5G European Validation platform for Extensive trials

Deliverable D5.2
Model-based testing framework

Project Details

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

Deliverable Details

Deliverable WP:	WP5
Deliverable Task:	Task T5.2
Deliverable Identifier:	5GEVE_D5.2
Deliverable Title:	Model-based testing framework
Editor(s):	Juan Rodriguez
Author(s):	Lourdes Luque, Luis Miguel Contreras, Juan Rodriguez (TID), Christos Ntogkas, Evangelos Kosmatos, Kostas Trichias, Panagiotis Demestichas, Nelly Giannopoulou, Orestis Zekai, Vera Stavroulaki, Vassilis Laskaridis (WINGS), Matteo Pergolesi, Mauro Femminella, Paolo Giaccone, Carla Fabiana Chiasserini, Claudio Casetti (CNIT), Gino Carrozzo, Elian Kraja, Giada Landi, Leonardo Agueci (NXW), Arturo Azcorra, Ginés García, Jaime García, Carlos Guimaraes, Winnie Nakimuli (UC3M), Marios Bougioukos (NOKIA-GR)
Reviewer(s):	Jaime Ruiz (NOKIA-ES), Ramon Perez (TELC)
Contractual Date of Delivery:	31/12/2019
Submission Date:	20/12/2019
Dissemination Level:	PU
Status:	Final
Version:	v1.00
File Name:	5GEVE_D5.2_v1.00

Deliverable History

Version	**Date**	**Modification**	**Modified by**
ToC_v0.1	*13/09/2019*	*ToC proposal*	*Juan Rodriguez*
ToC_v0.2	*24/09/2019*	*ToC modifications and sections assignments*	*Juan Rodriguez*
ToC_v0.3	*14/10/2019*	*Final sections assignment*	*Juan Rodriguez*
v0.1	*25/10/2019*	*Initial contributions*	*Juan Rodriguez*
v0.2	*19/11/2019*	*Integration of contributions to sections 1, 2 and 3*	*Juan Rodriguez, from partners contributions*
v0.3	*25/11/2019*	*Additional contributions to all sections*	*Juan Rodriguez, from partners contributions*
v0.4	*29/11/2019*	*Final contributions in all sections; version for review*	*Juan Rodriguez, from partners contributions*
v0.9	*16/12/2019*	*Integrated comments from reviews; version for QA*	*Juan Rodriguez, from reviewers comments*
v0.91	*17/12/2019*	*Full list of authors completed*	*Juan Rodriguez*
v0.92	*18/12/2019*	*QA review*	*Kostas Trichias*
v1.00	*20/12/2019*	*Final version*	*Juan Rodriguez*

Table of Contents

LIST OF ACRONYMS AND ABBREVIATIONS ... VI

LIST OF FIGURES ... VIII

LIST OF TABLES ... IX

EXECUTIVE SUMMARY ... 10

1 INTRODUCTION ... 13

1.1 Initial context and terminology ... 13
1.1.1 Terminology ... 14
1.2 Testing-as-a-Service (TaaS) 5G EVE approach ... 15
1.3 Document objectives ... 15
1.4 Document structure ... 16

2 MODEL-BASED TESTING AND VALIDATION FRAMEWORK ... 17

2.1 Test and validation methodology ... 17
2.1.1 Test and validation framework architecture ... 18
2.1.2 Test and validation of Network KPIs ... 20
2.2 High-level testing procedure ... 23
2.3 High-level validation procedure ... 24
2.4 High-level monitoring procedure ... 25
2.5 KPI Collection Framework ... 26
2.5.1 Data Collection Manager ... 26
2.5.2 Data Shippers ... 26

3 TESTING AND VALIDATION FRAMEWORK WORKFLOWS AND APIS ... 28

3.1 Experiment Design workflow and APIs ... 28
3.1.1 Test Case Blueprint information model ... 28
3.1.2 Metrics ... 29
3.1.3 KPI ... 30
3.1.4 Specific workflows ... 30
3.2 Test Plan templates ... 34
3.2.1 Update on high-level test plan templates ... 34
3.2.2 Low-level test plan templates ... 37
3.3 Experiment Preparation workflow ... 39
3.4 Experiment Execution workflow and APIs ... 39
3.4.1 Experiment Execution workflow: testing, validation and monitoring ... 39
3.4.2 Experiment Lifecycle Manager (ELM) and Experiment Execution Manager (EEM) API ... 42
3.4.3 Experiment Execution Manager (EEM) and Runtime Configurator API ... 47
3.5 Internal workflows and APIs ... 48

4 DESIGN OF THE TESTING AND VALIDATION FRAMEWORK COMPONENTS ... 50

4.1 Experiment Execution Manager (EEM) ... 50
4.2 Results Analysis & Validation (RAV) ... 52
4.2.1 Analysis & Validation ... 52
4.2.2 Reports ... 53

5 UPDATE OF TESTING AND VALIDATION RESULTS ... 55

5.1 Application in the ASTI use case ... 55
5.2 Application in the Segittur use case ... 57
5.3 Application in the WINGS Utilities use case ... 61

6 CONCLUSIONS AND NEXT STEPS ... 63

REFERENCES ... 65

ANNEX I: HIGH-LEVEL TEST PLAN TEMPLATES – AVAILABLE RESPONSES ... 66

Annex I.1: Experiment template for Utilities use case (Greece) ... 66

ANNEX I.2: EXPERIMENT TEMPLATE FOR INDUSTRY 4.0 USE CASE (SPAIN) 70
ANNEX I.3: EXPERIMENT TEMPLATE FOR MEDIA & ENTERTAINMENT USE CASE – VIRTUAL VISIT OVER 5G (FRANCE).. 73
ANNEX I.4: EXPERIMENT TEMPLATE FOR SMART TOURISM USE CASE (SPAIN)........ 77
ANNEX I.5: EXPERIMENT TEMPLATE FOR TRANSPORT USE CASE (SPAIN)........ 80

ANNEX II: LOW-LEVEL TEST PLAN TEMPLATES – AVAILABLE RESPONSES 83

ANNEX II.1: EXPERIMENT TEMPLATE FOR INDUSTRY 4.0 USE CASE (SPAIN) 83

ANNEX III: EXPERIMENT EXECUTION WORKFLOW; TESTING, VALIDATION AND MONITORING PROCEDURES........ 88

List of Acronyms and Abbreviations

Acronym	Description
ABR	Adaptive Bit Rate
ACL	Access List
AGV	Automatic Guided Vehicle
API	Application Programming Interface
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BBU	Base Band Unit
CB	Context Blueprint
CCTV	Closed Circuit Television
CD	Context Descriptor
COTS	Commercial Off-The-Shelf
CP	Control Plane
CPU	Central Processing Unit
DB	Database
DCM	Data Collection Manager
DHCP	Dynamic Host Configuration Protocol
DL	Downlink
E2E	End-to-end
EEM	Experiment Execution Manager
ELK	Elasticsearch, Logstash, Kibana (Stack)
ELM	Experiment Lifecycle Manager
eMBB	Enhanced Mobile Broadband
eNB	Enhanced Node B
EPC	Evolved Packet Core
ExpB	Experiment Blueprint
ExpD	Experiment Descriptor
FoV	Field of View
gNB	Next Generation Node B
GOP	Group Of Pictures (MPEG)
GPS	Global Positioning System
GUI	Graphical User Interface
HMD	Head Mounted Display
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IT	Information Technology
IWL	Interworking Layer
KPI	Key Performance Indicator
MAC	Medium Access Control
MEC	Multi-access Edge Computing
NFV	Network Functions Virtualization
NFVO	Network Functions Virtualization Orchestrator
NG-RAN	Next Generation Radio Access Network
NR	New Radio
NS	Network Service
NSD	Network Service Descriptor
NSI	Network Service Instance
NSO	Network Service Orchestrator
OTT	One Trip Time
PLC	Programmable Logic Controller
PNF	Physical Network Function
PRT	Power Restoration Time
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random-Access Memory
RAN	Radio Access Network
RAV	Results Analysis and Validation
RF	Robot Framework
RRU	Remote Radio Unit
RTT	Round Trip Time
SDR	Software Defined Radio
SIM	Subscriber Identity Module
SSH	Secure Shell

TaaS	Testing as a Service
TC	Test Case [also (Linux) Traffic Control, normally in lower case: tc]
TcB	Test Case Blueprint
TcD	Test Case Descriptor
TS	Test Script
TtRF	Time to Right Frame
UC	Use Case
UE	User Equipment
UL	Uplink
UP	User Plane
UPF	User Plane Function
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URLLC	Ultra-Reliable Low-Latency Communication
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
vEPC	Virtualized Evolved Packet Core
VM	Virtual Machine
VNF	Virtualized Network Function
VoD	Video on Demand
VR	Virtual Reality
VSB	Vertical Service Blueprint

List of Figures

Figure 1: 5G EVE Testing methodology in D5.1 13
Figure 2: High-level architecture of D5.1 testing tools 14
Figure 3: 5G EVE model-based Test & Validation methodology 17
Figure 4: 5G EVE Test & Validation framework architecture 19
Figure 5: Control plane signalling during transition from Inactive to Active states 22
Figure 6: 5G EVE testing procedure 23
Figure 7: 5G EVE validation procedure 25
Figure 8: Main services and functionalities offered by Kafka 26
Figure 9: Beats information flow 27
Figure 10: Onboarding of Test Case Blueprint 31
Figure 11: High-level workflow for the definition of experiment details. 32
Figure 12: Selection of infrastructure metrics for the ExpB 32
Figure 13: Definition of KPIs for the ExpB 33
Figure 14: Selection of TcBs for the ExpB 33
Figure 15: 5G EVE Experiment Execution workflow – part 1 40
Figure 16: 5G EVE Experiment Execution workflow – part 2 40
Figure 17: 5G EVE Experiment Execution workflow – part 3 41
Figure 18: North-bound interface of EEM (part1, base application) 43
Figure 19: North-bound interface of the EEM (part2, operations) 44
Figure 20: North-bound interface of the EEM (part3, notifications & subscriptions) 44
Figure 21: EEM Finite state Machine for an experiment execution 50
Figure 22: EEM high level design 51
Figure 23: Analysis and Validation module operation 52
Figure 24: Analysis and Validation functionalities workflow 53
Figure 25: Reporting 54
Figure 26: ASTI use case testbed 55
Figure 27: Guided error of the AGV for different extra delays 56
Figure 28: Guided error of the AGV for different packet losses 57
Figure 29: Viewport change speed expected distribution and obtained base results 58
Figure 30: TtRF expected distribution and obtained base results 58
Figure 31: Number of camera changes during playback – results over 4G 59
Figure 32: Average TtRF in ms – results over 4G 59
Figure 33: TtRF distribution – results over 4G 60
Figure 34: Average bitrate – results over 4G 60
Figure 35: Testbed for the latest Utilities results 61
Figure 36: Effect of delay on the Power Restoration Time 61
Figure 37: Effect of packet loss on Power Restoration Time 62
Figure 38: Experiment Execution; testing, validation and monitoring procedures 88

List of Tables

Table 1: 5G EVE list of Network KPIs 20
Table 2: Test Case Blueprint fields with relevant description 28
Table 3: Metric fields with relevant description 29
Table 4: KPI fields with relevant description 30
Table 5: Test Plan Template versions history 34
Table 6: ExperimentExecutionRequest data type 46
Table 7: ExperimentExecutionResponse data type 46
Table 8: ExperimentExecutionSubscriptionRequest 46
Table 9: ExperimentExecutionSubscriptionResponse 46
Table 10: ExperimentExecutionNotification 47
Table 11: Runtime configurator/RAV data model for notifications service 47
Table 12: EEM – Runtime Configurator API 48
Table 13: RAV module’s configuration message fields 48
Table 14: 5G EVE testing and validation framework support of requirements 63
Table 15: Utilities UC experiment components 66
Table 16: Utilities UC meaningful KPIs 67
Table 17: Utilities UC test plan 68
Table 18: Industry 4.0 UC experiment components 70
Table 19: Industry 4.0 UC meaningful KPIs 71
Table 20: Industry 4.0 UC test plan 71
Table 21: Media & Entertainment UC experiment components 73
Table 22: Media & Entertainment UC meaningful KPIs 74
Table 23: Media & Entertainment UC test plan 75
Table 24: Smart Tourism UC experiment components 77
Table 25: Smart Tourism UC meaningful KPIs 78
Table 26: Smart Tourism UC test plan 78
Table 27: Transport UC experiment components 80
Table 28: Transport UC meaningful KPIs 81
Table 29: Transport UC test plan 82

Executive Summary

The 5G EVE Experimentation Portal is the main visible component available for experimenters willing to run their experiments on top of 5G EVE infrastructure. All the direct interaction will be done through the Portal, but the Portal is just the front-end of the Testing-as-a-Service environment that 5G EVE is creating. In particular, the testing and validation procedures are not governed by the Portal, but by the WP5 modules conforming the Testing and Validation framework to which this Deliverable is dedicated.

Along the initial section, the main objective of this framework, and therefore of this Deliverable, has been re-written from that in the Description of Action. The final definition of such objective is the design of an "**overall framework enabling and supporting automated model-based testing and validation of 5G technologies, components and services**". The reality is that the planned scope has not changed, but including both the "validation", the "automation" and the "services" concepts in the definition provided a more complete "mantra" that could singlehandedly drive the whole design process.

The Deliverable continues with the description of the methodology and the architecture of the Testing and Validation framework. Both were initially outlined in the previous D5.1 Deliverable, but the definition here included is now finished, and fully aligned with WP1, throughout its four phases: test design, test preparation, test execution & monitoring, and test performance evaluation and analysis. The four phases are very self-explanatory, but still specific tasks have to be correctly allocated, like the creation of the blueprints or descriptors, and their on-boarding onto the platform. These are finally part of the test design phase, leaving the preparation phase for the customization of templates by experimenters, and for the agreement between experimenters and site managers about the test schedule. It has to be noted, therefore, that WP5 has no activity in this second phase of the methodology.

The main architectural modules of WP5 are the **Experiment Execution Manager (EEM)** and the **Results Analysis and Validation (RAV)** module. The EEM drives the execution of tests, after they have been deployed over the available infrastructure. In particular, the EEM triggers four main procedures:

- The day-2 configuration over the deployed components, towards the Runtime Configurator
- The execution of Test Cases, in order, based on scripts, again via the Runtime Configurator
- The validation of the test results, via the Results Analysis and Validation module
- The return to initial conditions, before a new Test Case is run (via Runtime Configurator)

The second step brings the opportunity to present the **Test Scripts Repository** module, storing the scripts associated with the Test Cases, and providing them to the EEM on-demand. The third step, on the other hand, means that the RAV module has to:

- Calculate KPIs from raw metrics when there is a relationship between them
- Validate the test results, applying a validation condition to the target KPIs
- Generate the final test report (pass/fail)

Following in the Deliverable is the description of the high-level procedures for testing, validation and monitoring. The **testing procedure** has been updated from D5.1 mainly to incorporate more clearly the benchmarking testing. It has to be reminded that in the previous Deliverable we only included latency and bandwidth as part of the available benchmarking tests, which simplified things as there are very well known procedures (and tools) to measure these. Now, the procedure includes both types of testing (benchmarking – or "5G technologies and components" – and Vertical's applications), so it is more complete. Simply put, however, it can still be summarized as a loop in which the test is executed "N" times over a specific set of external conditions, and then one testing variable changes and the test is executed again "N" times. All through the test, different metrics are gathered for KPI calculation, test results validation and visualization.

The **validation procedure** is also quite straightforward, being the complexity in which type of validation conditions are supported. The KPI can be formed by a single value (e.g. packet loss), or by a list of values (e.g. experienced delay, measured every second). The validation condition will compare these values with a threshold (or a range) and if all the values are correct, then the test will be passed; otherwise it will be declared as failed.

This single occurrence determining a failed test is important for verticals, as a single occurrence event may have very important impacts (e.g. an AGV deviating more than the expected from its guide can derive in a crash). The validation conditions supported, at least initially, are the classic mathematic expressions like “equal to”, “greater than”, etc.

Finally, the **monitoring procedure** is leveraging the available modules from other WPs, in particular the data broker of the Data Collection Manager, from where the RAV module gets the required metrics, and to where it can publish the calculated KPIs (so they are eventually visualized).

An initial, still to be improved, proposal for the test and validation of Network KPIs is also included in section 2.1.2, where we focus on those which have greater impact in the framework (from the whole list generated in WP1).

Section 3 of the Deliverable focuses in workflows and API definitions. The organization follows that of the methodology, so we have included workflows and APIs separated as defined by the four phases.

Regarding the Design phase, the information model of the Test Case blueprint (which is the one in which WP5 must collaborate for its creation) is presented, including the information about scripts and testing variables that will later on drive the experiment execution. Also, it is defined how to “specify” metrics and KPIs, useful in later phases for validation, monitoring, etc. The workflows included in this section have to do, mainly, with how we make these items available to other WPs and to modules of the Testing and Validation framework.

Next, an update about the test plans is provided. The Deliverable builds on the latest version of the template to comment on what lessons we have learned from the interaction with Verticals, and how these lessons have been transferred into the templates. It has to be noted that, as of today, we already have five responses to these templates, which appear to be key in the future interaction with ICT-19 projects. Also, the first example of how these high-level templates translate into low-level templates is provided in the text, associated with the ASTI use case.

As already stated, WP5 does not participate in the Preparation phase; still, a specific comment stating this has been included in the text. Finally, and since they are very closely related, we have mixed the final two phases in the Experiment execution section, providing a very detailed workflow on all the operations that happen from the moment the experiment is deployed till the moment the test report is available for the experimenter. Items like how we are managing the Kafka topics, how we are creating the day-2 configuration, how we are launching the test scripts, how we do the monitoring, how the validation module is fed with metrics, etc. are all included in section 3.4. This section also includes the two external APIs of WP5:

- The interface between the EEM and the Experiment Lifecycle Manager (ELM), used to trigger the execution of experiments
- The interface between the EEM and the Runtime Configurator, which in fact has been defined by WP3, so the text includes the appropriate references to understand how day-2 configuration and the running of scripts is executed.

The objective of section 4 is to start commenting on the specific internal design of the two key WP5 modules. Of course, none of these implementations has yet finished, so these internal schemes are susceptible of change. Maybe a more stable information in this section is about the reporting functionality of the RAV. This reporting process is also a phased procedure:

1. The RAV generates the Test Case Validation report, dealing exclusively with the target KPI
2. The EEM generates the Test Case report, more focused on the output of the different test stages, being therefore more an operational report
3. The ELM generates the Experiment Report, which includes all the high-level parameters defining the experiment as a whole
4. The Portal generates the Scheduling Report, finally incorporating data like the selected site, the time frame of the experiment and so on.

Deliverable D5.1 also included some preliminary results based on the execution of experiments with the provided Disjoint Tools. D5.2 now updates on some of these results, in particular for three Use Cases.

In the ASTI and the Utilities use cases, a network impairment emulator has already been included in the test scenario, so the internal communications can now suffer additional delay or packet loss. Graphs showing the effect of both in the respective vertical KPIs (called "error of the guide" in ASTI and "power restoration time" in WINGS) have been included. Potential next steps in these scenarios are automating all the tests execution, being able to execute real validations, and also to include multiple effects at the same time.

In the Segittur use case, on the other hand, the main goal is to compare the performance of Virtual Reality video applications in 4G and 5G. This Deliverable includes the first measurements over 4G scenarios, together with some expectations about what to expect with 5G.

Finally, some conclusions end the document. The fact is that we have designed very much from scratch a testing and validation framework that we expect will be able to host multitude of experiments in the coming months. Obviously, there is no other next step as the complete implementation of all the features here defined. However, and just as a very quick check (which is very interesting exercise to do after any design is made), we have given ourselves scores on how we meet or not the nine main requirements which can be derived from the original "mantra". The full table is included in section 6; the summary is that, scoring from 0 to 3, our mark is above 2.5. Beyond stating that we are going in the right direction, it also helps to find where we need to improve.

1 Introduction

Deliverable D5.2 – *Model-based testing framework* – represents the main outcome of Task 5.1 – *Overall methodology framework for model-based testing* –, and specifies a testing and validation architecture capable of providing the Testing-as-a-Service (TaaS) offer expected from the 5G EVE platform and site facilities.

1.1 Initial context and terminology

In D5.1 – *Disjoint testing and validation tools* – [1], several concepts related with experimentation and validation were already outlined as part of a preliminary description effort. The objective was to explain how those disjoint tools (as a whole) could be envisioned as an early effort to provide the most important features around the Testing and Validation Framework, including a minor group of external functionalities (from WPs 3 and 4) on which this Framework relied. In other words, in [1] we were providing preliminary tools replicating to some extent the desired behaviour for the final Testing and Validation Framework. Such tools have been useful for experimenters in the first phase of the project, and have permitted a better understanding on what was correctly designed and what required some revision.

From the description point of view, three topics were included in [1] that are the basis for this Deliverable.

First, an initial version of the model-based testing methodology was outlined, helping readers to correctly position the proposed tools on the general testing and validation workflow. In this sense, and based on a joint effort with WP1, four stages were identified, as depicted in Figure 1 below.

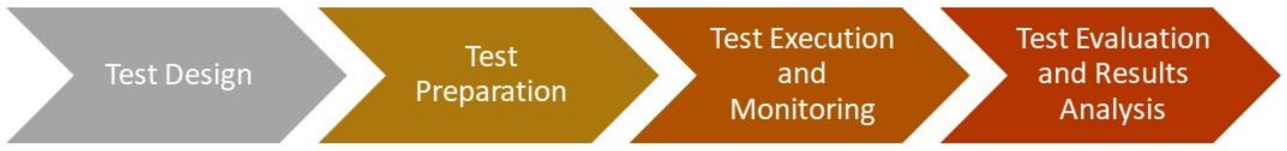

Test Design
Test Preparation
Test Execution and Monitoring
Test Evaluation and Results Analysis

Figure 1: 5G EVE Testing methodology in D5.1

While the workflow phases are very self-explanatory in the high-level, this Deliverable elaborates a little bit deeper in each one of the phases, describing the required actors, inputs and outputs, and each of the operations required to successfully execute the workflow. This extension is the content of the methodology part of section 2.1, which in turn contributes to the low-level workflow definitions of section 3.

Secondly, [1] included a description of the test execution procedures, i.e., the processes that happen after an experiment is instantiated and deployed. The heterogeneity of the potential experiments that could be executed over the 5G EVE infrastructure has not changed from the time Deliverable 5.1 was delivered, so it is still considered useless to define the specifics of every possible test. Instead, the obvious differences between “technology benchmarking” and “vertical’s application testing”, and their impact in the test execution processes, were used to draft two types of testing procedures.

In technology benchmarking, the most common objective is to characterize some infrastructure, in many occasions independently of the application. On the other hand, normally the objective of vertical’s application testing is to understand the relationship of a specific KPI of interest for the vertical with some external conditions, like the delay, the available bandwidth, etc. This leads, as said, to different testing approaches that will be extended from [1] in section 2.2 of the present document.

Additionally, this difference also affects other functionalities of interest for WP5, like the validation and the monitoring procedures. In technology benchmarking, for example, if the specific application is not important, it may become very difficult to define a validation condition that can be used to determine if a test has “passed” or “failed”. This is totally opposite when a Vertical is testing an application, since the Vertical can perfectly determine the conditions that will validate the test. In that sense, the validation and monitoring procedures had only been very briefly tackled in the previous Deliverable, so the material in sections 2.3 and 2.4 of this document are practically new compared to D5.1.

Thirdly, [1] provided a very high-level architectural description of the delivered tools as a whole.

