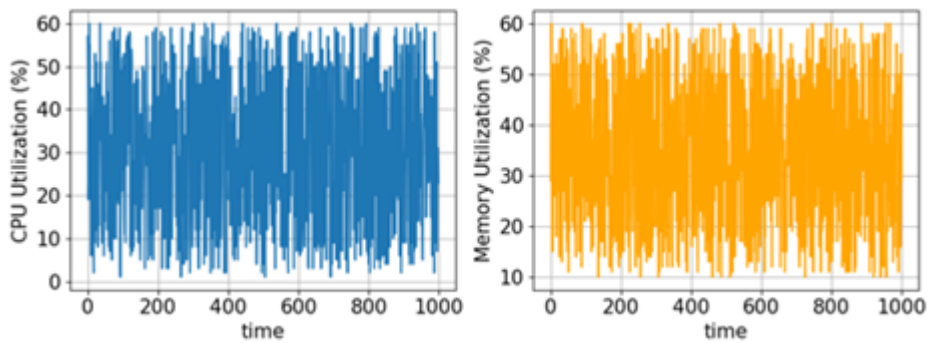


Figure 17: Normal node behaviour metrics

Figure 18(a) illustrates the results of the threshold investigation. As shown in Figure 18 (a), the f1-score, which can be interpreted as the weighted average of the true positive rate and the positive predicted value, decreases as the threshold increases. This means that smaller threshold led to more accurate results. Adjusting the threshold at the value of 0.1, we quantified the SOM performance using the Receiver Operating Characteristic ROC curve.

In Figure 18(b) the ROC curve is presented, which illustrates the performance of the SOM with respect to the false positive and true positive rates. The true positive rate represents the ability of SOM-based anomaly detector to correctly detect failure on VNFs. Also, in Figure 18 (b) the true positive rate is 97% while the false positive rate is less than 1%. These results indicate that the SOM detects failures with high accuracy.

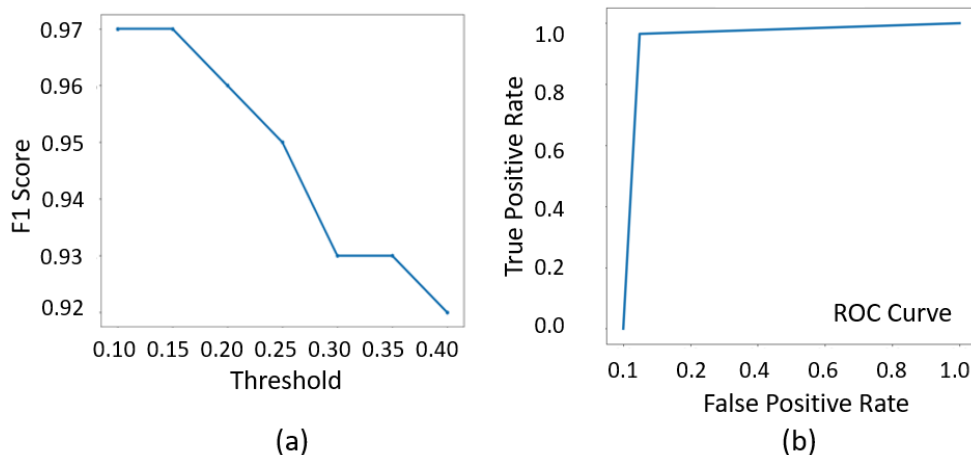


Figure 18: (a) Relation between f1 score and the fault detection threshold and (b) SOM algorithm accuracy

5.3.2 RCA Algorithm Evaluation

To evaluate the operation of the Root Cause Analysis algorithm as a standalone module, a respective testbed was developed. Using ContainerNet, a virtual network emulator was deployed that utilizes the realistic network emulator of MiniNet (to create virtual controllers, switches, and routers) and Docker containers as hosts. As a first step, a tree topology was created as shown in Figure 19.