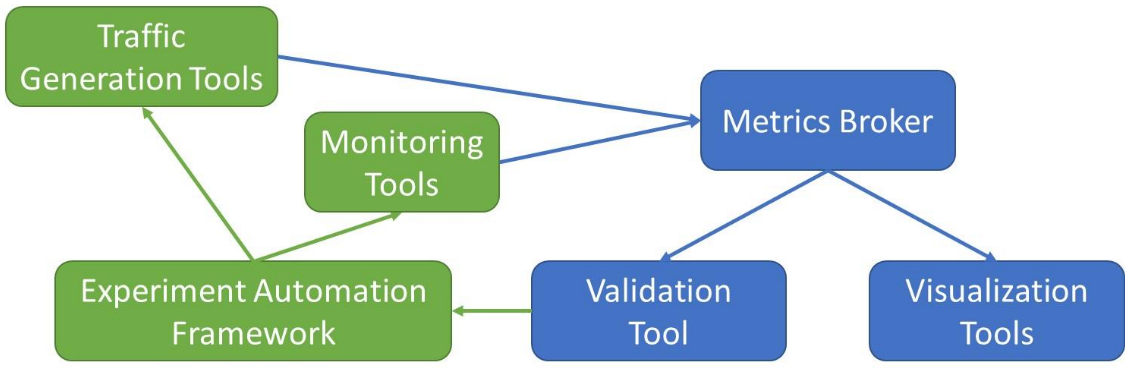


Figure 2: High-level architecture of D5.1 testing tools[1]

As seen, the so called “Experiment Automation Framework” was capable of orchestrating the execution of experiments, by interacting with the “Traffic Generation Tools” and the “Monitoring Tools”. These two modules were capable of pushing results or measurements into the “Metrics Broker”, which in turn made these available for the “Validation Tool” (pass/fail) and the “Visualization Tools” (graphical representation).

Some of these modules are not really components of WP5, like the Metrics Broker or the Visualization Tools. In 5G EVE, the Broker is actually implemented by the Data Collection Manager of the Interworking Layer (WP3), while the graphical representation will be made available via the Experimentation Portal, based on WP4 modules. Furthermore, between the WP5 framework and the nodes or tools at the local sites, there is yet another module from the Interworking Layer: the Runtime Configurator, hiding the specifics of such local nodes and tools and providing a unique northbound interface.

However, this helps to understand what was meant by “providing preliminary tools replicating to some extent the desired behaviour for the final Testing and Validation Framework”, sentence that was included in the first paragraph of this section. With the D5.1 tools, experimenters with an already deployed experiment were capable of executing tests in an automated fashion, and receive results both in terms of graphs (relating the target KPI with the desired external condition) and in terms of a final conclusion (pass/fail).

In section 2.1 of the present document, the final framework architecture will be presented, extending that of D5.1 and now fully covering the interaction with the rest of the WPs. This alignment will also be shown in the workflows of section 3 that involve external modules, like the experiment design or execution workflows.

1.1.1 Terminology

The following definitions were already included in Deliverable D5.1. We have brought them back to this Deliverable because they are very commonly used throughout the text.

- **KPIs**: key performance indicators represent measurable scenario parameters which provide meaningful performance information to the experimenter. KPIs can be of different types (network related or vertical related, for example), and the type of information that they provide can be used for many different purposes. Validation and monitoring are just two of these purposes.

[1] The different colors of the modules in this figure relates only to their implementation in D5.1. Modules in green were provided as part of one VM, while modules in blue were provided as part of another VM.

- **Validation parameters**: KPIs used to determine the result (PASS, FAIL) of a determined experiment will be called validation parameters. Together with a threshold, or range, validation parameters create the validation conditions which validation tools will use for their analysis.
- **Testing variables**: these are the parameters that the experimenter will want to modify during the test execution, so that (normally) a relationship between a testing variable and a validation parameter is found.

Another important definition is to differentiate between application metric and infrastructure metric.

- **Infrastructure metrics**: these are measurements, logs, events, etc. that come from infrastructure components like PNFs, 5G EVE internal VNFs, network devices, passive probes, servers, etc. Typically, this will be used exclusively for monitoring, ensuring that the experiment runs under appropriate conditions.
- **Application metrics**: these are measurements, logs, events, etc. that come either from the application that is being tested (e.g. Vertical VNFs), or from traffic tools like generators, active probes, etc. These can be monitored as well, but their main usage is as part of the results analysis and validation.

1.2 Testing-as-a-Service (TaaS) 5G EVE approach

The primary goal of 5G EVE with regards to experimentation is to offer a test environment for stakeholders interested in 5G scenarios, technologies, components or applications. Those stakeholders may be:

a) Internal to 5G EVE, for example, site facility operators or contributors interested in functional testing or benchmarking during the deployment of new 5G features, or internal Verticals providing Use Cases to the project.
b) External to 5G EVE, for example third parties using the end-to-end facility for experimentation or assessment of their own applications and services.

However, just offering a test environment is not the ultimate goal of 5G EVE. Instead, the ultimate goal of 5G EVE is to allow for the automated execution and evaluation of test cases, based on high-level test plan descriptions of the intended experiments, and according to model-based testing principals. This is the definition of the Testing-as-a-Service (TaaS) capability that 5G EVE is planning to offer, definition where “automated” is one of the keywords.

Another characteristic of the 5G EVE TaaS offer is its multi-site capability. Experiments will be able to run intra- or inter-facility, depending on the stakeholder’s requirements. In addition, just executing experiments will not be the only functionality. 5G EVE will also offer monitoring and test validation features, improving the TaaS offer with essentially very little effort from the stakeholder (adaptation to a very common data broker).

Key for this TaaS approach is also the Experimentation Portal, single point of interaction with interested experimenters, and from which they will be able to request the deployment of experiments, visualize different metrics during the execution and get the final test reports.

1.3 Document objectives

The main objective of this document (and that of Task 5.1), as per the 5G EVE Description of Action, is the *design of an overall framework enabling and supporting model-based testing*. Such framework *will allow for the thorough testing and evaluation of the involved 5G technologies and components*.

During the first months of 5G EVE, we already realized that this objective was not complete if stated that way, for two reasons.

A first reason is hinted in the Introduction to this section itself: it is said that the document specifies *a testing and validation architecture capable of providing TaaS*. The key differences are about “validation”, which was of course inherently considered in the original scope of 5G EVE, but not explicitly mentioned, and about “automation”, which is a concept directly coupled with the concept of TaaS. It has to be noted that these two processes (testing and validation) are also accompanied by the monitoring (both of the infrastructure and the applications results). However, from a high-level point of view, the monitoring is in fact a procedure supporting the validation: it may provide not only a visual validation of the test results, but also the validation of the infrastructure where tests are run (i.e., that there are no failures, or impairments affecting the results).

Secondly, the evaluation of 5G technologies and components (benchmarking) is only a partial objective in 5G EVE. It is also an objective to host 5G-based applications and services from third parties (typically Verticals). The main implication of this statement for WP5 is that we need to implement the capability to work with KPIs defined by Verticals (apart from the Network KPIs) and measured using their own tools, which broadens the scope of the modules conforming the testing and validation framework.

Having said this, a better-expressed main objective of this document is the "design of an overall framework enabling and supporting automated model-based testing and validation of 5G technologies, components and services".

It is important to mention that this document is a design document, not governed by potential limitations of the technologies used for implementation. In other words, design specifications will be made as general and inclusive as possible, even if relying on features not yet supported by the equipment deployed in sites. Later on, during the implementation phase, strict focus will be placed on those features that are needed by the Verticals experimenting at 5G EVE.

1.4 Document structure

Sections of Deliverable D5.2 following this Introduction are organized as follows:

- Section 2 includes the specification of the testing and validation framework: high-level methodology, aligned with WP1, and architecture, extended from what was presented in D5.1. It also presents the specific methods to validate Network KPIs, and the high-level definition for the three main procedures in 5G EVE: testing, validation and monitoring, the latter based on the KPI collection framework (again, aligned with WP1).
- In Section 3, we define all the workflows and APIs associated with the methodology. The description has been structured following the four stages depicted in Figure 1, and we include in each phase all the operations in which WP5 is involved. We also describe in this section the internal workflows and APIs among the two main WP5 modules (three if we count the scripts repository): the Experiment Execution Manager and the Results Analysis & Validation module.

  As part of the Experiment Design phase, this section includes an update of the Test Plan templates: how they have evolved from the delivery of D5.1 and what new answers we have received from Verticals.
- Some initial implementation details are given in Section 4 for these two components, which will be the basis for the full development to be executed in Task 5.3.
- Section 5 updates on tests results from different use cases; initial results were provided in D5.1.
- Finally, section 6 includes the conclusions and next steps with regards to WP5 work.

Three annexes are also included in this document: Annex I for the most recent responses to the high-level test plan templates, Annex II for the first low-level test plan template from the ASTI use case, and Annex II including the complete experiment execution workflow described in section 3.4.1.

2 Model-based testing and validation framework

This section includes the formal definition of the 5G EVE testing and validation framework. Apart from extending the definition of the methodology that was outlined in previous WP5 Deliverables, it includes high-level descriptions of the testing, validation and monitoring procedures. So far, only the first of these had been briefly defined in this Work package.

The section also includes the KPI collection framework, worked in conjunction with WP1.

It is important to notice the lack of a State-of-the-Art analysis in this design document. To the best of our knowledge, an automated model-based testing and validation framework for 5G technologies, components and applications does not exist today. For example, in 5GinFIRE, also offering NFV-based testing infrastructure for 5G use cases, none of the experiment execution or results analysis were automated. Instead, experimenters had remote access to their deployed experiments, and executed their test cases manually as if they were in their own premises. Therefore, all the WP5 definitions included in this document have been developed from scratch in 5G EVE.

2.1 Test and validation methodology

In Deliverable D5.1 [1], an initial version of the testing methodology was already outlined. This definition was elaborated in conjunction with WP1 and has evolved very little since its initial publication[2].

In particular, regarding the four phases of which it is composed, only the final stage has had a small change of name, from "Test Evaluation and Results Analysis" (as depicted in Figure 1) to the new "Test Performance Evaluation and Analysis" (Figure 3). This change was proposed to emphasize the fact that the evaluation and analysis of results in 5G EVE is always done against some threshold value, which separates the correct or incorrect performance areas of the devices under test. Apart from that, the scope of this last phase is equivalent to the one that was previously defined.

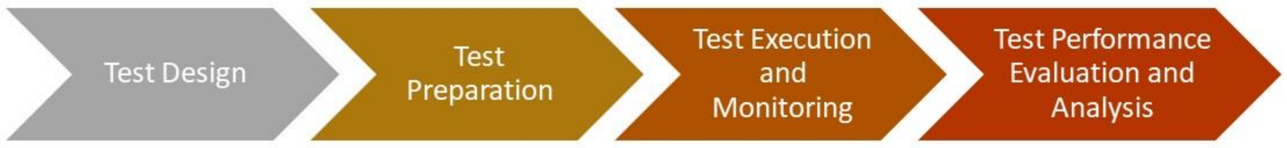

Test Design
Test Preparation
Test Execution and Monitoring
Test Performance Evaluation and Analysis

Figure 3: 5G EVE model-based Test & Validation methodology

More recently, in D1.3 – *5G EVE end-to-end facility reference architecture for Vertical industries and core applications* – [2], to be published in December 2019, these four phases have been further detailed. That description will not be included here to avoid duplications but, to ensure that this Deliverable is fully self-contained, here follows a brief summary before presenting the framework architecture.

- **Test Design**: its main goal is to permit 5G EVE Experiment Developers understand 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 good we are able to "understand" each other from the very beginning.

  The main external input to this phase, as already mentioned in D5.1, is the Vertical test plan template (also known as "high-level" test plan template), which Verticals use to transmit "in their own language"

[2] It has to be noted that, in D5.1, the methodology was named as "Testing Methodology". Although the title of D5.2 has not been changed ("*Model-based testing framework*"), it is more complete to refer to it as "Test and Validation Methodology" or "Framework", as explained in section 1.3.

what are their intentions. Based on that input, low-level test plan templates[3] are generated, which Experiment Developers use to translate the Vertical requirements into 5G EVE terminology and further objects, like blueprints (Vertical Service/Context/Test Case/Experiment blueprints), descriptors, test scripts, required VNFs/PNFs, etc.

Finally, it is also expected that VNF providers upload their packages into the Platform during this phase. Mechanisms will be implemented to make these packages available automatically across the whole 5G EVE facility once they are correctly approved and validated.

- **Test Preparation**: the objective of this phase is to get everything ready to run the desired experiment. All experiments have some initial conditions that must be set up by experimenters, and most of them have some variables and threshold values for validation that Verticals may want to particularize. This customization enables the generation of the required descriptors from the blueprints.

  From there, the interaction of experimenters and Site Managers is critical since, on the one hand, they must agree on a schedule for the experiment, and on the other, Site Managers must make sure that the testing environment is ready to host the experiment (e.g. for items outside of 5G EVE control, like the provisioning of SIMs).

- **Test Execution and Monitoring**: once the entire infrastructure is ready, the virtual environment to run the experiment is built (i.e., the experiment is instantiated under the control of the Multi-site NSO) and configured. This is 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 validations purposes.

- **Test Performance Evaluation and Analysis**: finally, and based on the gathered results, it is checked that the values of the KPIs are consistently inside the correct performance threshold; if so, the final test report will indicate that the test has been passed; otherwise, it will have failed.

WP5 modules do not participate in all these phases. In particular, they do not participate in the Test Preparation. For the Test Design, the process is covered in detail in section 3.1, while the execution, monitoring and validation have been jointly addressed in 3.4.

2.1.1 Test and validation framework architecture

Two are the key modules that WP5 has defined as part of the 5G EVE Test and Validation framework. These are the “Experiment Execution Manager” and the “Results Analysis and Validation” components. In fact, these are just more generalized names for the “Experiment Automation Framework” and “Validation Tool” that already appeared in D5.1 to describe the joint behaviour of the delivered disjoint testing tools.

Together with these two components, also a “Tests Scripts Inventory” has been included to store the different scripts associated with different Test Cases. This approach has been selected, in opposition to include the full scripts in the blueprints, to avoid the exchange of potentially huge blueprints in the most complex tests. This way, the blueprints will just include a script ID, simplifying the communication, and permitting the evolution of scripts without modification of the blueprints.

The WP5 modules and their relationship with the rest of the 5G EVE components involved in the test and validation processes are depicted in Figure 4 below. WP3 modules appear in yellow colour, while those of WP4 appear in green and those of WP5 in red. The colours of the different arrows show the allocation of the module that starts the communication.

[3] In D5.1 we included the low-level templates as part of the Preparation phase. We believe now that they fit better in the Design phase, mainly to split between a phase that generates “documents”, and the following one, which delivers “software code”. The later needs to be stored, to be automatically accessible to components of 5G EVE, etc., and its modification is much more critical.

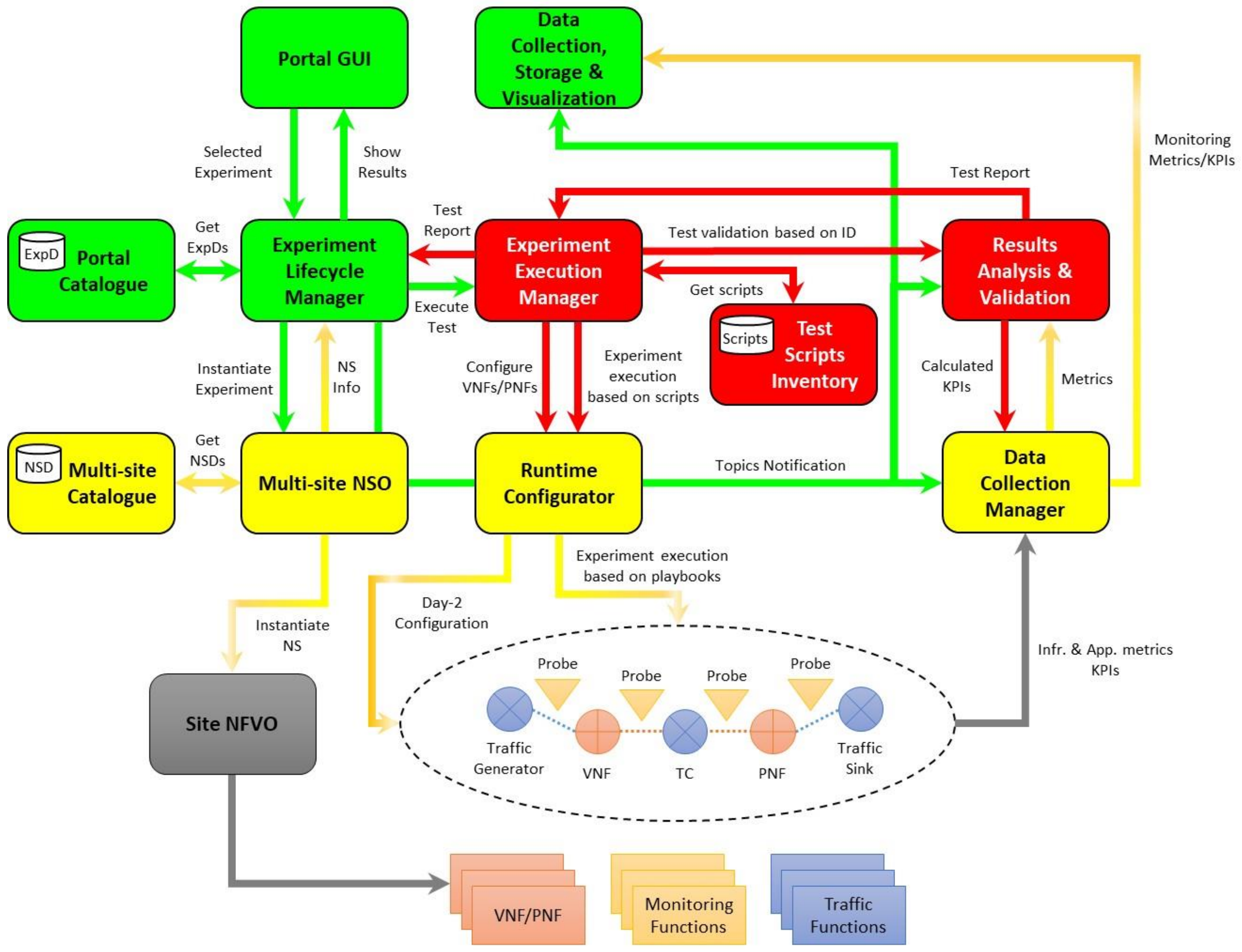


Figure 4: 5G EVE Test & Validation framework architecture

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). This is shown in Figure 4 by the "Execute Test" arrow from the ELM to the EEM. The request from the ELM includes meaningful information like:

- Network Service Instance (NSI) related fields, like the IP addresses of the deployed VNFs
- The Test Case Blueprints, which define the required scripts, the required KPIs, and all the Data Collection Manager topics that are needed for the specific experiment.

The EEM has four main functionalities:

- ➢ **To trigger the day-2 configuration**. After the ELM sends a request to the EEM, meaning that the whole experiment is deployed on top of the infrastructure, the latter needs to make sure that all VNFs, PNFs and the rest of the tools are correctly configured. This is done via the Runtime Configurator, to hide the site implementation details to WP5. The "Configure VNFs/PNFs" arrow shows this operation.
- ➢ **To trigger the execution of test cases**. Once day-2 configuration is correctly in place, the EEM will run each test case (in case there is more than one) in order, one at a time. For this, the EEM will get the required scripts from the Test Scripts Inventory and will again lean on the Runtime Configurator for the interaction with the site components: "Experiment Execution based on scripts".
- ➢ **To trigger the validation of test cases**. Once each test case is finished, the EEM triggers on the Results Analysis and Validation module the evaluation of the results ("Test validation based on ID"), and receives back a "Test Report", which distributes upwards to the ELM.
- ➢ **To return to initial conditions**. After every Test Case run on the infrastructure, the EEM will return everything back to original state to make sure there is no impact in the next executed Test Case.

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, and not in terms of successfully running the test scripts.

The main features of the RAV module are:

- **To calculate KPIs from raw metrics**. In some cases, it may happen that we can calculate meaningful KPIs from either the application or infrastructure metrics, or both. The RAV module receives from the ELM the Kafka topics to which it has to subscribe to receive these metrics ("Topics Notification"). Once these metrics are received, and KPIs are calculated, these are published back in the Data Collection Manager so that they can be visualized in the Portal.
- **To validate the test results**. Once the RAV module has received or calculated all the required KPIs, it will evaluate them using a threshold (or range) determined by a "correct performance" criteria. If all KPIs are inside the expected values, the experiment will be determined as PASSED; otherwise, it will have FAILED.
- **To generate the test report**. This passed or failed result, together with the obtained measurements, metrics, KPIs, etc., will generate the test report associated with the experiment, which, as already mentioned, will be sent upwards to the ELM.

The only functionality demanded from the **Test Scripts Inventory** is precisely to store the required scripts for all the experiments, making them available on demand to the EEM.

The role of the rest of the components included in Figure 10 will be fully explained in section 3, as part of the different internal and external workflows.

2.1.2 Test and validation of Network KPIs

In this section, we are providing a detailed description of the methodology (procedure and steps) followed for the verification and validation process of the 5G-PPP network KPIs that 5G EVE project is supporting. Table 1 summarizes these KPIs along with the minimum requirements. We also identify software tools (if available; this will be updated in future WP5 Deliverables) that can be used to provide the initial KPI measurements for an end-to-end system or service in an automated way.

First Deliverable of WP5, D5.1 [1], provided a general methodology, tools and measurements for bandwidth and latency metrics. The aim of this Deliverable is to analyse further the inter-categories of bandwidth and latency metrics along with the other KPIs. The first approach was conducted within WP1 and the related Deliverables (Table 1); however, our intention here is to focus on the specific KPIs that we have decided that are having more impact in our verification procedure.

Table 1: 5G EVE list of Network KPIs

5G EVE KPIs	ITU-R M.24010-0 (11/2017)	Minimum requirements
Speed	DL User Experienced Data Rate (Mbps)	100 Mbps
	UL User Experienced Data Rate (Mbps)	50 Mbps
Broadband Connectivity	DL Peak Data Rate (Gbps)	20 Gbps
	UL Peak Data Rate (Gbps)	10 Gbps
Capacity	Area Traffic Capacity (Mbit/s/m2)	10 Mbit/s/m2
Latency	UP Latency (ms)	1ms (URLLC), 4ms (eMBB)
	CP Latency (ms)	<20ms
Device Density	Connection Density (devices/km2)	1 M devices/ km2 (mMTC)
Mobility	Stationary (km/h)	0 km/h
	Pedestrian (km/h)	0 km/h to 10 km/h

The KPIs in which we are focusing at this stage are:

1. **Speed – Experienced user throughput (Mbps).** Specified as bi-directional data throughput, Downlink (DL) and Uplink (UL), which an end-user achieves on the user plane, preferably on MAC layer avoiding the overhead of the upper layers. This KPI is one of the dominant measurements for the Quality of Experience (QoE) level that a user experiences for an active application/service.

   As also described in D1.4 – *KPI collection Framework* – [3], to measure this KPI the application/service throughput across time must be measured and recorded.

   For DL (user-) throughput, the measurement should be performed at UE level. As a result, UE should have the capacity (logs, tools) to expose this metric. There are several open-source software tools that can measure the current DL traffic when a service is active. These tools can be installed on the UE side providing the required measurements. Another approach is to capture the ingress packets on UE (the ones that finally arrive at the device) using Wireshark tool. Upon monitoring the incoming traffic, you can easily extract the current DL throughput.

   For the UL (user-) traffic, we are focusing on the packets/throughput that finally arrive at the application server hosting the provided service. There are several software tools that can be used to capture the ingress traffic on the interface/port/IP accommodating the functioning service/application. Wireshark is one of them, as well.

   In case simulation procedure is going to be followed, iPerf seems the more relevant tool. The client-server model that illustrates to generate and handle the traffic suits better for monitoring (user-) DL (downlink client side) and (user-) UL (downlink server side).

2. **Broadband Connectivity – Peak Data Rate (Gbps).** Specified as highest data rate provision that a single user can reach during high traffic demand periods resembling in turn the peak data rate when required. The methodology of measuring this KPI follows the same steps as in the Speed case. As a result, same software tools can also be used for the validation of this KPI in a realistic or simulated way.