To determine the initial health status of each node, a set of user-defined thresholds was used to determine the validity of each metric. The metrics collected were selected as follows. For the system metrics, CPU, RAM, and disk utilization were measured for each host. For the network metrics, Tx and Rx network bandwidth was measured in each nodes network interface. Furthermore, network latency was measured for each node when communicating with the node in the upper level (i.e., node 0 latency with node 16, node 16 with node 15, etc). To collect a metric in the application level, each node was equipped with a simple flask server, to send and receive API requests and measure the respective response time, labelled as application latency).

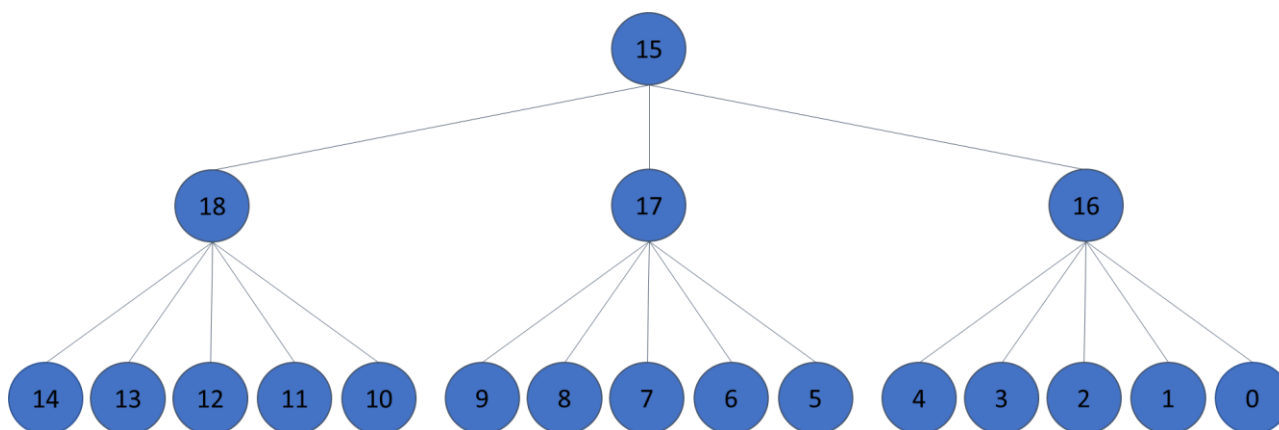


Figure 19: Topology used for the RCA algorithm's evaluation.

To validate the correct fault localization by the algorithm, fault injection was introduced to the nodes 16, 17, 18. Specifically, a randomly selected node of the three was stressed with the use of CPU and RAM heavy loads. As a result, highly increased application latency values were observed in the nodes connected to each of the three fault injected nodes.

To evaluate the algorithm's efficiency, nodes 0-14 were configured to perform requests between them, through nodes 15-18. The algorithm was able to identify the fault injected nodes as non-healthy, i.e., "Down", due to their system load and the nodes with high application latency values as non-healthy as well, characterizing them as "Unknown", due to their non-reachable status. The test scenario described was implemented for 1 hour, with the nodes sending requests every second, and the fault injected node changing between nodes 16, 17 and 18 every 20 seconds. The testing results showed overall accuracy of nearly 80%, meaning that for each request

sent, the node responsible for the high response time value was either identified, or included in a list of possible root cause nodes with a probability of 79,7%.