3. **Capacity – Area Traffic Capacity (Mbit/s/m^2).** Specified as total traffic throughput served per geographical area. The formula used to derive the result is the following:

   Area traffic capacity (bps/m2) = site density (site/ m2) x bandwidth (Hz) x spectral efficiency (bps/Hz/site)

   This KPI is also defined as **Traffic Volume Density**, describing the total user data volume transferred to/from end-user devices during a predefined time span divided by the area size covered by the radio nodes belonging to the RAN(s). For multi-hop solutions each user data is only counted once. In the network, can be generally computed by the sum of traffic volumes each produced by an end-user device, possibly differentiated between downlink and uplink direction, divided by the overall service area covered by the corresponding radio nodes.

   To compute this metric in a network level, RAN node(s) must provide the active/connected user within a reporting period along with the data rate of each user in order to conclude on the total user data volume (probably in average) within a pre-defined time span (time granularity). The traffic volume density is then computed dividing this value (bits/sec) by the total area covered by the node(s) (m^2).

   The specific simulation method is still to be defined and implemented; however, related information can be extracted from radio specification documents.

4. **Latency – User Plane & Control Plane (ms).** For user plane latency, we can consider two types of metrics. End-to-end latency or one-trip time (OTT) latency is the first type. The other type is the round-trip time (RTT) latency. Ideally, latency should be measured in the MAC layer, not accumulating any processing time on higher layers given the used application. On the other hand, control plane latency is defined by ITU as the transition time from a most "battery efficient" state (e.g. Idle state) to the start of continuous data transfer (e.g. Active State). This requirement is defined for the purpose of evaluation in the eMBB and URLLC usage scenarios. The minimum requirement for control plane latency is 20ms.

Proponents are encouraged to consider lower control latency, e.g. 10 ms [4]. An illustration of control plane signalling during transition from Inactive to Active states is shown in Figure 5.

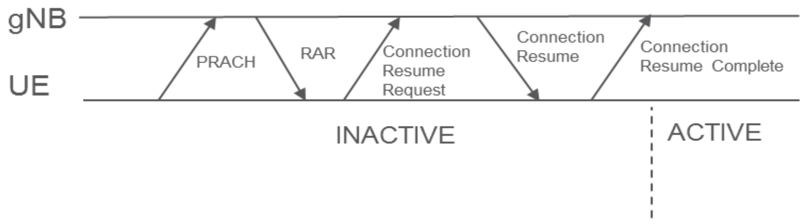


Figure 5: Control plane signalling during transition from Inactive to Active states

In real networks would be more useful to measure the OTT time for user plane, i.e. the packet(s) (one-way) time from source (user) to destination (application server). This time can be measured monitoring a specific packet given its unique identification (Identification Field in IP header) using the Wireshark tool. For sure, this metric will not be based on MAC layer but as a comparison between 4G and 5G networks it is enough. For control plane latency, we have to be sure when user is under IDLE state. Then time from requesting connectivity/service until the first uplink packet is our metric. Probably this metric can be extracted from 3GPP specification for 5G NR.

In simulation mode, user plane latency could be measured monitoring the RTT time. Pinging a server (ICMP messages) from UE is a way to measure this value exploiting Network Layer (Layer 3). For a more accurate measurement, latency on MAC layer can be measured. NetScanTools gives the response time (latency) of a device to an ARP packet i.e. on MAC layer.

The specific simulation method for the Control Plane is still to be defined and implemented; however, related information can be derived from radio specification documents.

5. **Device Density – Connection Density (devices/km^2).** Specified as the total number of simultaneous active connections per square kilometres fulfilling a target QoS. Active means the devices are exchanging data within the network. Number of connected/accessible devices are available in NG-RAN node.

   In real networks this metric can be derived gathering the number of active devices connected to a radio node in a considered area. This area can also be specified given the covering capacity of the radio nodes or the configured ones.

   The specific simulation method is still to be defined and implemented; however, related information can be derived from radio specification documents.

6. **Mobility – Stationary / Pedestrian (km/h).** Specified as the system's ability to provide seamless service experience to users that are moving. Two modes are considered in the frames of 5G-EVE project. Stationary having a speed of 0 km/h and Pedestrian having a speed of 0 to 10 km/h. Seamless service experience means that UE achieves QoS required by the service/application. Parameters to be considered having impact on the performance evaluation of any mobility activity is the signalling behaviour during a radio handover and the quality of the connectivity given the distance from the interacting cells. Therefore, in this context, this KPI can then be translated as the system's ability to reach measurable Vertical / UC requirements of 5G-EVE KPIs i.e. Latency, Speed, Capacity and Device Density to users that are moving.

   In real networks, first we have to measure the speed of the user/device. This can be done with tools/applications on the device or with probes (measuring the time) on the specified route the user/device is following. During this move, with the already defined methodologies we are able to validate the translated 5G-EVE KPIs experienced by the user.

   The specific simulation method is still to be defined and implemented; however, simulation techniques of the translated KPIs can be used.

2.2 High-level testing procedure

As already stated in section 1.3, the objective of this document is the design of an overall framework enabling and supporting automated model-based testing and validation of 5G technologies, components and services. This objective brings back the discussion of D5.1 about different test procedures being required for technology benchmarking or testing of Vertical applications.

As a reminder, in D5.1 we were defining the purpose of "technology benchmarking" as the characterization of some 5G architecture, component, technology, etc., normally based on Network KPIs. On the contrary, the goal of "Vertical applications testing" is to understand the behaviour of a specific application/service on top of a 5G network, behaviour typically identified by Vertical KPIs.

In D5.1 we focused on bandwidth and delay evaluation for technology benchmarking, using tools like iperf and ping, so a specific testing procedure was defined only for the testing of Vertical applications. In this Deliverable, the main design criteria to add technology benchmarking into the procedure was to try to modify it as least as possible, to avoid changes in the already developed code. The result is presented in Figure 6.

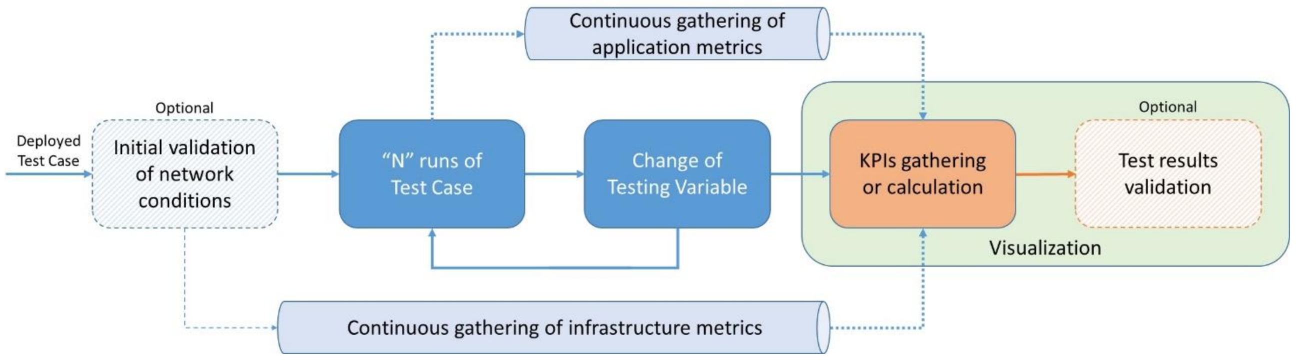

Continuous gathering of application metrics
Optional
Deployed Test Case
Initial validation of network conditions
"N" runs of Test Case
Change of Testing Variable
KPIs gathering or calculation
Optional
Test results validation
Visualization
Continuous gathering of infrastructure metrics

Figure 6: 5G EVE testing procedure

The testing procedure begins when the test case is fully deployed on top of the 5G EVE infrastructure. The first step is the potential validation of the deployment scenario, ensuring that the network is working under the appropriate conditions. For this, in D5.1 a quick initial test similar to what technology benchmarking could look like was proposed. Therefore, it has been changed to optional in this Deliverable: if the test is a benchmarking test, this preliminary review is not needed, and if the test is a Vertical application, in fact it is desirable but still left as optional. It must be reminded that in the Test Preparation phase (Figure 3), one of the key responsibilities of Site Managers is precisely to "make sure that the testing environment is ready to host the experiment".

After this preliminary quality assurance review, the execution of the Test Case itself starts. The procedure includes "N" runs of the Test Case under the same conditions, as a single test result may not be sufficient to understand the experiment. A very simple example showing this is executing a single latency measurement, i.e., a single ping. This would generate a result inside the range determined by average delay plus jitter, but we should not determine that the delay has a specific value only from that single measurement.

After we have run the Test Case "N" times under the same conditions, we can then move forward and change one of the testing variables ("X"). The loop then continues by running again the Test Case "N" times, changing a new testing variable ("Y"), and so on until all the scenarios are executed.

It has to be noted that this procedure is very tightly coupled with the other two included in this section: validation and monitoring. Regarding the latter, throughout the whole duration of the experiment infrastructure metrics are being generated and collected, from the different nodes composing the test scenario. In addition, application metrics (mainly from the traffic generation tools or from the deployed VNFs/PNFs) are being continuously gathered once the running of the test case starts. Both types of metrics feed on the one hand the Results Analysis and Validation module, but on the other hand, the collection, storage and visualization modules of WP4, for monitoring purposes.

Finally, in the figure, the Test Results Validation stage is put as optional. In this case, this accounts for those benchmarking scenarios in which validation conditions are not defined, or validation parameters are not measurable. It also accounts for those test cases in which Verticals are not able to provide their own KPI measurements, or there is no way to establish a mathematical relationship between Network KPIs and Vertical KPIs. In all these cases, the 5G EVE platform will only present (visualization) the obtained metrics versus the testing variable, without determining the final result of the experiment (pass/fail).

New Test Cases will be able to run on top of the same infrastructure by just bringing it to initial conditions in between tests.

2.3 High-level validation procedure

In 5G EVE, validation of a test case means to compare the obtained test results, in the form of KPIs, with a certain range or threshold conforming a “correct performance” area.

The validation conditions can be very simple or very complex, depending on the test case. In 5G EVE, and since the KPIs will normally be in numeric format, it has been decided to implement validation conditions based on mathematical expressions like “is greater than”, “is lower than”, “is between a minimum and a maximum”, “is equal to”, etc. This means that more complex validation conditions, for example of the form “X out of Y measurements are <condition>”, will not be supported in the first phase.

Also, each KPI will be individually validated, excluding for example conditional validation conditions of the type “if KPI “A” is <single_condition> then KPI “B” must be <single_condition>; otherwise…”.

Having said this, numeric KPIs can be formed of a single value, or a list of values. Examples of the former are a QoE value for a specific video being transmitted and played at a remote device, under certain network conditions, or even more simply, the percentage of packet loss throughout the entire experiment. A list-based KPI could be the deviation error of an AGV from the guide, measured for example every millisecond, or the roundtrip delay measured by ping every second.

In case the KPI is a single value, the validation condition will be applied of course over that value: QoE must be greater than 4 to consider the test passed, or packet loss must be lower than 0.001%. For list-based KPIs, all the values in the list will have to comply with the comparison condition. This might seem a very strict policy, but in Vertical’s environments it is fully justified in most cases. Imagine an AGV that deviates more than the threshold (for example, 5cm) only once, and goes 6cm in one direction; this might be sufficient in the production environment to have the AGV hitting something and creating a huge impact.

Finally, validation could be executed both in real time or after the whole test is finished. Typically, for KPIs that have a single value, we will have to wait until the test is finished, and this value is either generated or calculated. However, in list-based KPIs, these will be typically produced along the whole experiment. We could think about executing the validation on each value or storing the values and executing the validation at the end over the whole list. The first mechanism would be more agile and would permit actions like stopping a long test at the very beginning, if results are not OK from the start. However, the logic would be more complex, and depending on how fast KPI values were generated, the processing power to do the comparison could be very high. Therefore, the initial agreement in WP5 is to trigger the validation process once the whole Test Case has been finished.

Figure 7 below covers these ideas: after the test case is finished, and the validation modules gather all the values of the KPIs, a loop comparing each value with the validation condition is executed. For single-value KPIs this will be executed just one; for list-based KPIs, the loop will be executed a number of times, equal to the number of values in the list. After that, the result of the experiment will be declared as PASSED if all the values meet the validation condition or FAILED even if only one value is outside the correct performance area.

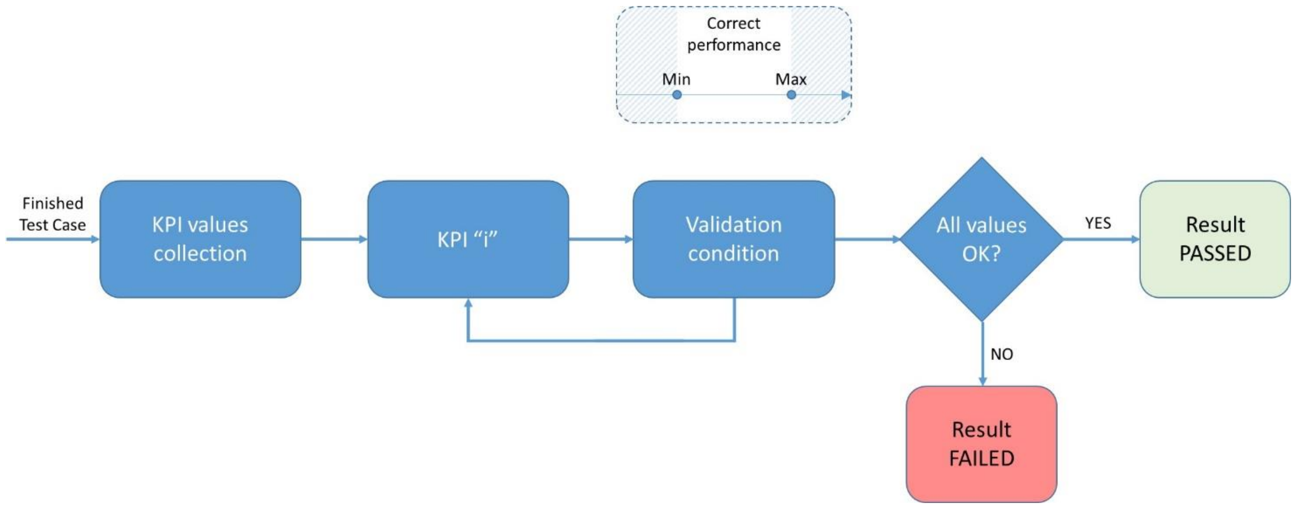


Figure 7: 5G EVE validation procedure

2.4 High-level monitoring procedure

Monitoring in 5G EVE is not an exclusive task of WP5. Even in Figure 6, where infrastructure and application metrics are shown to be continuously gathered, to be able to generate KPIs:

- The generation of these metrics is not triggered by WP5 modules, but by the Runtime Configurator (WP3)
- The distribution of these metrics is not done by WP5 modules, but by the Data Collection Manager (WP3)
- And the visualization of these metrics is not done by WP5 modules, but by the Kibana component of the Portal (WP4)

In other words, in WP5 we are consumers of two out of the three types of parameters that need monitoring: the mentioned infrastructure and application metrics. The procedure to consume these metrics by the Results Analysis and Validation (RAV) component is as follows:

1. As a pre-condition, the RAV must be subscribed to the signalling topics of the Data Collection Manager (DCM). From these signalling topics it will receive (live) information about which test-related topics are interesting for its activity, and therefore to which test-related topics it should subscribe
2. Each time a new experiment is being deployed, the Experiment Lifecycle Manager (ELM) generates the topics associated with that experiment, publishes them in the signalling topic of the DCM, and the RAV subscribes to that topic.
3. The RAV receives all the measurements published in the topics it is subscribed.

Nevertheless, in WP5 we also generate the KPIs, which is the third type of parameters considered for monitoring. In this case, the procedure is as follows:

1. Again, the RAV must be subscribed to the signalling topics of the DCM as a pre-condition. In this case, it will receive from the ELM the name it must place to the KPI-related topics of each experiment.
2. Once KPIs are calculated, or obtained by any means, the RAV publishes them in the DCM under the appropriate topic.
3. The collection module at the Portal receives all the KPIs published by the RAV, as a prior step to their storage and visualization.

As seen, the conclusion is that the monitoring procedure in WP5 is leveraging completely the mechanisms provided by other WPs, avoiding the implementation of different modules at different Work Packages doing the same (or very similar) functionalities.

2.5 KPI Collection Framework

The KPI Collection Framework has been defined at a high level in the deliverable D1.4. The modules that comprise it span different Work Packages, specifically WP3 and WP5. The two main building blocks of the framework are:

- Data Collection Manager
- Data Shippers

2.5.1 Data Collection Manager

The Data Collection Manager is implemented in the context of WP3. The DCM software architecture is based on Apache Kafka, which is a well-known publish-subscribe tool that acts as a bus, being able to manage and send data to different applications connected to it. The main functionalities of Kafka can be seen in Figure 8.

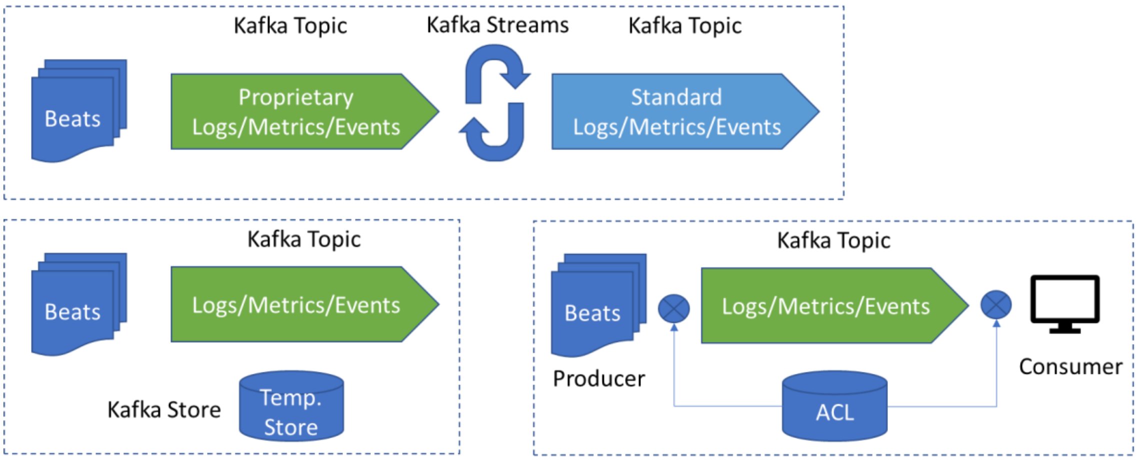


Figure 8: Main services and functionalities offered by Kafka

Currently, the implementation of the DCM is a dockerized environment for testing the 5G EVE Monitoring & Data Collection tools from WP4, which includes a specific Kafka container for performing the publish-subscribe queue functionality. That environment is already available in the 5G EVE repository. The expected solution is a clustered Apache Kafka environment, coordinated by a dedicated tool such as Apache ZooKeeper, complemented with a management platform like Kafka Confluent. This complete architecture is expected to be finalized for the deliverable D3.4. Examples of the various operations supported by this DCM has already been described in detail in D3.3.

2.5.2 Data Shippers

Based on the proposed initial design and taking into account the parts of this architecture implemented in WP3, the main focus of the KPI Collection Framework, pertaining to WP5, revolves around the information collection agents responsible for gathering metrics and logs and shipping them to the DCM. For the implementation of these agents to be robust and modular, the tool proposed for this task are the Beats by Elastic. They come in various flavours based on the kind of data we need to send like log files, metrics, events, etc... They are also simple to customize and can cover most of the needs required from the Collection Framework's implementation. Also worth noting is the fact that they do not require significant resources to operate. It is unclear yet, though, how the metrics or logs created from physical instrumentation will be sent to the DCM. If the interfacing of these tools with a terminal is viable then there is no problem. If no such interface is possible then another route must be taken. The typical, but not mandatory, flow of information when using Beats as data-shipper can be seen in Figure 9.

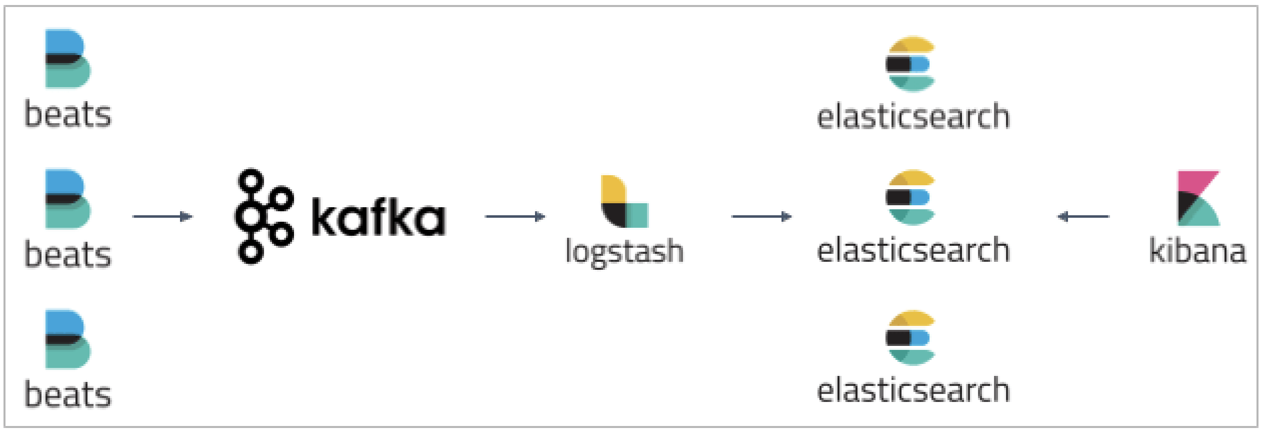


Figure 9: Beats information flow

This flow is in accordance with the DCM from WP3 and Monitoring and Visualization tools from WP4. Deploying Beats is very straightforward and simple both in physical and virtual nodes.

While they cover the part of shipping the information to the appropriate components, the other issue that needs to be addressed is the producers of that information. In order the simplify the log shipping process the various information producers should share one or more common log storing locations so as to minimize the diversity in the configuration of the data shippers. Another issue the matter of logging format. It is a given that each tool, in-house created software as well as Vertical software do not produce similar logs. As such, it would be of great benefit to the implementation effort an adherence to some basic principles in the generated logs. A typical example of such a format would be enriching the metrics, logs and other kind of messages with additional fields in order to be able to easily identify and categorize these messages. A typical example of such fields would be timestamps, the type of message, which node or software generated the message, what KPI it pertains to and so on.

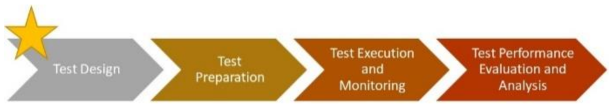
Since the metrics pertaining to the Network KPIs are valuable information for all use cases and experiments alike, there are two options to tackle the issue of collecting those metrics. First option is that software instrumentation be placed on all the nodes of the various VNFs so that network metrics are monitored by default and logs are generated automatically. Second option is that these producers, be passed or activated as part of Day-2 configuration of the VNFs. This task will be managed by the Runtime Configurator.

Regarding Vertical KPIs, in any case, it is necessary for the Verticals to produce the logs of their service which can then be shipped to the DCM accordingly. So, while no additional producers are needed for the Vertical KPIs some implementation is necessary either for parsing the data beforehand and then shipping them or parsing the data at a separate component like the Analysis and Validation modules. For the latter additional input might be need from the Verticals regarding the form of the produced messages and the specific analysis method they desire.