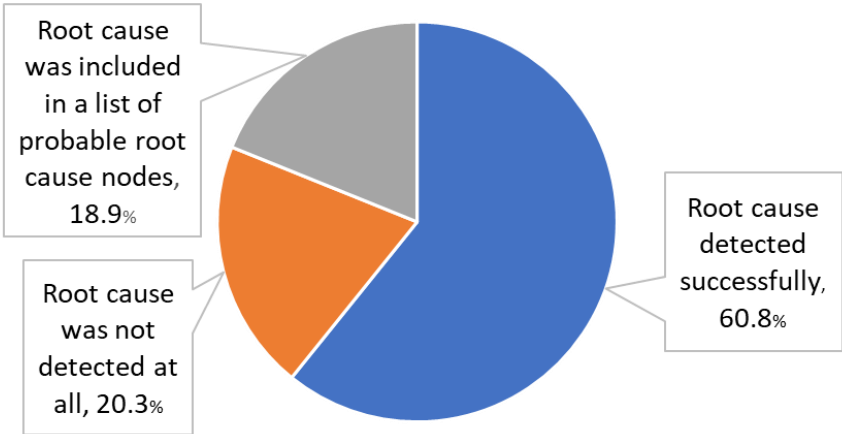


Figure 20: RCA algorithm's evaluation results

The results shown in Figure 20 can be considered successful, considering the simulated scenario’s functionality and the testbed’s characteristics. Firstly, the workload assigned to each of the “relay” nodes (16-18) was considerable, not only because of the fault injection producing high system load, but because of the increased number of requests and responses those nodes had to retransmit. Considering the nodes were containers with minimum resources, it is possible that some of the nodes were actually non-healthy even without purposeful system load introduced to them. Moreover, the nature of ContainerNet was causing relative instability. Although each node’s system resources were seemingly independent, the use of dockers running on the same system means that each node always had a relative dependence to the resources of the host system.

Considering that the algorithm was previously tested using random “Up” and “Down” assignments, without collecting actual metrics, and the results showed 100% accuracy on similar topologies, it was decided that the algorithm could be successfully deployed alongside a more complex solution. In that context, the SOM component was deployed to have an accurate decision on the nodes’ health status, and the RCA module was embedded to the Performance Diagnosis tool to perform fault localization, provided the SOM module’s output.

5.3.3 Evaluation of the SOM and RCA algorithms on the 5G EVE platform

To evaluate the deployment of the two algorithms on a 5G EVE Network Service, the deployment of a WINGS Application was simulated, with the topology shown in Figure 21.

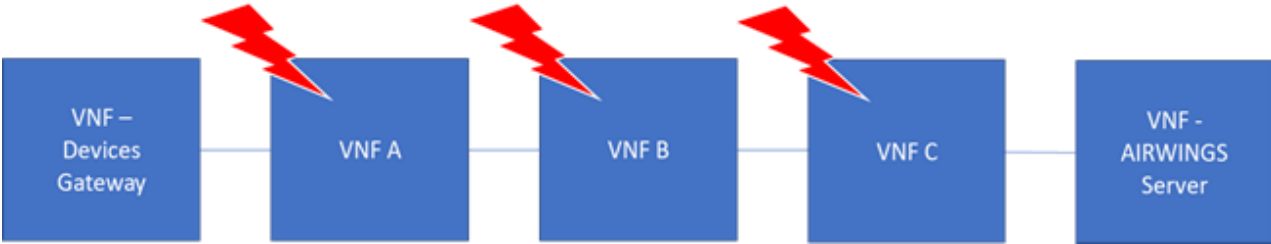


Figure 21: AirWINGS application deployed as a Network Service.

The Gateway-VNF simulates several devices sending requests to the AirWINGS server, through three VNFs used as relays. Fault injection is introduced to one relay at a time, stressing either the VNF’s CPU or RAM. The evaluation of each algorithm is carried through using the same techniques described above.

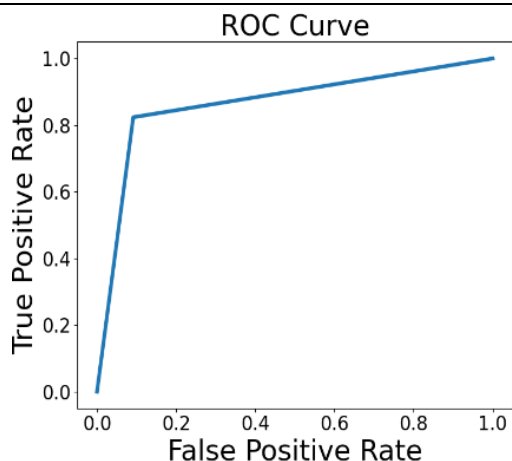


Figure 22: Accuracy of the deployed SOM algorithm

As shown in Figure 22 the true positive rate is nearly 80% while the false positive rate is nearly 10%. This means that SOM algorithm was able to detect the injected faults with 80% accuracy. The accuracy decreased compared to the results obtained from the tests that were conducted with the standalone SOM algorithm. This is mainly due to the sharp fluctuations of the values of metrics and the fact that the results come from a real deployed network service.

Regarding the evaluation of the RCA algorithm, the effect of fault injection in the relay VNFs on the response time measured for the requests sent by the Gateway VNF to the AirWINGS Server was examined. Due to the topology’s linear structure and the reduced number of nodes-VNFs, the algorithm was able to detect every stressed relay causing increased Application Latency values (i.e., response times) in the Gateway’s metrics. Each time the Gateway was not reachable from the Server and vice versa, the increased Application Latency values led to the characterization of the Gateway as “Unknown”, adding the extra information regarding the Root Cause Node to the final Performance Diagnosis Results.

Overall, the results indicate that the SOM algorithm can detect failures with high accuracy and the RCA algorithm succeeds in exploiting the SOM output and correctly identifying a significant number of root cause nodes. After the evaluation experiment execution is completed, the validation report is generated. The various segments of the validation report generated from this test run can be seen in Figure 23 to Figure 26.

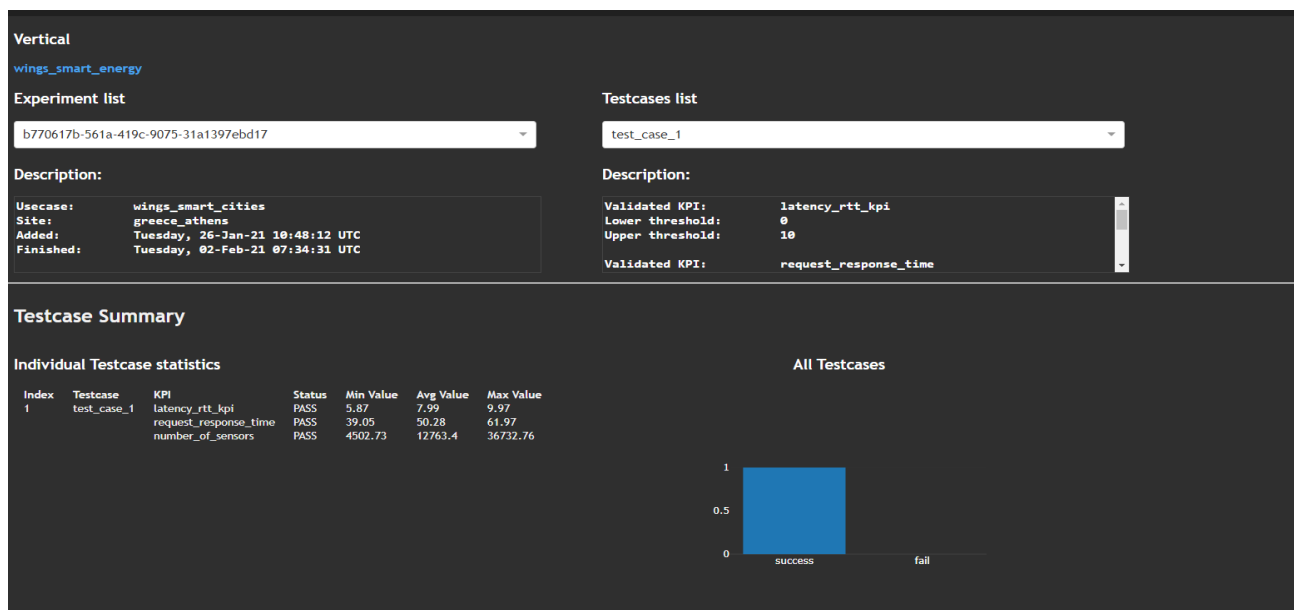


Figure 23: General experiment information and KPI statistics

The general information regarding the experiment, the testcase and KPI statistics can be seen in the Figure 23.