3 Testing and validation framework workflows and APIs

In this section, we detail the specific workflows, APIs and triggers of the 5G EVE testing and validation framework. The description follows the stages of the overall experiment workflow, evolving from the initial Experiment Design step in the 5G EVE portal down to Experiment Preparation, Experiment Execution and, finally, Result Analysis and validation procedures are described in a way to highlight the most important operations to be supported by the software modules implementing the 5G EVE portal. The APIs of the Experiment Execution Manager are provided as this module is a core component of the Model-based testing framework, responsible for coordinating all the actions related to experiment execution and validation for 5G experiments instantiated in the 5G EVE portal.

3.1 Experiment Design workflow and APIs


Test Design
Test Preparation
Test Execution and Monitoring
Test Performance Evaluation and Analysis

The design and formal definition of experiments in 5G EVE is facilitated by the introduction of high-level, intent-based information models called blueprints. A blueprint is a template that describes a piece of information about the execution of an experiment in 5G EVE facilities. Being a template, it enables some customization from Experimenters (final users of the 5G EVE platform). Blueprints are mapped or translated to more detailed lower-level entities, which include all the details for the correct deployment, execution, and data collection of the Experiment.

Many actors are involved in the Experiment Design workflow. A thorough description of the actors' roles, interaction workflows, and information models can be found in D4.1 [5].

In this section of this document, we are going to integrate some of the workflows defined in Section 3 of D4.1 and we are going to discuss more thoroughly the information models related to Test Case Blueprints, metrics, and KPIs.

3.1.1 Test Case Blueprint information model

The Experiment Blueprint (ExpB) includes a list of Test Case Blueprints (TcB). A TcB is a customizable template that describes the actions to be performed during the execution of the experiment. Actions are collected in a script and user parameters and infrastructure variables can be used to customize the script execution. Table 2 reports the fields composing an ExpB with their short description.

Table 2: Test Case Blueprint fields with relevant description

Field	Description
testcaseBlueprintId	Unique Identifier for the Test Case Blueprint
Script	Source code of the script to be executed
userParameters	Mapping of script variables to user parameters for the test customization
infrastructureParameters	Mapping of script variables to infrastructure parameters for the dynamic test configuration based on the specific NS related to the experiment (e.g., IP addresses)

"userParameters" and "infrastructureParameters" are used to make the TcB customizable, dynamic, and reusable. "userParameters" are used to inject the value of some parameters into the test script to be executed. As an example, we can consider a Test Case Blueprint associated to an experiment to evaluate the performance of a network service under the effect of artificial background traffic. A trivial user parameter is the amount of background traffic to apply during the experiment. The "userParameters" field will have a key-value pair to map the parameter name to the variable name used in the "script" source code to control the amount of background traffic.

The "infrastructureParameters" is used to make the TcB applicable to many network services. Considering again the example used before, the application of artificial background traffic trivially needs two endpoints to be applied, a source and a sink. The "infrastructureParameters" will then include the necessary key-value pairs to inject in the test script the IP addresses to be used as source and sink for the background traffic.

3.1.2 Metrics

Vertical Service Blueprint (VSB), Context Blueprint (CB), and ExpB information models include a list of metrics. This field is used to advertise what kind of monitoring data the service, context, or experiment can provide once they are deployed. The formal representation of a metric is reported in Table 3.

The three types of data collection for metrics are the following:

- CUMULATIVE: Increasing over time (e.g., CPU time)
- DELTA: Changing over time (e.g., bandwidth)
- GAUGE: Discrete items and fluctuating values (e.g., disk I/O, CPU utilization in %)

Metrics are partitioned in two sets: application metrics and infrastructure metrics. As the names suggest, the first set is composed of metrics related to the application or service domain, while the latest is composed of metrics available on the computing and network infrastructure. Metrics should be explicitly selected in the blueprints as the approach to collect all the available metrics is not sustainable in terms of data rates and data volumes.

Table 3: Metric fields with relevant description

Field	Description
metricId	Unique Identifier for the metric
Name	Name for the metric
metricCollectionType	Can be one of the following: CUMULATIVE, DELTA, GAUGE
Unit	The measurement unit of the metric (e.g., seconds)
interval	The sampling interval of this metric
Topic (application metric only)	Topic for publish/subscribe broker. Needed for collecting data
iMetricType (infrastructure metric only)	The infrastructure metric type.

The "topic" field is needed for the data collection of application metrics. As many services can be deployed on the 5G platform, it is needed to define a topic for the Data Collection Manager to properly store and retrieve metric data.

The topic is not needed for Infrastructure Metrics, as they are statically defined in the platform and already associated to dedicated topics. The available values for the "iMetricType" field are the following:

- LOST_PKT: lost packets count
- RECEIVED_PKT: received packets count
- SENT_PKT: sent packets count
- BANDWIDTH: network bandwidth
- LATENCY: network latency average

- JITTER: network latency jitter
- CPU_CONSUMPTION
- MEMORY_CONSUMPTION

3.1.3 KPI

The ExpB includes a list of Key Performance Indicators (KPIs). A KPI is used to evaluate the performance of the experiment on one specific aspect. It can be constructed upon one simple metric or upon more metrics combined in a mathematical formula. An objective (or goal, or threshold) can be defined to validate the KPI. The latter is included in the Experiment Descriptor. Table 4 reports the fields composing an ExpB with their short description.

Table 4: KPI fields with relevant description

Field	Description
kpiId	Unique Identifier for the KPI
name	Name for the KPI
formula	Mathematical formula built on one or more metrics
unit	The measurement unit of the KPI (e.g., seconds)
metricIds	List of metrics involved in this KPI
interval	The sampling interval of this KPI

While the “metricIds” is a simple list of the metrics involved in the KPI construction, the “formula” specifies how to mathematically combine the listed metrics to obtain the KPI value. As an example, we can consider the percentage of lost packets by one of the components of a network service. This can be obtained by combining the RECEIVED_PKT and LOST_PKT infrastructure metrics showed in the previous section.

3.1.4 Specific workflows

3.1.4.1 Definition and on-boarding of a Test Case Blueprint

The Test Case Blueprint includes information about tests to be executed in the experiment. The Experiment Blueprint contains references to one or more Test Case Blueprints.

The definition and on-boarding of Test Case Blueprints requires the interaction between the Experiment Developer, the portal catalogue, and the test script inventory. Figure 10 below shows the workflow for the onboarding of a Test Case Blueprint and the relevant Test Scripts.

The Experiment Developer first defines the Test Script offline. The Experiment Developer sends an on-boarding request to the Test Script inventory (Step 1). The latter generates an ID for the Test Script (Step 2) and stores the content of the script (Step 3). Finally, it returns a notification to the Experiment Developer with the Test Script ID attached (Step 4). The Experiment Developer can now define the TcB by including the ID of the Test Script just onboarded. When ready, the Experiment Developer sends an on-boarding request to the Portal Catalogue for the TcB (Step 5). The latter generates an ID for the TcB (Step 6) and it stores the TcB with references to the just onboarded Test Scripts (Step 7). Finally, it returns a notification of the successful onboarding to the Experiment Developer with the TcB ID attached (Step 8).

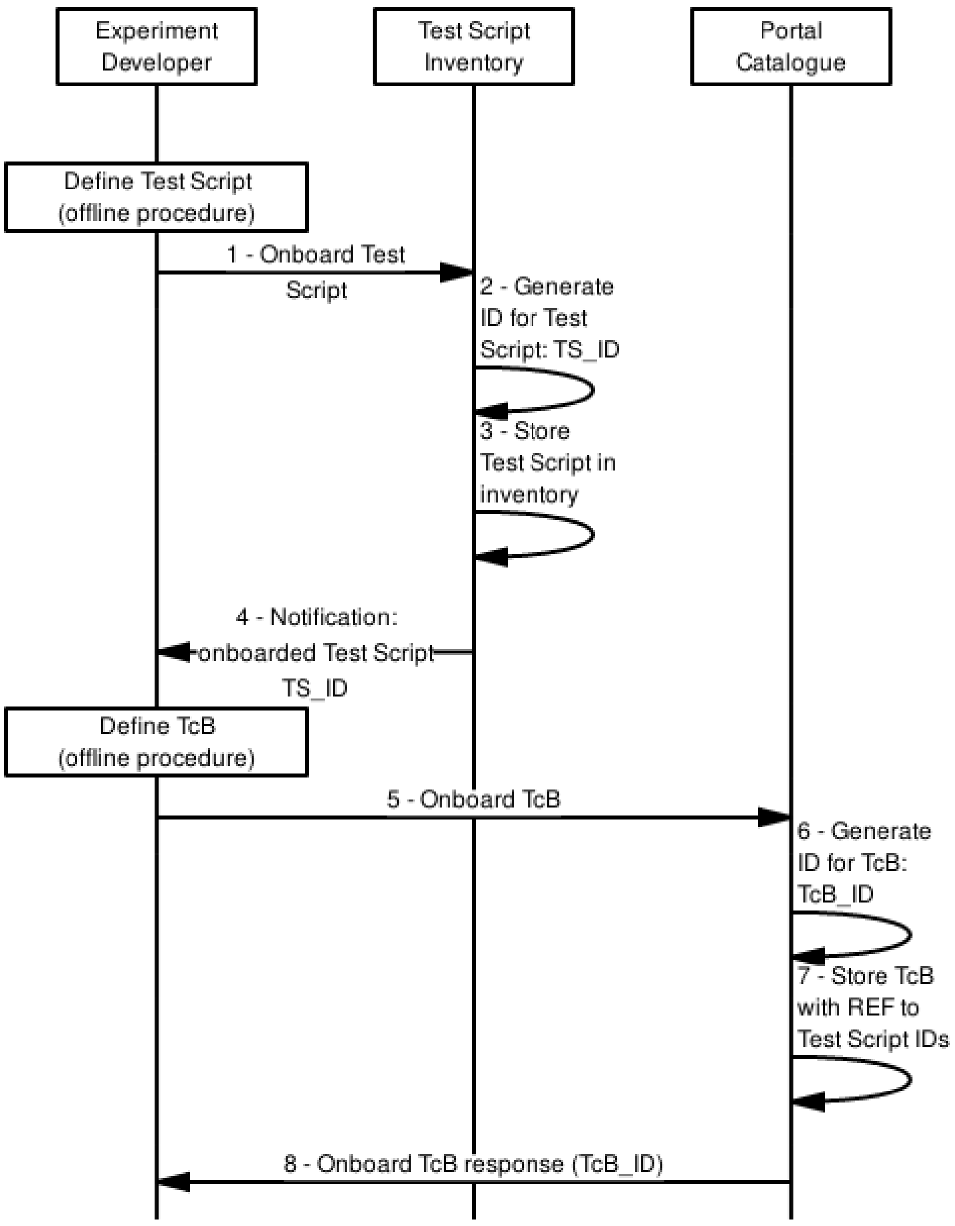

Experiment Developer
Test Script Inventory
Portal Catalogue
Define Test Script (offline procedure)
1 - Onboard Test Script
2 - Generate ID for Test Script: TS_ID
3 - Store Test Script in inventory
4 - Notification: onboarded Test Script TS_ID
Define TcB (offline procedure)
5 - Onboard TcB
6 - Generate ID for TcB: TcB_ID
7 - Store TcB with REF to Test Script IDs
8 - Onboard TcB response (TcB_ID)

Figure 10: Onboarding of Test Case Blueprint

3.1.4.2 Definition of Experiment Blueprint Details

This Section extends the workflow described in deliverable D4.1, Section 3.2.1.2, Figure 12, that shows the Definition of Experiment details in a summary way. Part of the workflow is reported in Figure 11 below. We are going to describe with more details the steps in each of the boxes between the dashed lines.

The definition of an Experiment Blueprint involves the Experiment Blueprint Builder component, that facilitates the developer in the definition of the Experiment Blueprint and in the interactions with the Portal Catalogue for the on-boarding. During the final steps of the ExpB definition, the Experiment Developer is asked to select and configure metrics, KPIs, and TcBs.

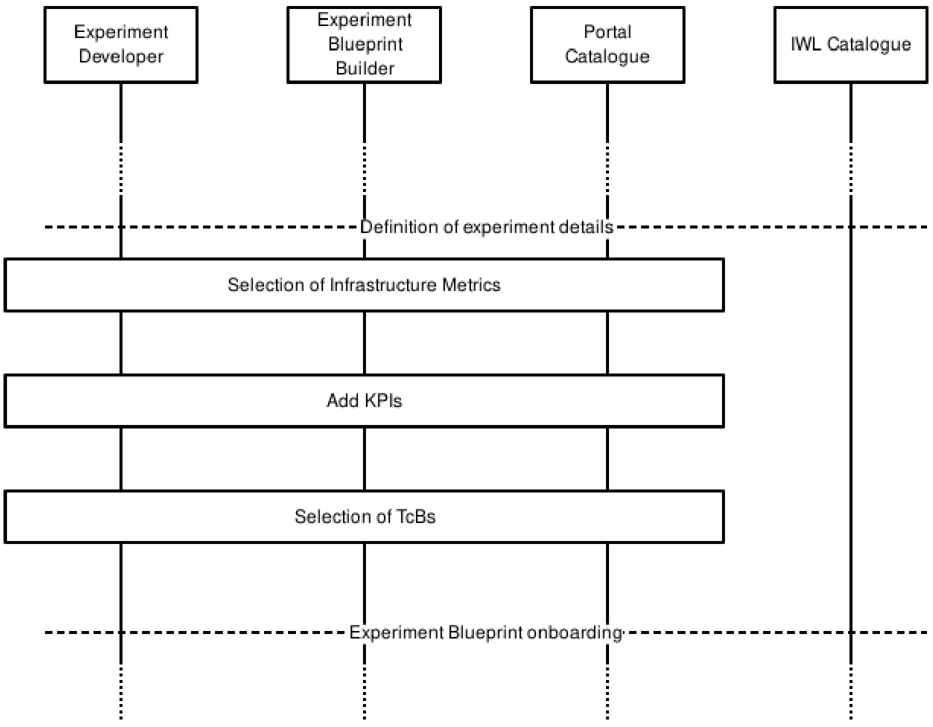

Experiment Developer
Experiment Blueprint Builder
Portal Catalogue
IWL Catalogue
Definition of experiment details
Selection of Infrastructure Metrics
Add KPIs
Selection of TcBs
Experiment Blueprint onboarding

Figure 11: High-level workflow for the definition of experiment details.

3.1.4.3 Selection of infrastructure metrics

Figure 12 shows the workflow for the selection of infrastructure metrics to be collected during the experiment execution.

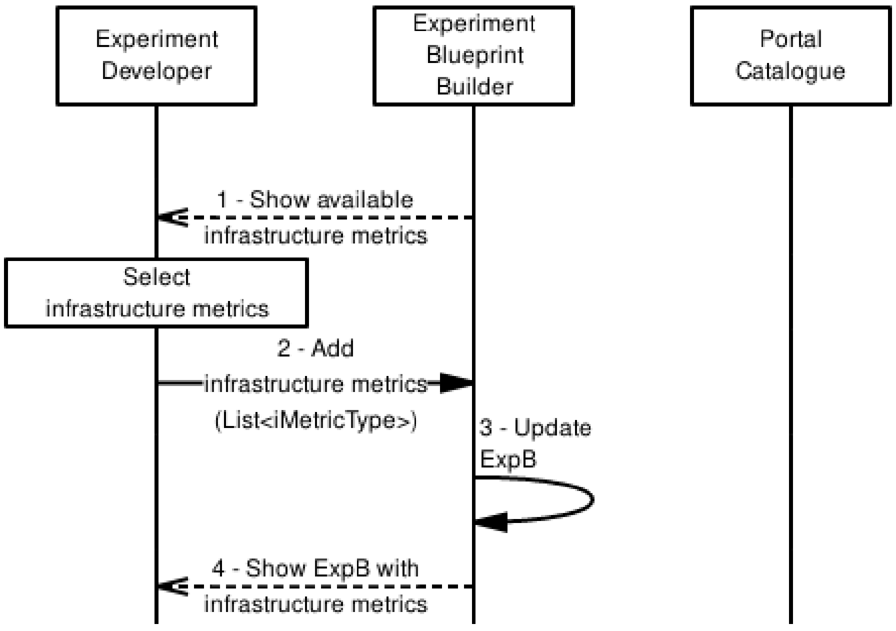

Experiment Developer
Experiment Blueprint Builder
Portal Catalogue
1 - Show available infrastructure metrics
Select infrastructure metrics
2 - Add infrastructure metrics (List<iMetricType>)
3 - Update ExpB
4 - Show ExpB with infrastructure metrics

Figure 12: Selection of infrastructure metrics for the ExpB

The Experiment Developer gets the list of available infrastructure metrics from the Experiment Blueprint Builder (Step 1). The Experiment Developer selects the infrastructure metrics he wants to be collected during the execution of the experiment and adds them to the ExpB (Step 2). The ExpB is updated (Step 3) and the new version is shown to the Experiment Developer (Step 4).

3.1.4.4 Definition of KPIs

Figure 13 shows the workflow for the definition of KPIs to be validated during the experiment.

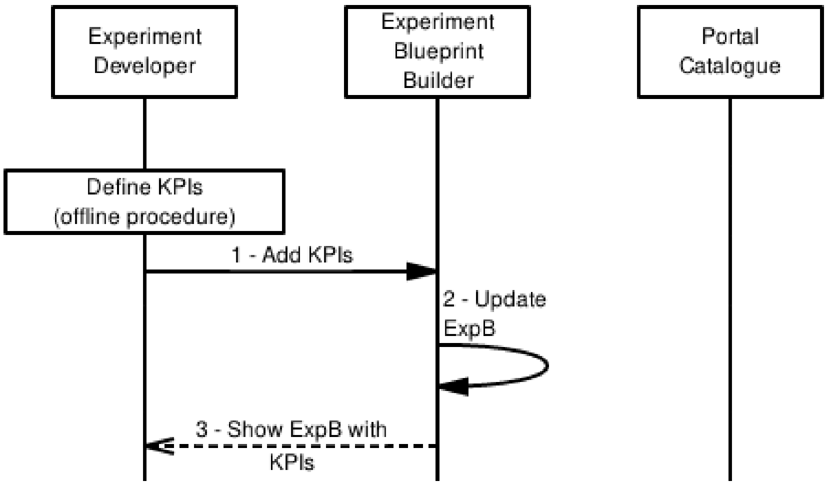


Figure 13: Definition of KPIs for the ExpB

The KPI definition happens offline with the collection of information from the verticals and the formalization into the model defined in Section 3.2.4. The Experiment Developer adds the KPIs to the ExpB he is building (Step 2). The Experiment Builder takes care of updating the ExpB (Step 2) and show the results to the Experiment Developer (Step 3).

3.1.4.5 Selection of TcBs

Figure 14 shows the workflow for the selection of TcBs, describing the execution of tests.

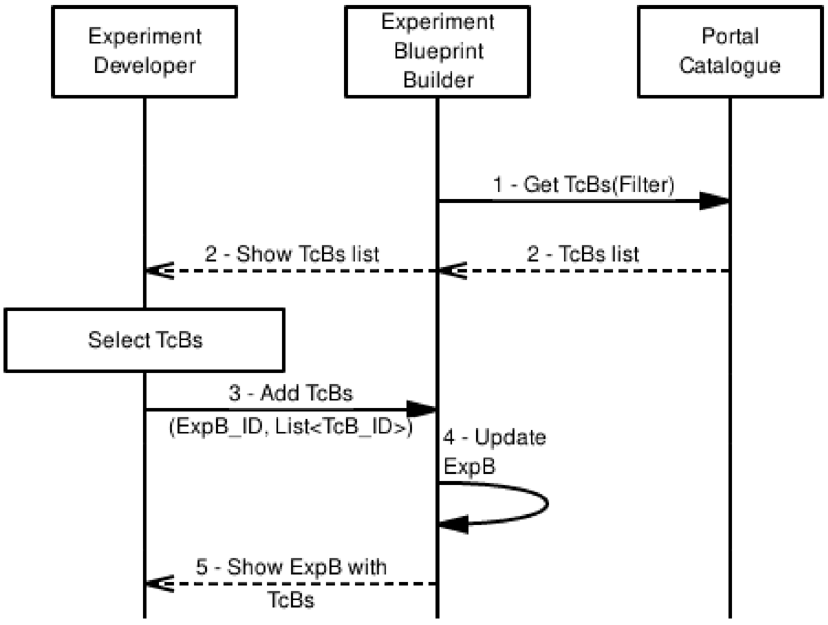


Figure 14: Selection of TcBs for the ExpB

The Experiment Blueprint Builder gets the list of available TcBs from the Portal Catalogue (Step 1) and shows it to the Experiment Developer (Step 2). The Experiment Developer selects the desired TcBs and sends them to the Experiment Blueprint Builder (Step 3). The Experiment Blueprint Builder updates the ExpB (Step 4) and shows the final result to the Experiment Developer (Step 5).

3.2 Test Plan templates

As already commented in D5.1, test plan templates are key tools generated in 5G EVE to gather the requirements from Verticals in an understandable language, valid for non-technical (in 5G) application developers. The completion of these test plans by all the participating experimenters is important for various reasons: the creation of blueprints, the assurance of the capabilities to host the experiments in the required sites, the definition of validation conditions and thresholds, etc. All these come from the low-level test plans that experiment developers generate from the high-level test plans coming from Verticals.

In the previous Deliverable, a complete test plan for the ASTI use case was presented, just as an example on how Verticals should fill in such templates. Since then, there has been ongoing work with the Verticals in order to complete all the respective test plans pertaining to the corresponding use cases. Also, there has been ongoing work to improve the template, as we realized from the interaction with Verticals that some concepts were not clear or needed an extension. This is summarized in the following section. Later on, we will also provide the first example of low-level test plan, again, coming from ASTI use case.

3.2.1 Update on high-level test plan templates

3.2.1.1 Update on the edition of the template

The template version that was included in D5.1, and first shared with Verticals, was the template version v0.2. Currently, we are working with template version v0.4. All changes in versions v0.3 and v0.4 derive from lessons learnt from the interaction with Verticals, as even mentioned in the template history (Table 5):

Table 5: Test Plan Template versions history

Version	Date	Modification	Modified by
v0.1	*09/01/2019*	*First draft*	
v0.2	*17/01/2019*	*Inclusion of KPIs and sub-cases*	*TID, based on global inputs*
v0.3	*22/03/2019*	*Improved explanation for validation conditions*	*TID, based on input by Verticals*
v0.4	*27/05/2019*	*Improved fields definition in Test Plan*	*TID, based on Verticals responses*

As a reminder, the first section in the test plan is called “Experiment Description”. Here, experimenters fill in the following items:

- Experiment description and expected capabilities (description)
- Experiment components (table)
- Experiment architecture (description)
- Meaningful KPIs (table)

A few changes have happened in this section from v0.2 to v0.4. In particular:

- The instructions in the “experiments components” part have been extended. We detected that Verticals were including in the table the expected tools to execute the tests, even if they were not providing them. For example, they were including traffic generators for data plane tests. The extension now clarifies that this section is there to identify the additional hosting requirements for the destination sites (for example, how much RAM is required for the Vertical VNFs), and that the required external tools are already in place, being 5G EVE who will determine which are the most appropriate for each test case. This also makes life a little bit easier for Verticals.
- Both the instructions and the table for the KPIs have been modified. We realized that, in the original table, we were presenting the same questions (“Is it measurable?” “Can it be derived from network parameters?”) for Network KPIs and Vertical KPIs. The fact is that these questions do not apply to Network KPIs, since 5G EVE will be measuring them in any case. Therefore, we split them in a new

table format. Also, it was clarified that the relationship between Vertical KPIs and Network KPIs, requested for those Vertical KPIs that could not be measured, had to be a mathematical relationship, to be able to make calculations. We were receiving responses of the type: "if one increases, the other one as well", which although interesting, did not help at validating test cases.