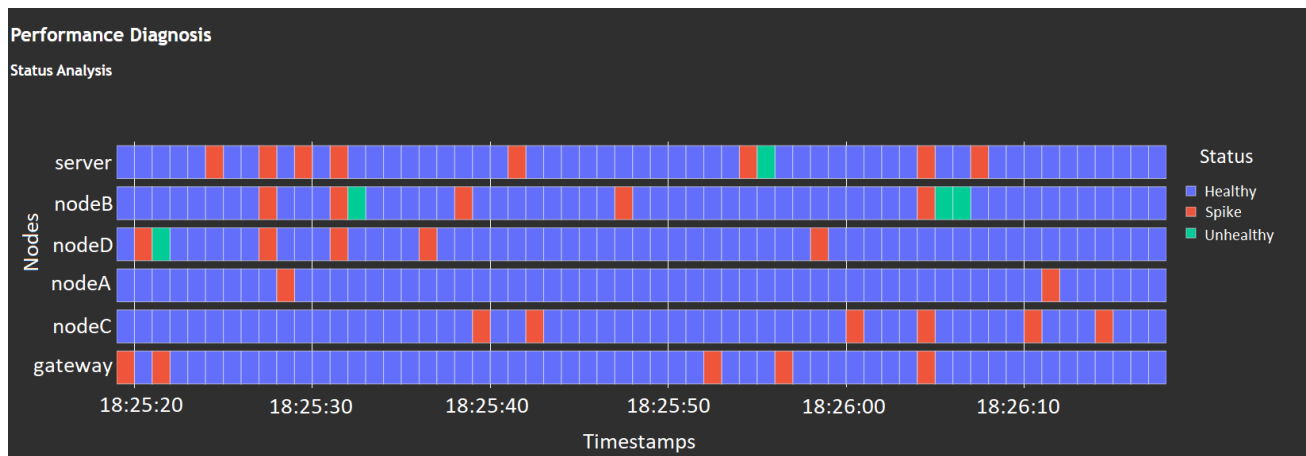


Figure 24: Node status timeline

The first part of the PD results, seen in Figure 24, consists of the status timeline of the nodes deployed for the experiment as generated from the SOM algorithm. The second part of the PD results, seen in Figure 25, consists of the topology graph. On that graph, upon selecting an unhealthy node the graph below it appears, displaying the weights of each metrics on the node status decision as well as a list with possible root causes for the unhealthy status of the node.