The second section in the template is the test plan itself, with one table per test case. Here, the main changes have to do with the instructions:

- The field "Target KPI", defined initially as the KPI per test case that wanted to be measured, is now defined as the KPI which will be used to validate the experiment, to avoid misunderstanding with the Network KPIs, all of which are actually measured.
- It is also included that the relationship between variable parameter in each test case and target KPI (in case this can be measured or mathematically derived) will be presented by default as part of the test results, using the same figures as the ones produced by the visualization tool. This accounts for some Verticals that were including essentially the same test case twice, one requesting information about the variable, and the other about the KPI, so they could relate each other.
- Finally, it is also clarified that test reports will include the values of the variable at which the experiment fails, if this is the case. This was considered from the beginning by 5G EVE but was explicitly requested by Verticals since it was not correctly explained in the template.

We are not including as part of this document an empty template to compare with that of D5.1, but those interested in checking the new look and feel can find the answers from Verticals in the Annex I.

3.2.1.2 Update on the responses to templates

Several new responses have been received from the moment D5.1 was delivered, mainly from those Verticals that are being used as first examples, aiding in the initial development of 5G EVE modules. In particular, there are four new responses, together with an update from ASTI (since they had initially responded over v0.2). All five are included fully in Annex I.

In this section, and as a summary, the initial section of each response is included below, in the same order as that of Annex I.

Utilities UC – Greek site

Experiment Description:

The experiment will test the performance of the Utility Platform in a 5G network infrastructure in two different deployments. In first deployment the Utility Services are running on the Cloud, while on second deployment the Utility Services are running on the MEC. The experiment will examine the performance of the Platform/Infrastructure in case of power failures.

Expected Capabilities:

A high performance and mobile network connecting the Voltage/Current Sensors, the CCTV Cameras, the Replay Nodes, the 5G Gateways with the Utilities Servers. A very fast power recovery is expected in cases of power failures with low power restoration time thanks to the 5G infrastructure and MEC capabilities.

Industry 4.0 UC – Spanish site

Experiment Description:

The experiment will test the virtualization of the control algorithms of AGVs. In the current state of the art the AGVs have an on-board PLC in charge of governing the internal control loop, which is collecting the information of the guiding sensors, taking the appropriate control decisions and generating the necessary signals to regulate the speed of the motors. During the experiment, this internal control loop will be relocated out of the AGV thanks to the 5G communication technologies.

Expected Capabilities:

A high performance and reliable mobile network connecting the AGVs to the platform with the virtual machines is required. The most challenging requirements to ensure the deployment of the solution in the future factories are the high reliability and the low latency and jitter.

Media & Entertainment UC – French site

Experiment Description:

The experiment will test the virtualization of some 5G components and especially the video 360° server. The main video use-cases typically deport the video server in the cloud. The main question is how far it can be deported especially when it corresponds to Virtual Reality (VR) contents. This is the main expected objective of this experiment.

Expected Capabilities:

To prevent nausea when showing high quality VR video format, it is mandatory that the E2E transmission provides low latency and jitter, the throughput coming in background for the user's point of view (possibility to adapt and degrade a little bit the video format). Also, the buffer state at the receiver has a great QoE impact on the video transmission. So, the most challenging requirement to ensure the deployment of the solution and expect the capabilities is the low latency. However, it is crucial to know exactly where (at what layer) the latency value must be measured.

Smart Tourism UC – Spanish site

Experiment Description:

The experiment aims at evaluation what benefits 5G brings to applications that take advantage of VR technology and real time 360-degree video to enhance the experience of participants in events or conventions. These applications will be run on top of a 4G network first, and then on top of a 5G network to compare the performance.

Expected Capabilities:

A test environment where tests can be executed over 4G or 5G environments as required. High bandwidth in the 5G solution is a must, as is a high-speed Internet connection for transmission of video flows (4K/8K) from AWS. It must also be possible to emulate additional delay in the network.

Transport UC – Italian site

Experiment Description:

The Transport vertical use cases can be tested applying for a tracking and recognition service experiment where an end user, equipped with a 5G terminal, moves around a 5G covered area performing different transportation modes (e.g. walking, biking, driving a vehicle or using public transportation means) and sending the collected data from the mobile device sensors (mainly GPS, accelerometer, magnetometer and gyroscope) to a backend platform where the real-time data are elaborated to estimate the end user transportation mode.

Expected Capabilities:

A high reliable and low-latency mobile network connecting the 5G terminal to the platform with the virtual machines hosting the tracking and recognition services is required. The most challenging requirements to *ensure the deployment of the above described solution in the future smart cities are the high reliability and the low latency.*

In the next months, the template will also be shared with the Verticals coming from ICT-19 projects. Since these are companies external to 5G EVE, the completion of the templates is even of higher importance since there is no (or little) previous interaction between the incoming Verticals and the local experiment developers.

3.2.2 Low-level test plan templates

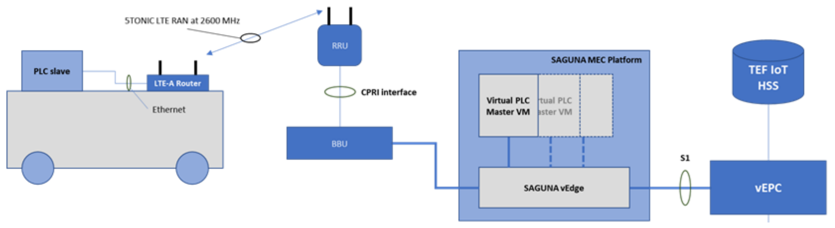
Low-level test plan templates, built by experiment developers, should contain all the meaningful information describing the experiment with a 5G EVE terminology. It must be noted that several inputs are key from these templates, like the desired infrastructure/application metrics, the meaningful KPIs (all required to build the test case blueprints), the test procedures that will derive in the test scripts, or the validation conditions that the Results Analysis & Validation module should execute.

The first version of the low-level test plan for the ASTI use case is included below (only the first Test Case, as an example; the full test plan is included in Annex II.1). It must be noted that improvements are also expected in this template, as happened with the high-level one. Of course, since low-level templates are built by 5G EVE personnel, not by experimenters, these inputs will not come from the latter. Internal iterations will happen when this template is distributed to new test developers for new test cases, until a final version is ready.

It has to be noted that some information is duplicated among test cases. The reason is that it is important that each test case is fully self-contained, as they could be further analysed by different experiment developers, even if all form a single test plan.

Industry 4.0 UC – Spanish site

Test Case 1	
Test Name:	*Effect of latency in Navigation Level*
Test Objective:	*Infrastructure metrics: “latency”* *Application metrics: “navigation_level” (obtained from Vertical’s components, not from traffic tools)* *Target KPI: “navigation_level”* *In this case, the KPI to be validated coincides with a metric that is provided by an ASTI component. The latency is the variable parameter, so it is also required to monitor this metric.*
Test Prerequisites:	*The “navigation_level” is measured by the AGV, and sent to the Master PLC as part of their communication. A mechanism will have to be implemented to propagate this metric to the Kafka bus. Options:* *a) Capture traffic between AGV and Master PLC, decode it, and publish in Kafka (this would require an additional element not provided by ASTI)* *b) Check if Master PLC can publish these values in Kafta* *The AGVs need specific space to deploy the circuit and associated components (e.g. battery charger) → this limits the hosting sites to 5TONIC*
Required Capabilities:	*In the hosting site we will need:* • *Compute capacity at MEC to host the Master PLC, a Windows based VM (requirements to be determined)* ○ *Local compute resources should be controlled by the local NFVO, so that instantiation can be triggered by the Multi-site NSO*

		• *A component emulating latency, under the control of the Runtime Configurator* • *Full 5G network: access + EPC*
Sub-Case 1	**Test Topology:**	 STONIC LTE RAN at 2600 MHz RRU PLC slave LTE-A Router Ethernet CPRI interface BBU SAGUNA MEC Platform Virtual PLC Master VM tual PLC ster VM SAGUNA vEdge TEF IoT HSS S1 vEPC *The AGV counts with a router to connect to the 5G network. The Master PLC is deployed at MEC, so the latency emulator must be placed somewhere between the AGV and the MEC.*
	Test Variables:	*The test variable is the latency. It must be noted that the additional latency introduced by the latency emulator is not the latency E2E. It is not trivial to measure this latency, since the Slave PLC at the AGV is not designed to do it. Either we can connect an additional device to the same router, or we should characterize the initial latency, and add it to the emulated one when results are presented.*
	Test Procedure:	***Deployment****: both the Master PLC and the latency emulator should be deployed real time as part of this phase.* ***Procedure****: after deployment, the AGV should start running (to be checked if this can be done automatically)* *Initial emulated "latency" = 0ms; "step" = 5ms* *Loop*:* *run experiment for 3 minutes* *latency = latency + step* **The ideal would be to execute the loop until the target KPI passes the validation threshold. However, since the validation cannot be done in real time at this stage, the number of runs of the loop is something to be determined. We might also provide a manual mechanism to stop the test, as it is evident to experimenters when the AGV is working far beyond the correct performance conditions.* ***Monitoring****: it is mandatory to receive the "navigation level" from the AGV continuously from the start of the experiment. This shall be visualized together with the E2E latency.*
	Expected results:	*Once the loop is finished, all the "navigation level" values will be compared with the specified threshold (still to be defined). If all the values are lower than the threshold, the result will be PASS, if at least one value is higher, the result will be FAIL.* *The test report should include the graphics "navigation level" versus "latency", plus the final result (PASS/FAIL), and the latency value in which the experiment fails, if this is the case.*

3.3 Experiment Preparation workflow

Test Design
Test Preparation
Test Execution and Monitoring
Test Performance Evaluation and Analysis

As already mentioned in the description of the 5G EVE Test and Validation Methodology (section 2.1), no automatic action is implemented during the Preparation phase. The declared initial conditions that must be configured before an experiment can start are human-driven actions which result from the interaction between experimenters and Site Managers who agree on a schedule for the experiment and the preparatory configurations of the testing environment (tools, devices, testers, etc.). There is no specific workflow to export in this case but rather a set of best practices and procedures specific of the Site Manager organizations which will be involved in the 5G EVE experiment execution. The section has been maintained for completeness, to avoid introducing gaps in the narrative.

3.4 Experiment Execution workflow and APIs

Test Design
Test Preparation
Test Execution and Monitoring
Test Performance Evaluation and Analysis

In Deliverables from other Work Packages, and justified by the appropriate understanding of the workflows there presented, components from WP5 (either the Experiment Execution Manager, or the Results Analysis and Validation module, or both) have been included participating in experiment execution related activities. This section covers the part of the experiment execution workflow specific to WP5, including also the operations that will enable the validation and monitoring procedures of the methodology described in section 2.1.

3.4.1 Experiment Execution workflow: testing, validation and monitoring

Due to the length of this workflow, we have divided it in three parts in this section. The full workflow can be found in Figure 38, in the Annex III.

The first part (Figure 15) relies mainly on the WP3/WP4 workflows for the instantiation of an experiment, and extends it to show how the Kafka topics are notified to the Results Analysis and Validation module, and to the Data Collection and Storage modules (WP4). The latter is included as it will ultimate end in the visualization (monitoring) of metrics and KPIs.

When the Vertical triggers the execution of an experiment, from the GUI, a first request is sent to the ELM (1). The initial part of the workflow can be found in detail in WP3 and WP4 Deliverables, but here suffice to say that the ELM retrieves the required Experiment Descriptors from the Portal Catalogue, and triggers the Network Service (NS) instantiation (2) towards the Multi-site Network Service Orchestrator (NSO). This module, in turn, retrieves the NS Descriptor from the Multi-site Catalogue, and contacts the local orchestrator(s) for the instantiation. Once it is finished, and the experiment is deployed, it responds to the ELM (2') with the NS Instance ID. The ELM then queries the Multi-site NSO again (3), to retrieve all the details related to the desired NSI, information that is provided in (3').

At this stage, ELM could trigger towards the EEM the execution of the experiment, had testing been the only objective in 5G EVE. However, for the validation and monitoring procedures, ELM has yet to create the name of the Kafka topics associated with this experiment, by concatenating the global name of metrics/KPIs with the NSI ID. This will ensure that results are not mixed in the Data Collection Manager.

After generating the name of the topics, these are sent to the Data Collection Manager (4), and are automatically redistributed to the Results Analysis & Validation (5) and the Data Collection & Storage (5bis) modules. Although not depicted in the figure, for simplicity, this automatic redirection of topics is possible because both modules had initially subscribed to the signalling topics of the Data Collection Manager. This subscription is a strict requirement for these modules before any experimentation is to be run on top of 5G EVE.

(5') and (5bis') show the subscription of these validation and monitoring modules to the required topics in the Kafka bus, typically metrics and/or KPIs, and from that moment on, they are ready to receive the target measurements as soon as they are generated and published into the bus.

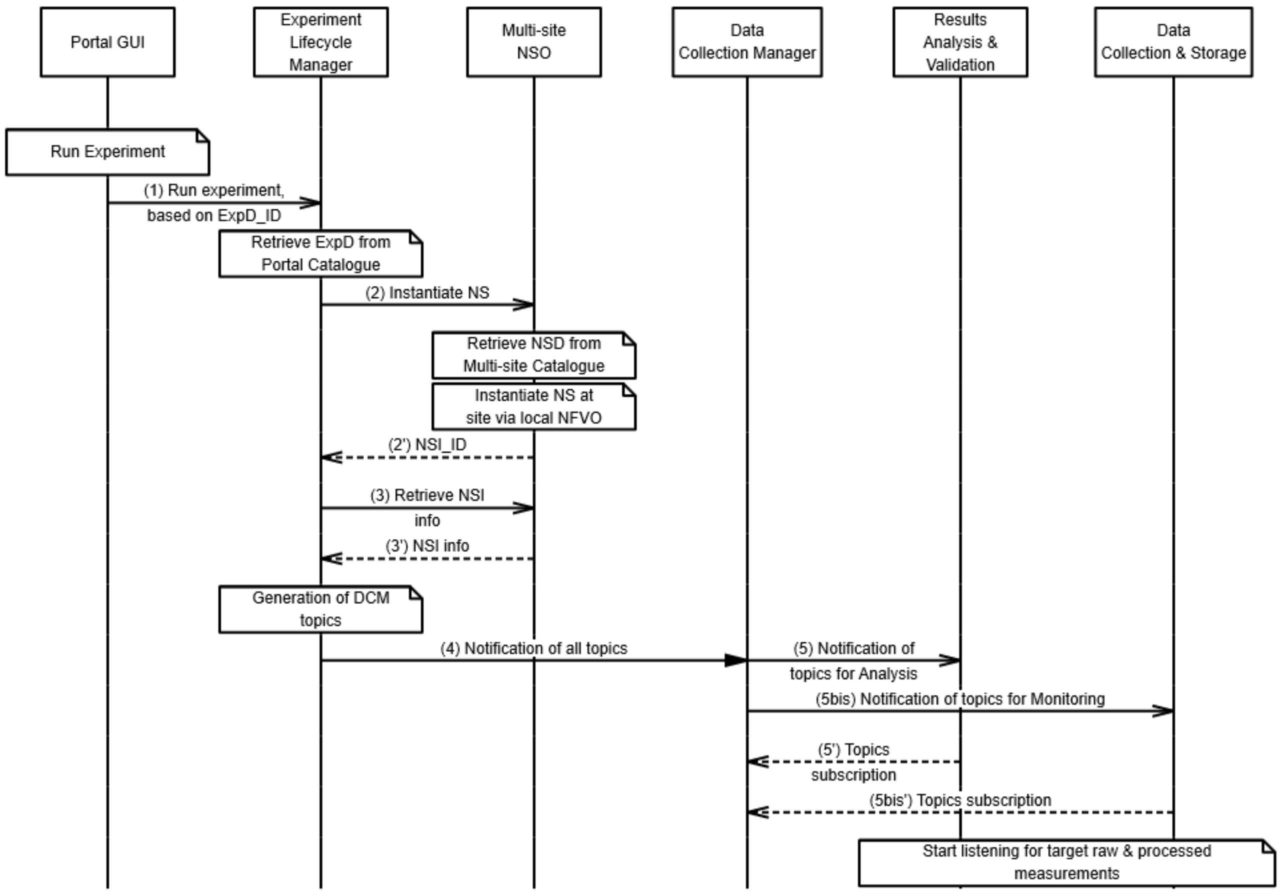

Portal GUI
Experiment Lifecycle Manager
Multi-site NSO
Data Collection Manager
Results Analysis & Validation
Data Collection & Storage
Run Experiment
(1) Run experiment, based on ExpD_ID
Retrieve ExpD from Portal Catalogue
(2) Instantiate NS
Retrieve NSD from Multi-site Catalogue
Instantiate NS at site via local NFVO
(2') NSI_ID
(3) Retrieve NSI info
(3') NSI info
Generation of DCM topics
(4) Notification of all topics
(5) Notification of topics for Analysis
(5bis) Notification of topics for Monitoring
(5') Topics subscription
(5bis') Topics subscription
Start listening for target raw & processed measurements

Figure 15: 5G EVE Experiment Execution workflow – part 1

The second part of the workflow is depicted in Figure 16.

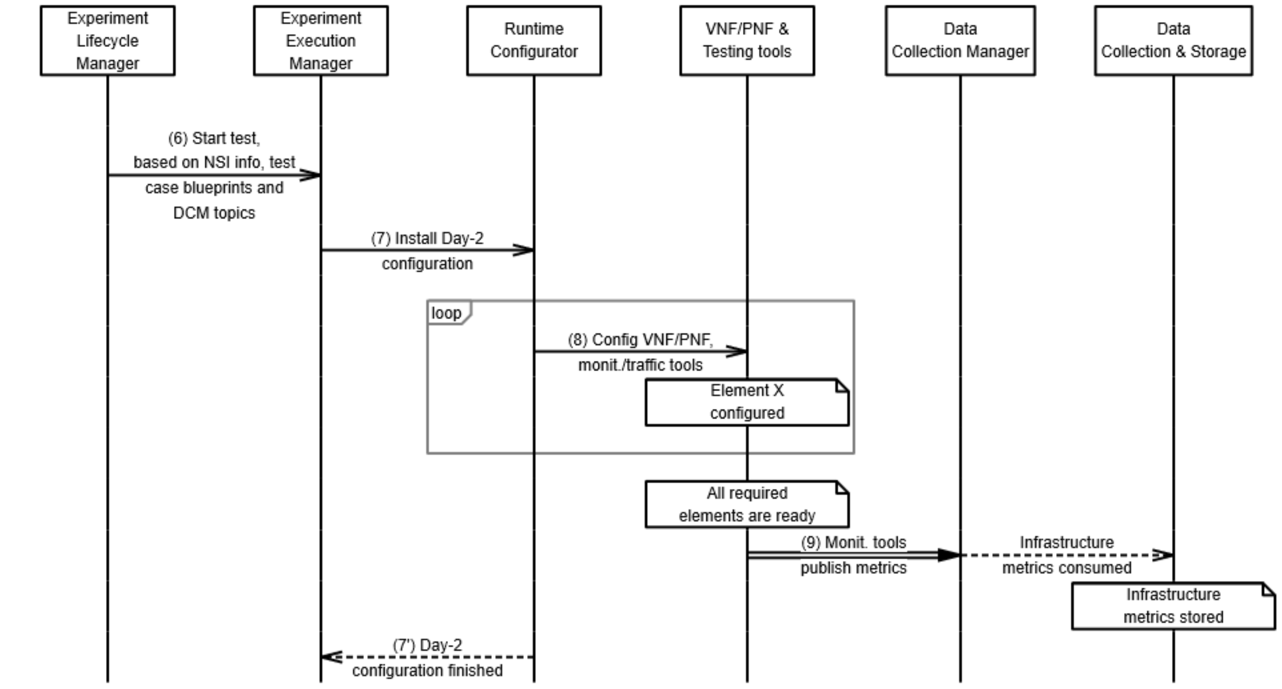

Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
VNF/PNF & Testing tools
Data Collection Manager
Data Collection & Storage
(6) Start test, based on NSI info, test case blueprints and DCM topics
(7) Install Day-2 configuration
loop
(8) Config VNF/PNF, monit./traffic tools
Element X configured
All required elements are ready
(9) Monit. tools publish metrics
Infrastructure metrics consumed
Infrastructure metrics stored
(7') Day-2 configuration finished

Figure 16: 5G EVE Experiment Execution workflow – part 2

Now the ELM is ready to trigger the execution of the experiment towards the EEM, providing information like the NSI ID, all the deployment-related parameters of the NSI received from the Multi-site NSO, the test case blueprints and the Kafka topics (6).

The EEM, as already mentioned, is the one in charge of governing the testing and validation procedures, but before this, the deployed experiment must be correctly configured. This is done via the Runtime Configurator (7), which executes a loop to configure (8) each site element (VNFs, PNFs, monitoring tools or traffic generation tools). Strictly speaking, there is not a response to message (8), since the Runtime Configurator will be running Ansible playbooks in the desired elements, and this does not produce any message back. The Runtime Configurator itself will determine when this day-2 configuration process is finished ("All required elements are ready" box in the figure), and will provide a response to the EEM (7').

Part of this configuration to the site components, again, has to do with the validation and monitoring processes, as the Runtime Configurator will be providing the topics for publication to the Beats modules in each component. Once these components are configured, and metrics start to be available, then they automatically publish the desired information in the Kafka bus. At this stage, since the specific tests have not yet started, it is envisioned that only the infrastructure metrics from monitoring tools will be available, and these are the ones that are published (9) and made available to the Data Collection & Storage module (for later visualization). Although not explicitly included in the figure (for simplicity), the RAV module could also need, in some experiments, the infrastructure metrics, so the Kafka bus could equivalently make them available for it. The final part of the workflow is included in Figure 17:

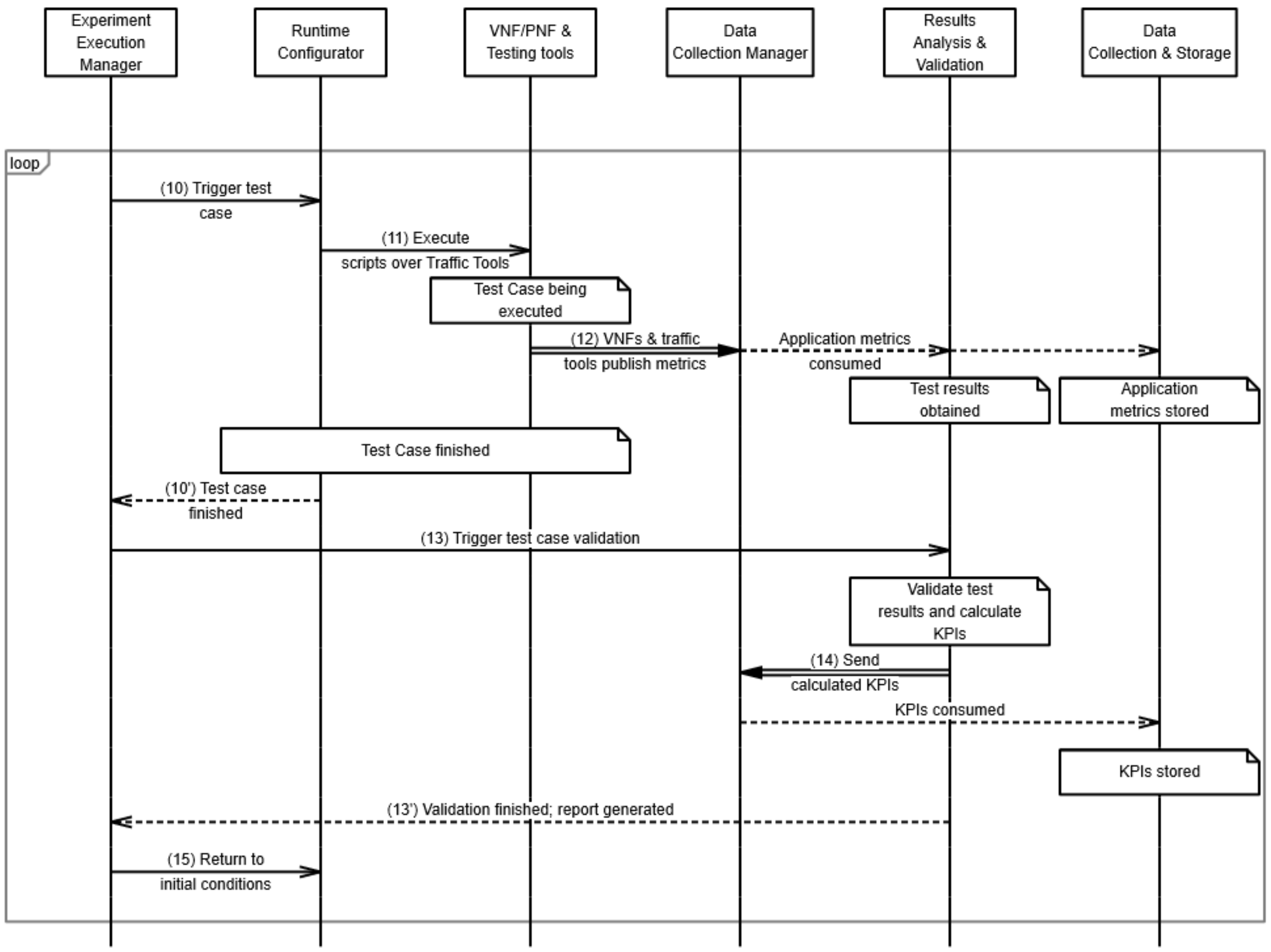

Experiment Execution Manager
Runtime Configurator
VNF/PNF & Testing tools
Data Collection Manager
Results Analysis & Validation
Data Collection & Storage
loop
(10) Trigger test case
(11) Execute scripts over Traffic Tools
Test Case being executed
(12) VNFs & traffic tools publish metrics
Application metrics consumed
Test results obtained
Application metrics stored
Test Case finished
(10') Test case finished
(13) Trigger test case validation
Validate test results and calculate KPIs
(14) Send calculated KPIs
KPIs consumed
KPIs stored
(13') Validation finished; report generated
(15) Return to initial conditions

Figure 17: 5G EVE Experiment Execution workflow – part 3

Once the experiment is correctly deployed and configured, testing activities can start. Since more than one Test Case can be executed in the same experiment, the EEM will execute them in a loop, one at a time. Again, for simplicity, this is not included in the figure, but the EEM first gets the scripts from the Test Scripts Repository, and then triggers (10) the execution of the experiment at the Runtime Configurator.