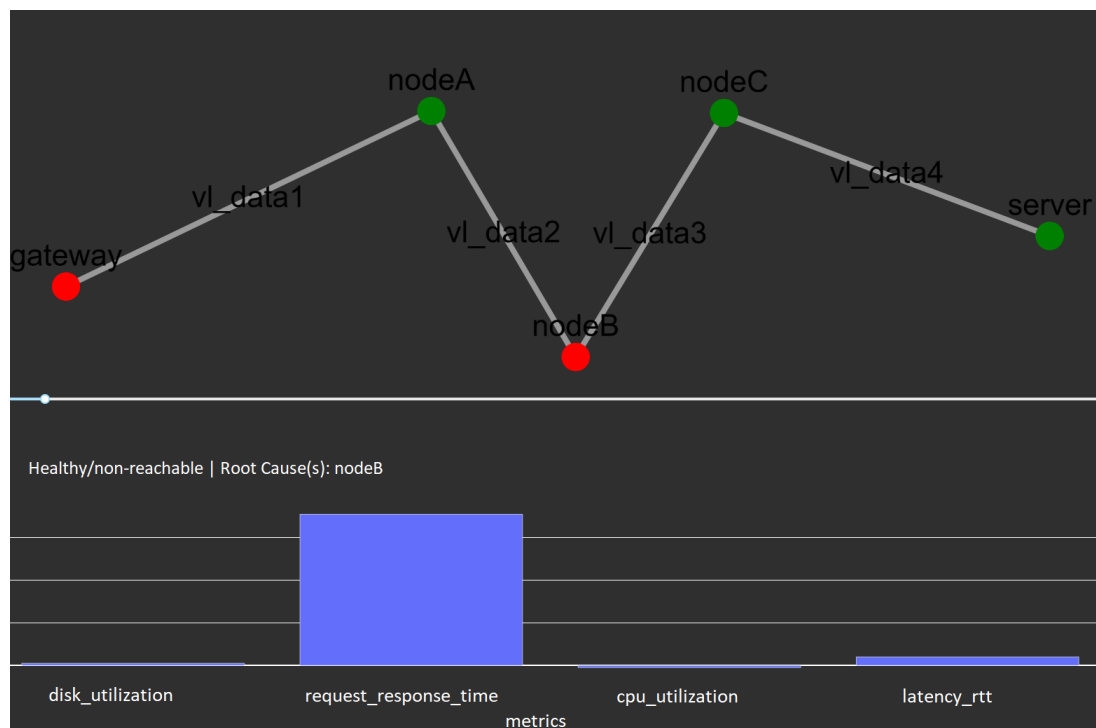


Figure 25: Experiment topology and metrics weights graphs

Finally, on Figure 26, the profile for each KPI of the service is displayed for each of the tested deployments. Some additional statistics are also generated such as the percentage difference of these KPIs between the various deployments.



Figure 26: Service KPI profiles

6 Conclusion

In this deliverable, the final versions of the testing and validation components participating in the final version of the 5G EVE platform are presented. These components named “Experiment Execution Manager”, “Results Analysis and Validation” and “Performance Diagnosis” are responsible for the execution and the lifecycle management of the experiment, the analysis of the collected metrics, the validation of the generated KPIs and finally the generation of the final reports, which are made available to the experimenters through the 5G EVE Portal.

In the document, for each component, the internal architecture, the procedures and the implementation details are presented and explained, as well as the extensions and additions realised compared to the previous versions reported in previous WP5 deliverables. In detail, the final version of the “Experiment Execution Manager” component is similar to the previous version reported in D5.2 [2] and D5.5 [4], with some additions related with the support of multi-site use cases and performance diagnosis support. The final version of “Results Analysis and Validation” has some extensions compared to the previous version reported in D5.5 [4], related with the reporting methodology and the support and illustration of performance diagnosis results. Finally, the “Performance Diagnosis” component has some small extensions to the previous version reported in D5.5 [4], mainly related with the development and integration of extension on SOM based algorithms and RCA algorithms, as well as updates on the outcomes (insights) of the performance diagnosis process.

References

- [1] 5G EVE D5.1: “Disjoint testing and validation tools”, April 2019, <https://zenodo.org/record/3625619#.YILsSugzaUk>
- [2] 5G EVE D5.2: “Model-based testing framework”, December 2019, <https://zenodo.org/record/3628341#.YILsUOgzaUk>
- [3] 5G EVE D5.3: “Testing environmental conditions document with first version of testing and validation suite”, June 2020, <https://zenodo.org/record/3946265#.YILsWugzaUk>
- [4] 5G EVE D5.5 “Performance diagnosis mechanism and reporting methodology document”, June 2020, <https://zenodo.org/record/3946255#.YILsW-gzaUk>
- [5] 5G EVE D3.4: “*Second implementation of the interworking reference model*”, June 2020, <https://zenodo.org/record/3946323#.YILsa-gzaUk>