Via Ansible playbooks, the Runtime Configurator executes its own scripts over the experiment components (11), typically over traffic tools like traffic generators or network impairments emulators. It can be expected at this moment that more metrics, the application metrics, are now available, so VNFs and traffic tools will be publishing them at the bus (12). In this case, the consumer will not only be the pre-visualization module, but also the Results Analysis & Validation (RAV) component, which will obtain therefore important test results.

Similarly to step (8), there is not a specific response to the execution of the test playbooks (there is not a (11') message); therefore, when the Runtime Configurator considers the test finished, it sends a notification to the EEM (10'). The validation procedure can the start.

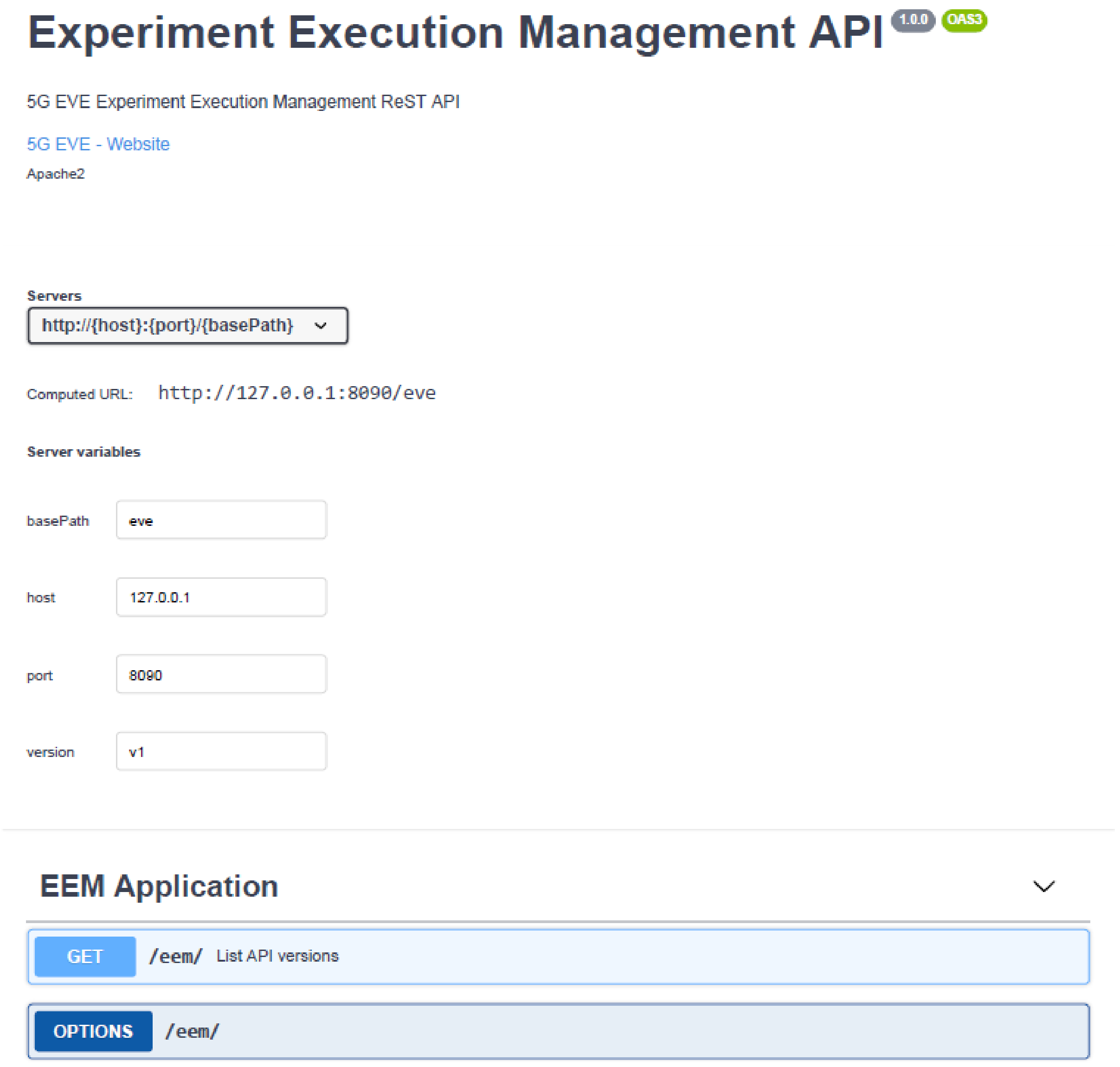
EEM triggers this process at the RAV (13), which makes the required calculations to obtain KPIs. RAV then, on the one side publishes the KPIs at the Kafka (14), so they can be eventually visualized (the Data Collection & Storage components consume these KPIs from the bus), and on the other side, compares the KPIs with the expected values, determining the result of the experiment (PASS/FAIL). Once this comparison is finished, it sends back the test report to the EEM (13'), which considers the Test Case finished, triggers the return to initial conditions message (15) and moves on to the next Test Case.

The last two messages in the workflow can be seen in the full version of Figure 38 (Annex III): they correspond to (6'), where EEM declares to the ELM that the test is finished (and sends the full report with the results from all Test Cases), and then (1'), which is the notification and report submission from the ELM to the Portal.

3.4.2 Experiment Lifecycle Manager (ELM) and Experiment Execution Manager (EEM) API

In compliance with the workflows described in section 3.4.1, the EEM API is designed to allow external components (and in particular the Experiment Lifecycle Manager, ELM) to configure executions and run Test Cases of an experiments.

This API represents the north-bound interface of the EEM. It is consumed mainly by the Experiment Lifecycle Manager according to the workflows described in Section 3.4.1, as result of the triggering of the experiment execution from the Vertical user through the 5G EVE Portal. The API is structured in four sections as shown in the following Figure 18, Figure 19 and Figure 20.



Experiment Execution Management API 1.0.0 OAS3

5G EVE Experiment Execution Management ReST API

5G EVE - Website

Apache2

Servers

http://{host}:{port}/{basePath}

Computed URL: `http://127.0.0.1:8090/eve`

Server variables

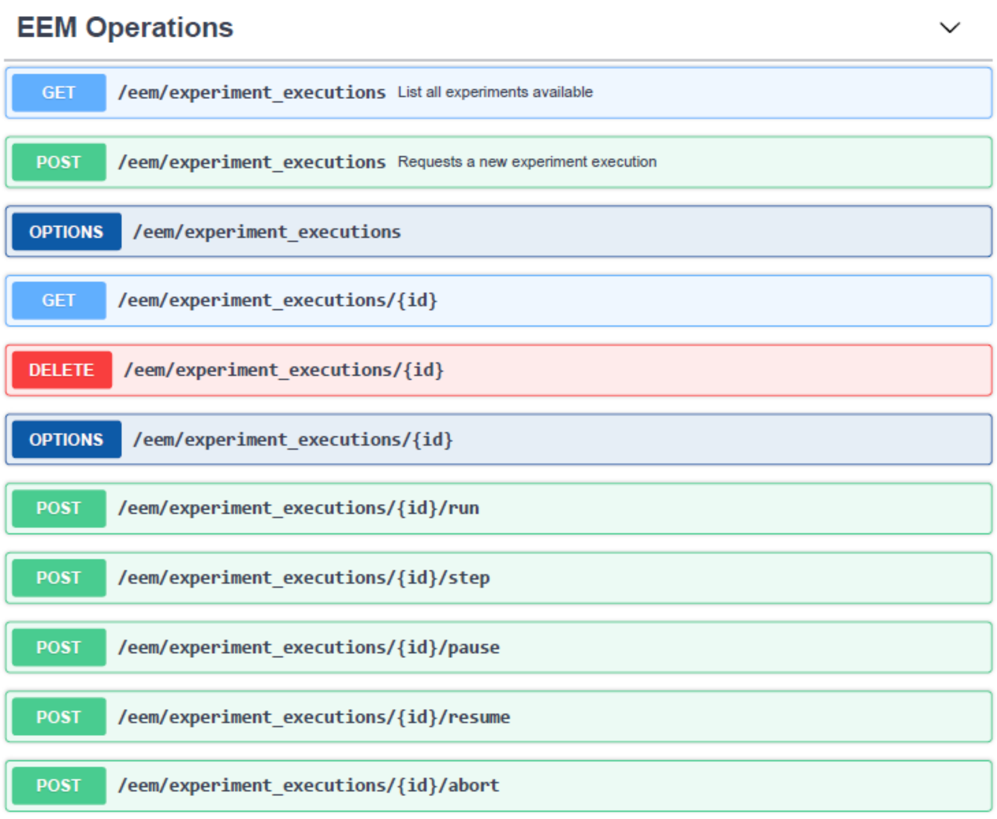
basePath	eve
host	127.0.0.1
port	8090
version	v1

EEM Application

GET `/eem/` List API versions

OPTIONS `/eem/`

Figure 18: North-bound interface of EEM (part1, base application)



EEM Operations

GET /eem/experiment_executions List all experiments available

POST /eem/experiment_executions Requests a new experiment execution

OPTIONS /eem/experiment_executions

GET /eem/experiment_executions/{id}

DELETE /eem/experiment_executions/{id}

OPTIONS /eem/experiment_executions/{id}

POST /eem/experiment_executions/{id}/run

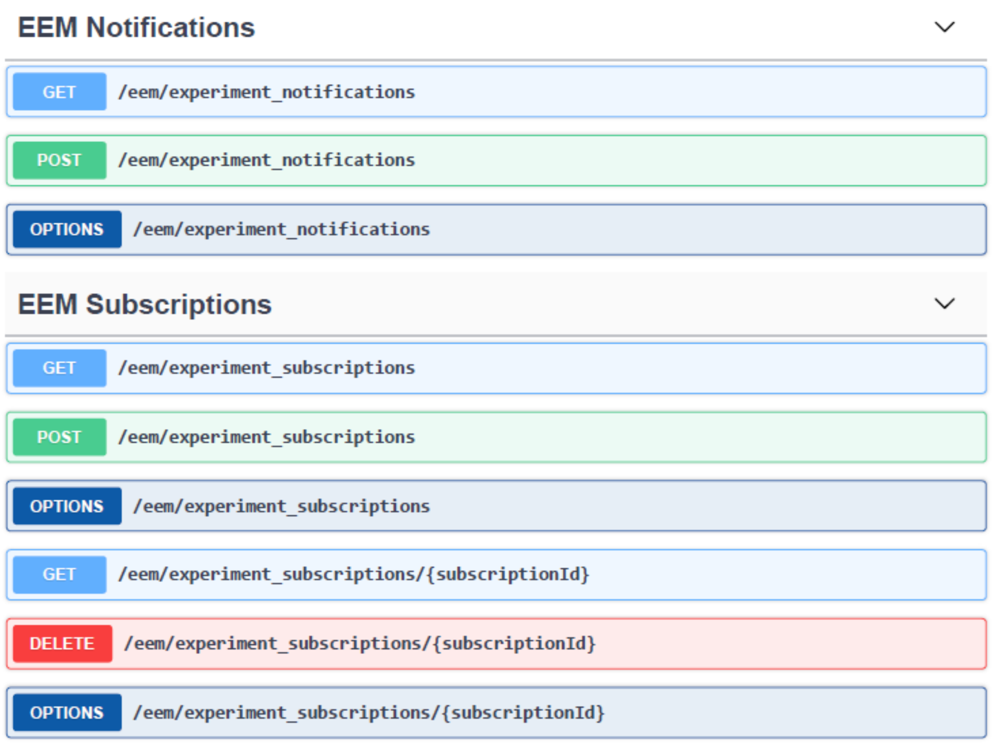
POST /eem/experiment_executions/{id}/step

POST /eem/experiment_executions/{id}/pause

POST /eem/experiment_executions/{id}/resume

POST /eem/experiment_executions/{id}/abort

Figure 19: North-bound interface of the EEM (part2, operations)



EEM Notifications

GET /eem/experiment_notifications

POST /eem/experiment_notifications

OPTIONS /eem/experiment_notifications

EEM Subscriptions

GET /eem/experiment_subscriptions

POST /eem/experiment_subscriptions

OPTIONS /eem/experiment_subscriptions

GET /eem/experiment_subscriptions/{subscriptionId}

DELETE /eem/experiment_subscriptions/{subscriptionId}

OPTIONS /eem/experiment_subscriptions/{subscriptionId}

Figure 20: North-bound interface of the EEM (part3, notifications & subscriptions)

The groups of actions exposed via the EEM ReST API respond to a specific scope of the targeted resources and actions:

- *EEM Application*: Allow the clients to identify the available versions of the application.
- *EEM Operations*: Groups all the methods related to the execution of the experiment.
- *EEM Subscription*: Groups all the methods related to the publish/subscribe paradigm used from the clients to subscribe to their executed experiments.
- *EEM Notifications*: Groups all the methods related to the subscriptions of the EEM to the Runtime Configurator and the Results Analysis and Validation (RAV). In this case, the EEM receives notifications related to the state of the Day-2 configuration and the progress of the running execution (in case the EEM subscribes to the Runtime Configurator for these resources), and related to the validation state (if it subscribes to the RAV).

Through EEM Operations, a Vertical (or the ELM module on his/her behalf) can manage the execution of an experiment.

Through the endpoint /experiment_executions the main core actions on EEM can be activated. The HTTP GET method is used to return list of all the instances of experiment executions managed by EEM with related experiments information. Through the POST method it is possible to create a new instance of an experiment execution. Once the ELM receives the identifier, it can request the execution of the experiment. In this case, the ELM performs a HTTP POST /experiment_execution/{id}/run providing all the information to the EEM that will be used to retrieve the necessary information to get the network service and the test cases blueprints.

The EEM allows two different types of run:

- RUN_ALL: The experiment will be run without intermediate phases. Once started, the Vertical will not have the possibility to interfere with the execution of the experiment, unless for the abort action.
- RUN_IN_STEPS: The experiment will be running as a pipeline and each stage need to be started by the Vertical through the EVE Platform. The state of the experiment after each stage will be "PAUSED".

The type of run is a query parameter named "runType". The default value is "RUN_ALL".

The subscribed clients receive notifications upon state changes related to the experiment.

An execution can be aborted in any moment through the HTTP POST /experiment_executions/{id}/abort method. This action is irreversible and stops the running experiment by rolling back all the pending test cases and statuses of the involved systems. For all the cases where the experiment execution is implemented as a pipeline, an HTTP POST on the endpoints /experiment_executions/{id}/pause and /experiment_executions/{id}/step allows to run the experiment execution in interactive mode. When the pause action is triggered, the execution stops upon completion of the ongoing Test Case execution run; from that point onwards, each step is explicitly activated through a step action.

The EEM can request the state of the execution of the experiment triggering the GET method on the endpoint /experiment_executions/{id}.

If needed, a Vertical can remove the data related to the execution of an experiment by triggering the HTTP DELETE method on the /experiment_executions/{id}.

The EEM Subscriptions section allows the Vertical to subscribe to experiment execution notifications. The POST method on the /experiment_subscriptions allows to subscribe to a specific execution instance.

The info model related to the endpoints discussed above includes different datatypes reported in the following tables (Table 6 to Table 10).

The ExperimentExecutionRequest data is expected to be provided by the ELM to the EEM when an experiment execution starts (/experiment_executions/{id}/run).

Table 6: ExperimentExecutionRequest data type

Name	Type
executionId*	String
experimentDescriptorId*	String
nsInstanceId*	String
testCaseDescriptorConfiguration	Map<String, Map<String, String>>

* mandatory fields

The ExperimentExecutionResponse contains information related to the state of execution, and if available, the URL where the report is stored. It contains also a map with the list of the test cases requested to be executed and their result.

Table 7: ExperimentExecutionResponse data type

Name	Type
eemSubscriptionId*	string
executionId*	string Identifier of the executed experiment
reportUrl	string URL containing the report of the execution
state*	Enum: [INIT, CONFIGURING, RUNNING, RUNNING_STEP, PAUSED, VALIDATING, COMPLETED, ABORTING, ABORTED, FAILED]
testCaseResult	Map<String, ExecutionResult>

* mandatory fields

The EEM should provide an URI where the notification will be sent, and a subscription type.

Table 8: ExperimentExecutionSubscriptionRequest

Name	Type
callbackURI*	URI
executionId*	Experiment execution identifier
subscriptionType*	Enum: [EXPERIMENT_EXECUTION_CHANGE_STATE]

* mandatory fields

As response, the EEM should send to the ELM the ExperimentExecutionSubscriptionResponse.

Table 9: ExperimentExecutionSubscriptionResponse

Name	Type
id*	string
callbackURI*	URI
executionId*	string

subscriptionType*	Enum: [EXPERIMENT_EXECUTION_CHANGE_STATE]

* mandatory fields

The EEM will notify the ELM over the callback URI each time the execution of the experiment change its state.

The Vertical can request the list of subscriptions or a single subscription using the GET method on the respective endpoint: /experiment_subscriptions for all the subscriptions and /experiment_subscriptions/{subscriptionId} for a single subscription, identified by the subscriptionId. The payload of the notification sent to the ELM, when the state of the execution changes is described at the following table:

Table 10: ExperimentExecutionNotification

Name	Type
subscriptionId*	String
callbackURI*	URI
state*	Enum: [INIT, CONFIGURING, RUNNING, RUNNING_STEP, PAUSED, VALIDATING, COMPLETED, ABORTING, ABORTED, FAILED]
executionId*	String

* mandatory fields

The EEM Notifications section offers an endpoint to the Runtime Configurator and RAV for the notifications during the Day-2 configuration and validation phases. The Runtime Configurator/RAV should include as payload of the POST request the following data model.

Table 11: Runtime configurator/RAV data model for notifications service

Name	Type
configurationChangeState*	Enum: [CONFIGURING, CONFIGURED, CONFIGURATION_FAILED, VALIDATING, VALIDATED, VALIDATION_ERROR]
executionId*	String
notificationInfo*	String
notificationError	ErrorInfo

* mandatory fields

3.4.3 Experiment Execution Manager (EEM) and Runtime Configurator API

In 5G EVE, we have determined that the "server side" is the one responsible of defining the different APIs. In this case, the EEM acts as a client of the Runtime Configurator, so the specification of this API has been led by WP3. In that sense, the spec of the Runtime Configurator's NBI was started in D3.2 – *Interworking Reference Model* – [6], and updated in D3.3 – *First implementation of the interworking reference model* – [7].

All the details regarding this API can be found in those references, so only the most updated table will be included here for completeness:

Table 12: EEM – Runtime Configurator API

Runtime Configurator NBI		
Execution of templates for given steps of a test case		
Description	Invocation of the templates related to a given operation in a test case, in order to i) provide Day-2 configuration to the VNFs/PNFs referenced in the test case, or ii) trigger the execution of a test step	
Reference Standards	OpenSSH protocol or REST API interface in case of incompatibilities.	
Operations Exposed	***Information Exchanged***	***Information Model***
Execute a template for providing configuration operations (execute_config_templates operation)	• A list with the templates ID. • The experiment execution ID. It will return the result of the execution of the commands included in the used templates.	In case of using OpenSSH, the input information will be provided in a specific script to be executed in the Runtime Configurator server, which will include the necessary logic to process it, obtain the IP addressing configuration of the components to be configured and apply the templates to these components.
Execute a template for providing test-related operations (execute_exec_templates operation)	• A list with the templates ID. • The experiment execution ID. It will return the result of the execution of the commands included in the used templates.	In case of using OpenSSH, the input information will be provided in a specific script to be executed in the Runtime Configurator server, which will include the necessary logic to process it, obtain the IP addressing configuration of the components to be configured and apply the templates to these components.

3.5 Internal workflows and APIs

Two internal operations are the ones that need to be tackled in this section. Both relate to Figure 17 in section 3.4.1, where the last part of the experiment execution workflow was depicted.

In step (10) we had the EEM triggering the execution of the experiment at the Runtime Configurator. For simplicity it did not appear in the workflow, but it was commented that, prior to that, the EEM got the required scripts from the Test Scripts Repository. This repository has similar functions as those of the Multi-site Catalogue in WP3, when the Multi-site NSO requests the needed descriptors. Therefore, a similar API may be implemented, and it will not be included here since it can be found in Deliverables D3.2 [6] and D3.3 [7].

The other operation is depicted in step (13), when the EEM triggers the validation process at the RAV.

The RAV module exposes an API towards the other platform entities in order to provide them with operational control. This API includes the data types for the EEM to execute the mentioned trigger, but it also contains other data types. For example, during the experiment setup procedure, the ELM notifies the DCM about the topics that will be used for publishing the metrics, and then the DCM notifies the RAV module about these topics and the broker information. An example of the API provided to the RAV module can be seen in Table 13.

Table 13: RAV module's configuration message fields

Field	Description
brokerAddr	IP address and port of the Broker
name	The name of the topic to subscribe to for the specific KPI
kpi	The specific KPI that will be monitored

analysis	Analysis method for the published messages. Can have various functionalities (parsing, normalizing, applying functions to the metrics, etc…)
conditions	The conditions based on which the target KPI will be validated

The analysis field is a selection of functionalities that can be mapped to different combinations of parsing and organizing the published raw or processed metrics. Outlier detection, message sanitization and enrichment of the metrics with additional information related to some other process, like the Performance Diagnostics, are only a few of the possible functionalities that can be implemented in the analysis process. Additional configuration options for the module may be added depending on the requirements of the current and future Use cases.

Finally, the RAV, to let the EEM know that the validation process has been finished, will generate an explicit notification – step (13’) in Figure 17 – in which it will also provide the test report, normally in the form of an URL.

4 Design of the testing and validation framework components

4.1 Experiment Execution Manager (EEM)

The Experiment Execution Manager is a self-contained and stand-alone component written in Java, using Spring-boot framework. It provides support to the 5G EVE Platform for the configuration and the execution of the experiments.

In particular, the EEM component is able to configure and run a set of test cases for a given experiment through the API described in section 3.4.2 and the workflow described in section 3.4.1.

When a request of a run is triggered (step 6 on the workflow described in section 3.4.1), the EEM contacts the MSNO to retrieve the NSInfo, given the nsInstanceId and the Catalogue to retrieve the Experiment Descriptors related to the execution.

Once all the information is available, the EEM starts the translation of the test cases descriptors to Robot Framework files that are sent to the Runtime configurator, which is in charge of the day-2 configuration of the VNFs and the execution of the tests.

The internal state machine related to an experiment execution is described in the following Figure 21.

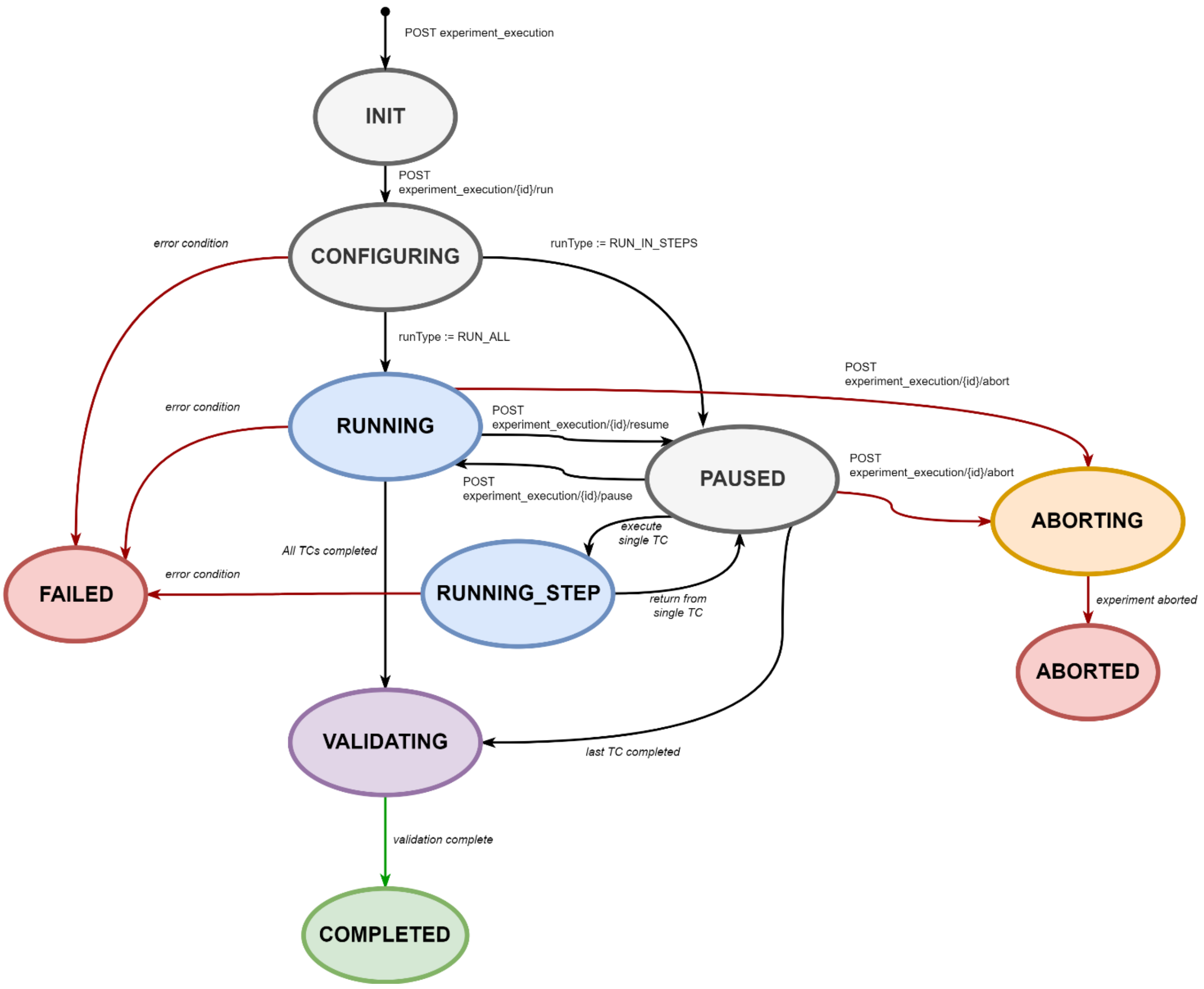


Figure 21: EEM Finite state Machine for an experiment execution

Upon creation (POST /experiment_executions), a unique experiment execution identified is generated and the execution transits in INIT state, ready to receive subscriptions for notifications or run triggers. Upon a run command, the experiment transits into CONFIGURING state, where all the configurations on Runtime Configurator (for VNFs and NS), on monitoring platform and RAV are implemented. In this state, the translation of both the Test Case descriptors into Robot Framework files and Jenkins pipeline jobs is implemented. When all the configurations are completed, the experiment transits into RUNNING or PAUSED depending of the run type specified by the API client. In case of experiment execution in steps, the EEM controls the completion of each TC execution and waits for the explicit trigger to progress via API (POST /experiment_executions/{id}/step). In case of complete run (runType := RUN_ALL) all the TCs are executed sequentially as specified in the experiment descriptor.

When all the TCs are completed or the last TC is completed, the experiment transits in VALIDATING, thus interacting with the RAV to start the KPI analysis phase and the production of the reports.

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. The high-level design of the EEM module under development is depicted in Figure 22.

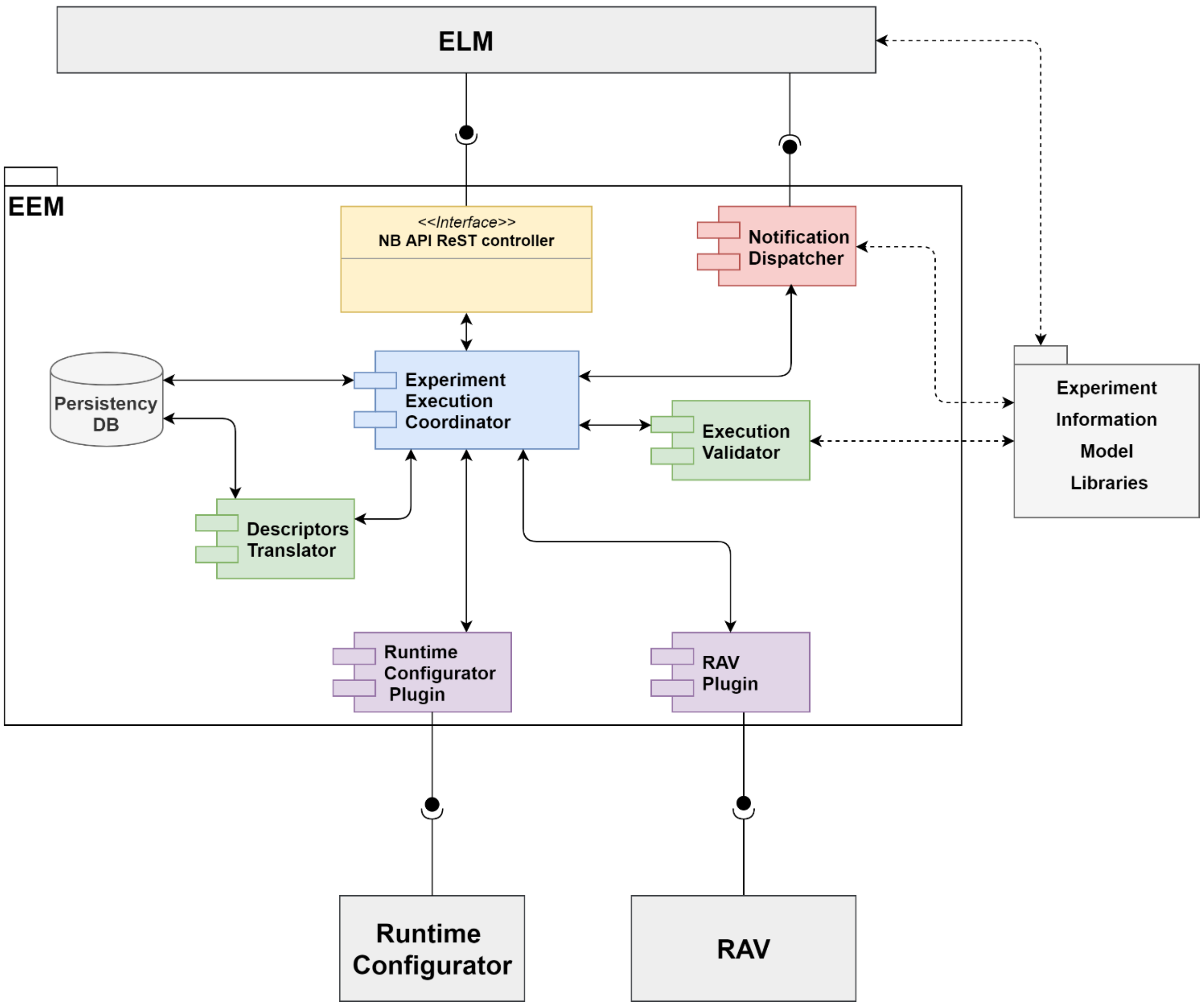


Figure 22: EEM high level design

The EEM builds around the core functionality of an Experiment Execution Coordinator, which receives parsed commands related to the managed experiment and executes the following sequence of operations:

1. validate experiment execution request in an Execution Validator
2. trigger translation of experiment execution descriptors (ExpD, CD, TcD, NSD) into Robot Framework files and Jenkins configuration descriptors
3. dispatch the configuration files to RAV and Runtime Configurators via related plugins
4. write received records and configurations in persistent storage
5. run and evolve the EEM Finite State Machine depending on experiment configurations and EEM API commands
6. dispatch experiment notifications to subscribed clients

4.2 Results Analysis & Validation (RAV)

The initial implementation of the RAV module for the disjoint testing and validations tools delivered with D5.1 was as a sub-functionality of the Robot Framework testing module. In order to enhance its capabilities and enable the interfacing of the RAV module with other entities of the 5G EVE platform, such as the Data Collection Manager (DCM), Experiment Lifecycle Manager (ELM) and so on, it has since moved to a standalone module. Another benefit of its, now, standalone status is the ability to instantiate multiple instances of the module.

The new functionalities implemented in this new version of the RAV module are:

- An API to configure and control its' operation.
- Modular analysis and validation functionalities that allow for Use case / Test case specific analysis and validation algorithms.
- Reporting functionality that ties in with the reporting components of other EVE platform elements in order to generate a complete experiment report.

The module's operation is shown in Figure 23.

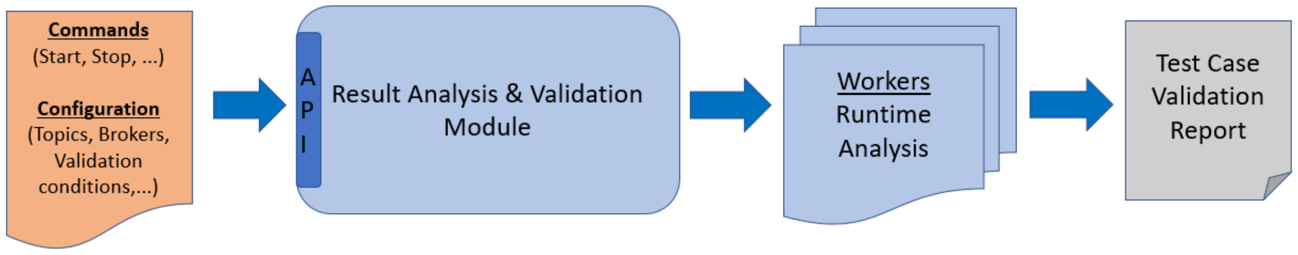

Commands
(Start, Stop, ...)
Configuration
(Topics, Brokers, Validation conditions,...)
API
Result Analysis & Validation Module
Workers
Runtime Analysis
Test Case Validation Report

Figure 23: Analysis and Validation module operation

The operation of the RAV module, and especially the Analysis functionality will create the base upon which the Performance Diagnosis mechanism will be created later on in the project. Collecting the metrics, post-processing and organizing them to proper structures, thus generating a pool of actionable data and relations will feed the diagnosis procedure.

4.2.1 Analysis & Validation

In the current state of the module the analysis of the metrics, normalization and organizing in useful data structures happens while the Test case is executed. After the Test case execution has finished, the validation functionality is triggered. The previous agreement is that each Test Case is used to validate a single KPI. Having the validation functionality take place after the execution has concluded means that by having the data for other related metrics available as well, it is possible to reach meaningful conclusions about more complex KPIs, possibly introduced by current or future Use cases in the next drops of the EVE platform.

The operation stages of the Analysis and the Validation functionalities are shown in Figure 24.

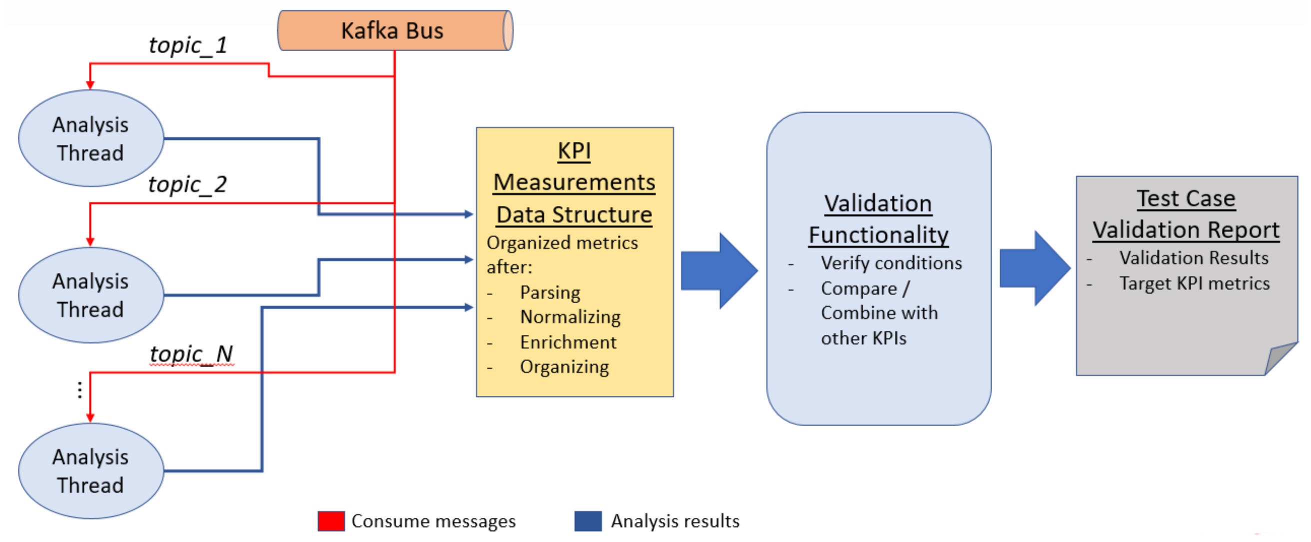


Figure 24: Analysis and Validation functionalities workflow

So far, most of the KPIs that need to be validated require simple calculations and comparisons. As such, the default operation of the Validation functionality is adequate. The modularity of the design, though, allows for the importing of an external validation methodology by the corresponding vertical that requires it. It can also be configured to use “black-box” validation mechanisms as long as those scripts or binaries are provided to the platform.

4.2.2 Reports

The reporting of the experiment results happens in various stages of the experiment definition and execution process. The final report generated for the vertical consists of 4 separate reports created by 4 entities of the EVE Platform. These are:

- Test Case Validation report: generated by the RAV module. It pertains to the targeted KPI that the Vertical wants to validate. It contains information regarding the behaviour of the KPI throughout the test run as well as the final result of the validation process.
- Test Case report: generated by the EEM. It includes the reports produced by the Robot Framework (RF) and it is an operational report more focused on the different stages of the test execution process and less about the KPIs.
- Experiment report: generated by the ELM. This report contains all the information regarding the requested parameters, technologies used, use case details and other details pertaining to the experiment at a higher level.
- Scheduling report: generated by the Portal. The information produced by the Portal, as the name suggests, are all related to scheduling the experiment execution like the time slot, the one or more selected sites and so on.

The complete reporting workflow can be seen in Figure 25 below.

The generated reports, especially the Test Case Validation reports and the Test Case reports, except being useful to the experimenter will also provide the necessary input for the Performance Diagnosis mechanism that will be implemented in the next drops of the platform. Having the detailed reports of multiple test case runs available beforehand can speed up the process and aid in increasing the accuracy of the performance diagnosis as well as the impact of the performance improvement suggestions.

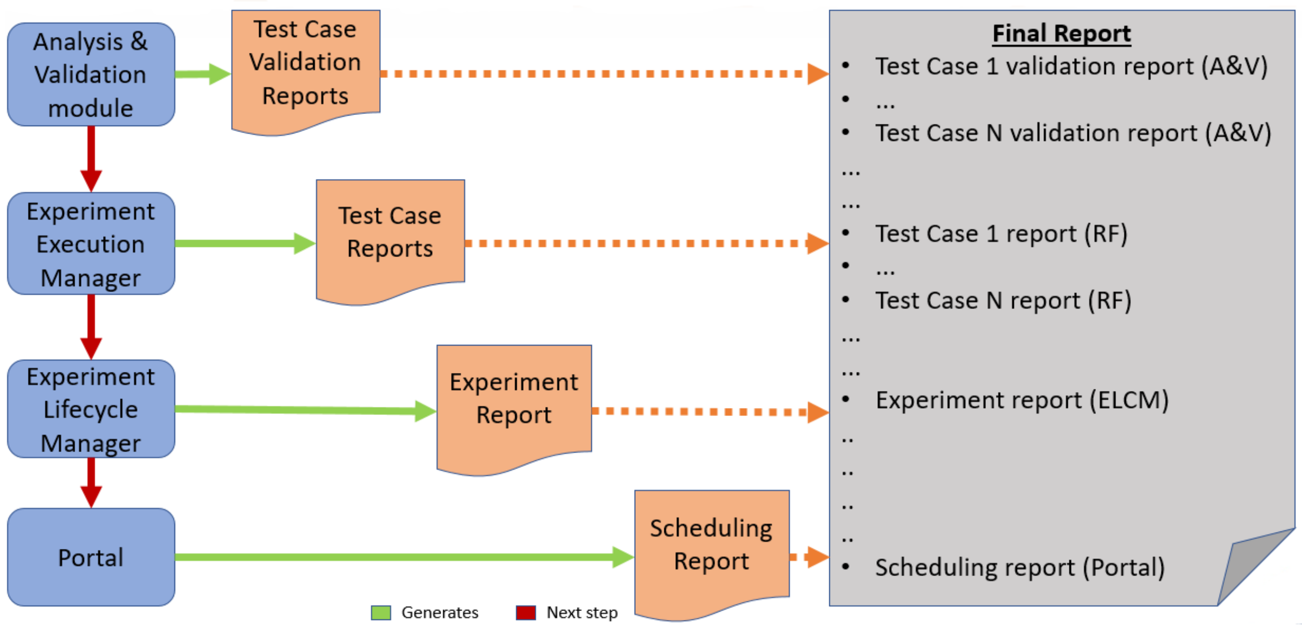

Analysis & Validation module
Experiment Execution Manager
Experiment Lifecycle Manager
Portal
Test Case Validation Reports
Test Case Reports
Experiment Report
Scheduling Report
Final Report
Test Case 1 validation report (A&V)
...
Test Case N validation report (A&V)
Test Case 1 report (RF)
...
Test Case N report (RF)
Experiment report (ELCM)
Scheduling report (Portal)
🟩 Generates
🟥 Next step

Figure 25: Reporting

5 Update of testing and validation results

This section includes the main improvements achieved in the last months compared with the initial results of some use cases already presented in section 3.4 of D5.1.

5.1 Application in the ASTI use case

Results of the most basic tests of the ASTI use case was already presented in D5.1, where an AGV was controlled by a master PLC (Programmable Logic Controller) running as a virtual machine hosted by a server attached to a MEC (Multi-Access Edge Computing). For this deliverable the testbed has been extended to include a new component between the AGV and the master PLC to generate synthetic impairments, like delays and packet losses. This testbed is shown in Figure 26, where this component is identified as “tc”, because it is implemented using the traffic control (tc), which is a tool available in Linux, and more specifically using the NetEm extension. This figure includes a box called Robot Framework, because it is planned to manage the “tc” box using the Robot Framework tool, although at this stage the “tc” is managed manually.

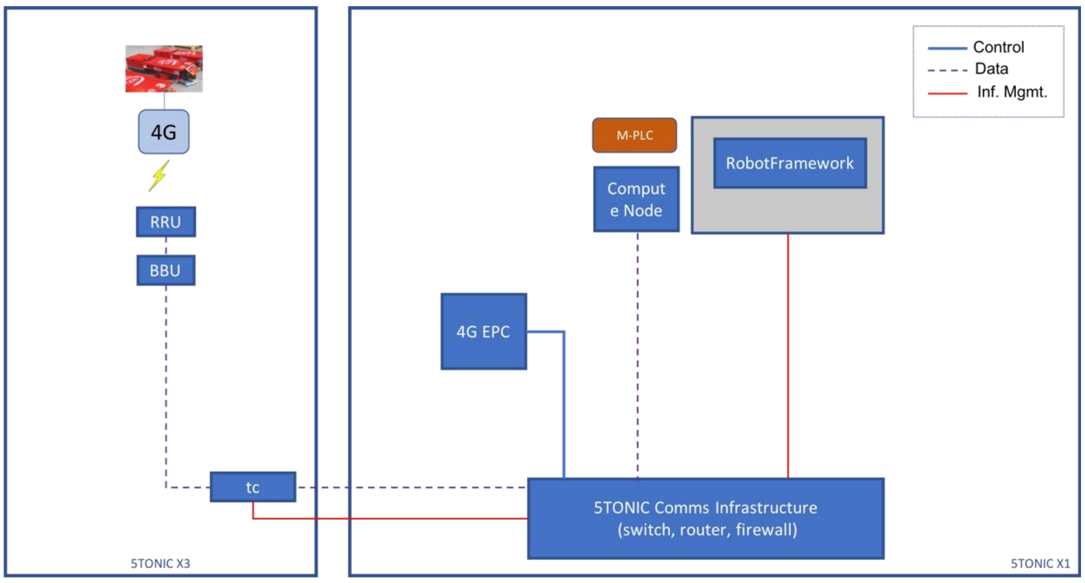


Figure 26: ASTI use case testbed

Two types of tests were performed at this stage: delay and packet losses impairments. The main goal in all these tests is to check how the *guided error* detected by the master PLC is affected by these changes configured at the *traffic control*. The *guided error* is a value that is correlated with the degree of deviation of the AGV compared with the optimal path. Of course, this value is affected by delays and packet losses, and the goal of these tests is to get some maximum values for these impairments.

Regarding delays, Figure 27 shows the *guided error* of the AGV in one lap to the circuit for different values of delays, starting from 0 and then adding extra round-trip delays of 100 ms. Because at 300 ms the AGV was not properly controlled, exiting the path, it was decided to include intermediate delays. Results show that an extra 200 ms round-trip delay is acceptable for controlling the AGV, but above this value the AGV cannot be controlled effectively (with 225 ms extra delay, the *guided error* is 0 at some points because the AGV losses its path, so a local alarm stops it completely).

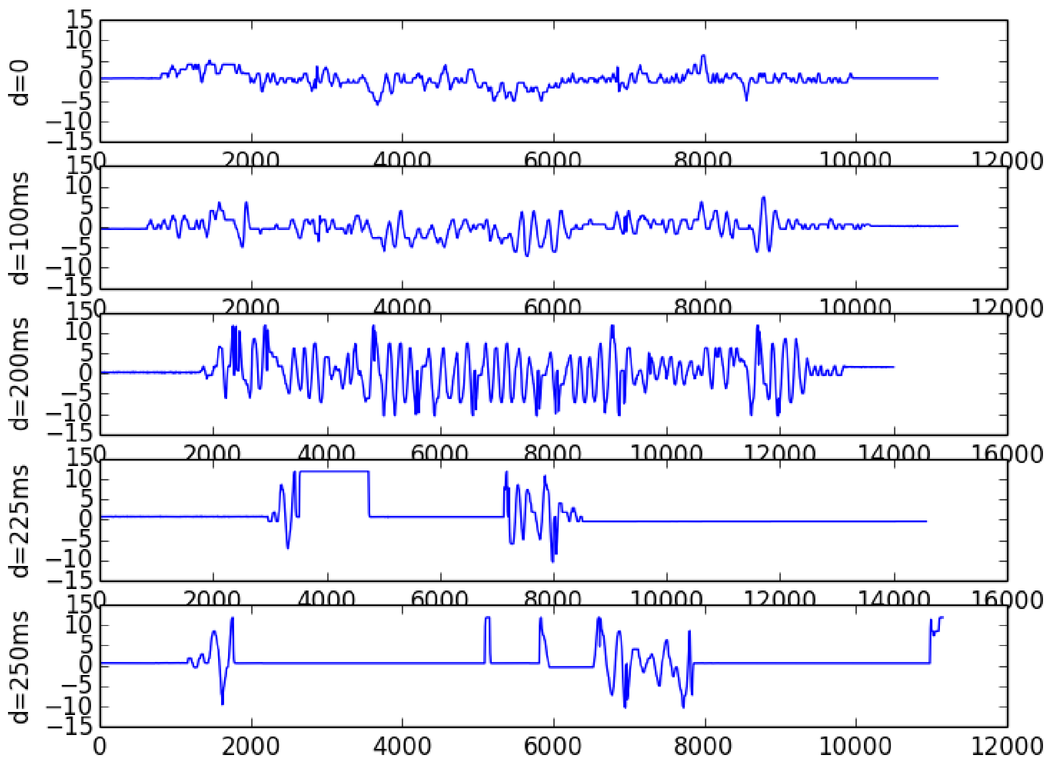


Figure 27: Guided error of the AGV for different extra delays

Figure 28 shows the *guided error* for different synthetic percentage of packet losses, both in the uplink (sensors information sent by the AGV) and downlink (control information sent by the master PLC to the AGV). These losses are identified in the figure using the acronyms *lu* (losses at the uplink) and *ld (*losses at the downlink). As it can be seen in the figure, the AGV does not lose the path during the lap, even with 30% of packet losses. It is important to highlight that these packet losses are computed per packet, so it does not include bursts, which could be also configured at the *traffic control* tool, although for these experiments this parameter was not included. In future experiments we plan to include more tests, controlling all of them using Robot Framework.

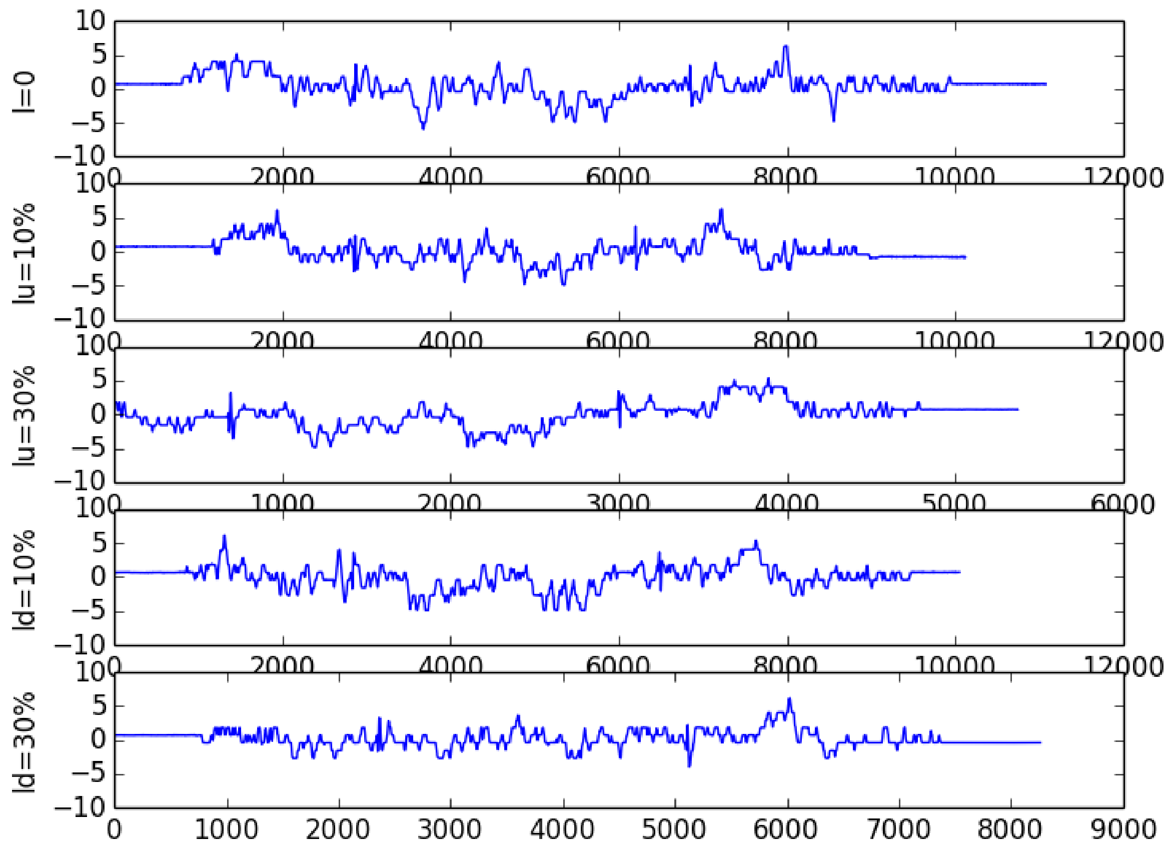


Figure 28: Guided error of the AGV for different packet losses

5.2 Application in the Segittur use case

As commented in section 3.2.1.2, where the updated responses to the high-level test plan templates are outlined, the main goal of the Smart Tourism use case (in which Segittur is participating) is to understand how 5G can improve the capabilities or performance of applications destined to transform the experience of participants in events, conventions, etc. In that sense, part of the testing activities are focused on characterizing what the results are on top of 4G networks, so later on these can be compared with those over 5G. This section reports on the tests executed so far.

The first test case in which some results are available is the one trying to determine the impact of latency on VR video content (Test Case 1 in Annex I.4). The main KPI of this test case is called “viewport changes”, or “Field of View (FoV) changes”. The guidelines determining “correct behaviour” for this KPI look like the dotted representation in Figure 29. For example, it can be determined that 50% of the viewports should have changed within 200ms of the video buffer, while 99% should have changed within 500ms.

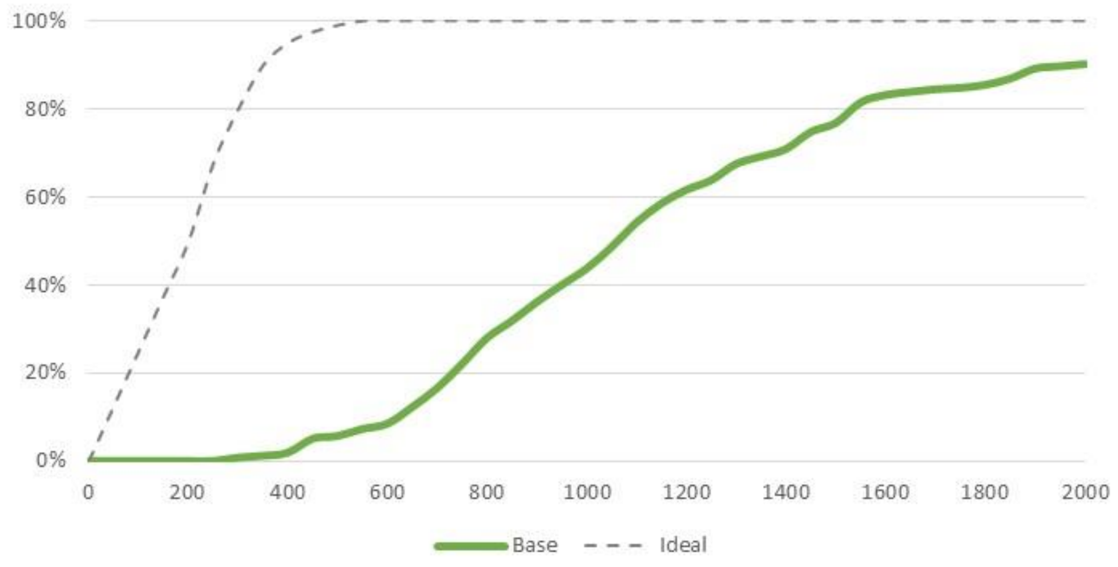


Figure 29: Viewport change speed expected distribution and obtained base results

Another KPI in this test case is the Time to Right Frame (TtRF), governed by distributions like the dotted line of Figure 30. In this case, the TtRF should be 500ms or less during all the playback or should be 200ms or less for 95% of the playback.

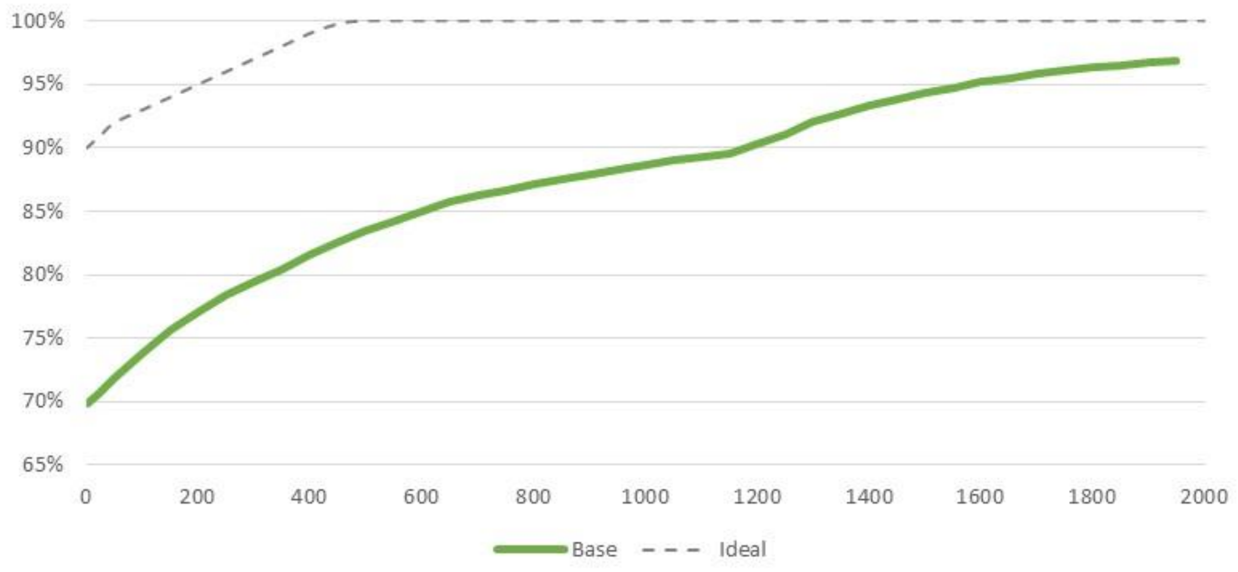


Figure 30: TtRF expected distribution and obtained base results

Results available over today's technologies (in other words, results in the production environment today) still differ greatly from the expected distributions, as can be observed in the solid line of the previous figures. The expectation in this use case is that 4G may even worsen the base case (latency may not be able to cope with the short adaptation times required), but that 5G could provide significant improvements, even close to the ideal scenario.

The second test case also considers the effect of latency, but in this scenario, the KPI is the number of camera changes during playback. The expected result is that 5G outperforms 4G at least by a 50%, given the delta in the expected latencies. So far, results for 4G have been already obtained, as shown in figures from Figure 31 to Figure 33. As stated in the test plan in Annex I.4, all tests are executed over three delay scenarios, depending on how much delay is incorporated to the experiment (none, 50ms or 150ms).

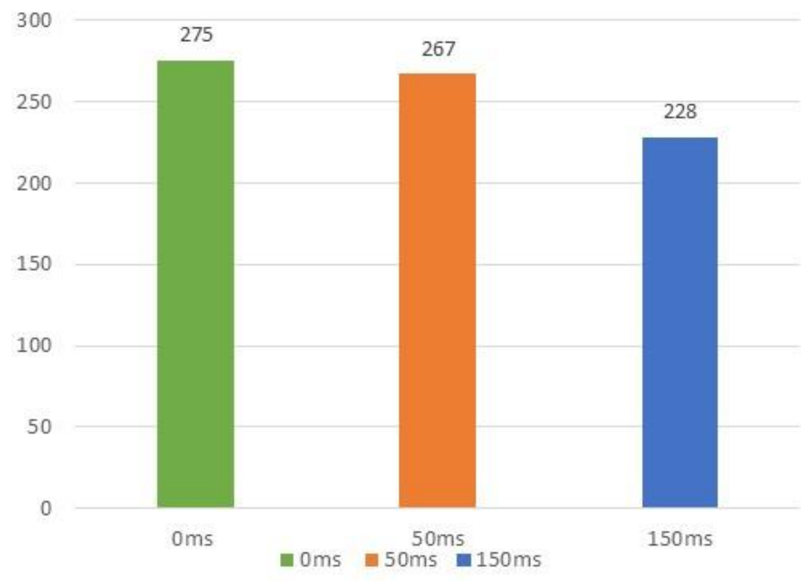


Figure 31: Number of camera changes during playback – results over 4G

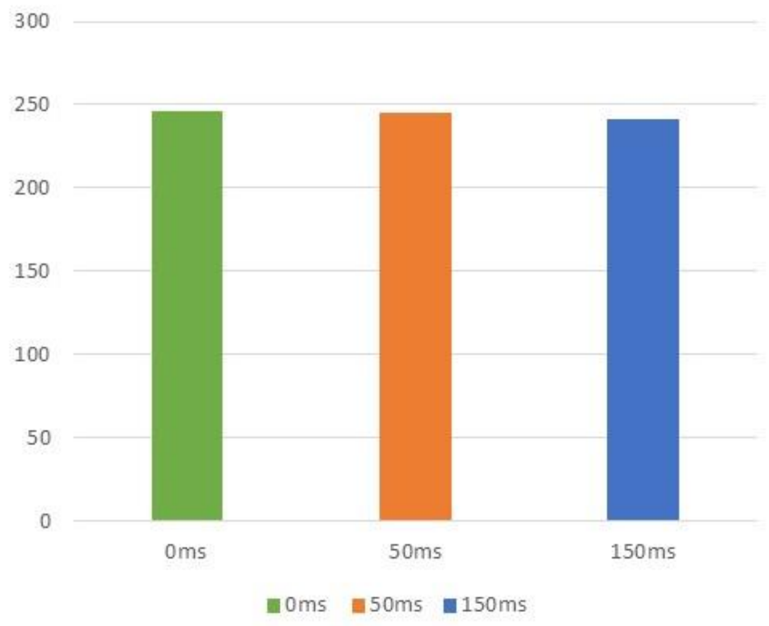


Figure 32: Average TtRF in ms – results over 4G

It has to be noted in the next figure (Figure 33) that results for values over 500ms of the video buffer need to be revisited to ensure their correctness; these are preliminary results that will be reviewed (and updated if necessary in future documents).

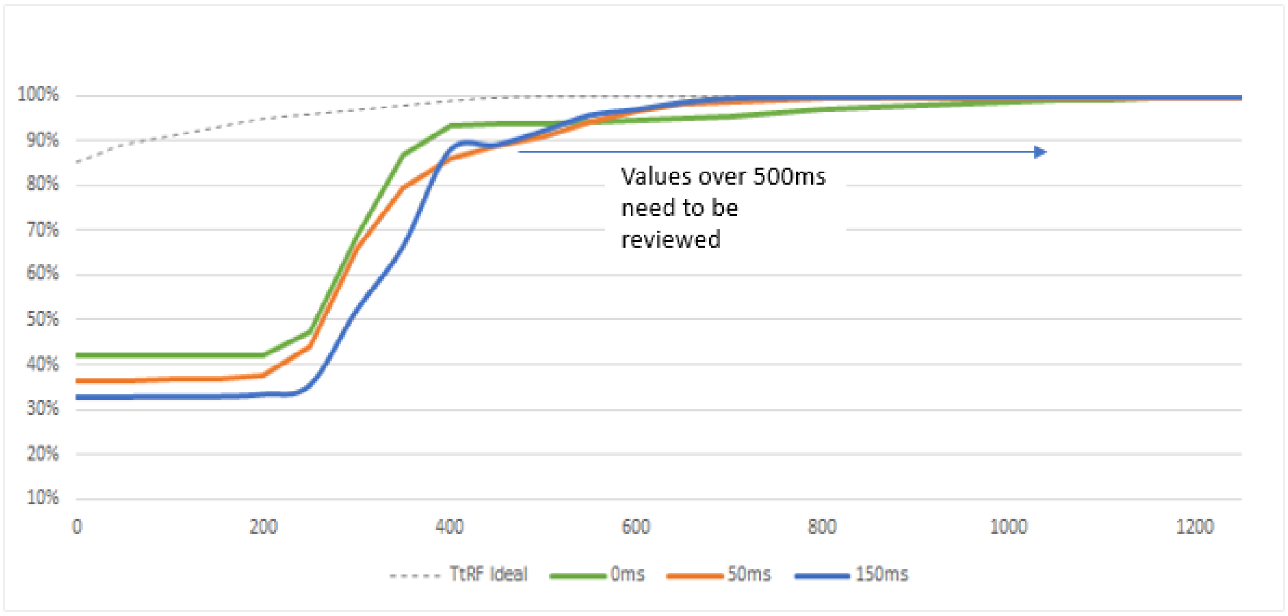


Figure 33: TtRF distribution – results over 4G

The last of the results already obtained has to do with Test Case 3, in which the downlink bit rate is evaluated. The video source transmits an average bitrate of 50Mbps, and it can be seen that this bandwidth is correctly received at the destination when using a 4G network, independently of the introduced additional delay (again, zero, 50 or 150ms.).

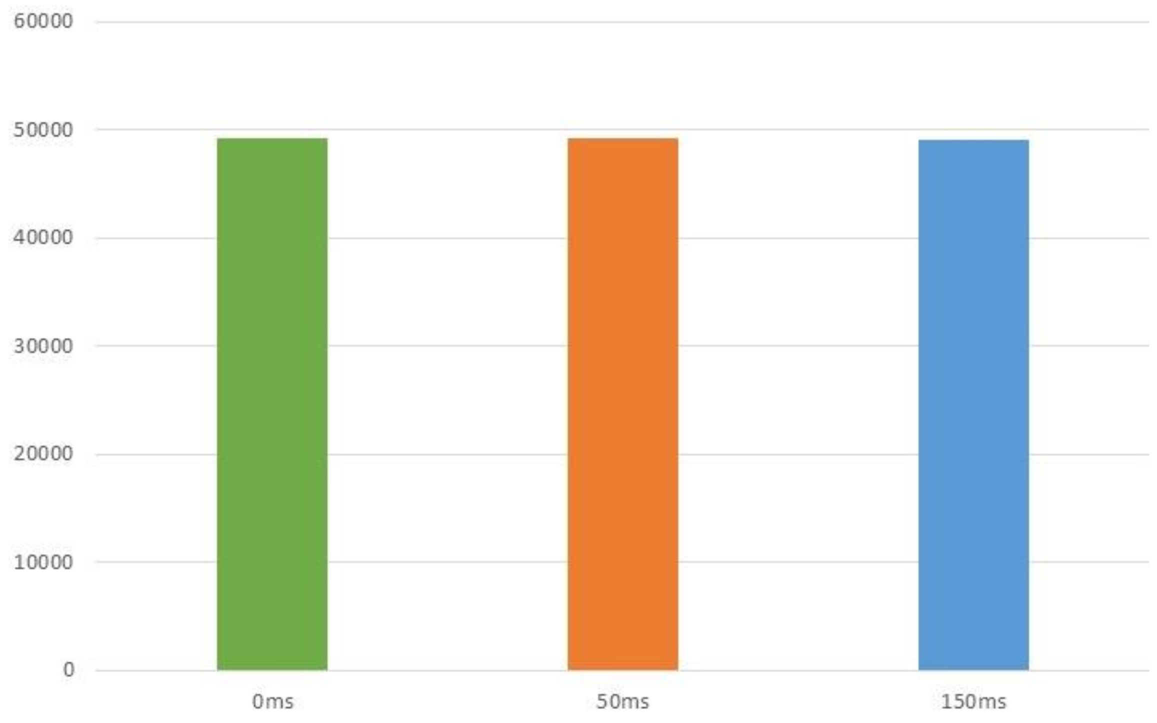


Figure 34: Average bitrate – results over 4G

5.3 Application in the WINGS Utilities use case

Some initial results of the Utilities use case were also presented in D5.1. The experiment revolved around a basic operation of the power grid restoration service. The setup used in that initial experiment included a remote server handling the node monitoring and rerouting decision making, smart nodes consisting of power consumers and power generators and telecom infrastructure in the form of an eNodeB and an EPC in order to connect the smart nodes with the remote server. The testbed that generated the new results, shown in Figure 35, is the same as the previous one expanded with an impairment node. The Robot Framework handles the preparation and execution of the necessary functionalities on all the nodes. The impairment node is responsible for injecting network impairments to the link between the smart nodes and the remote server, in this way simulating real network conditions. The experiment is executed using the latest version of the testing tool, expanded since the first disjoint version with additional analysis, validation capabilities.

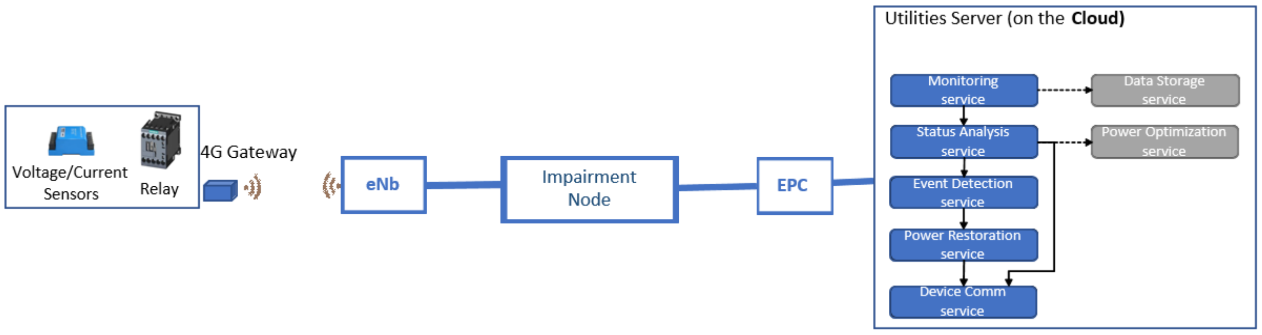


Figure 35: Testbed for the latest Utilities results

The test executed focuses around two network impairments, delay and packet loss. The focus of these tests will be the description of the effect these impairments have on one of the Vertical KPIs, the Power Restoration Time (PRT). The PRT consists of four parts:

- Transmit time
- Server rerouting decision time
- Receive time
- Reconfiguration time

The transmit and receive times are directly affected by the impairments, in opposition to the server decision time and the reconfiguration time, as these are platform intrinsic and node intrinsic respectively. The vertical has set as a validation condition that the PRT should be below 1 ms. The experiment will show under what conditions this requirement can be fulfilled. First, the impact of the increasing latency can be seen in Figure 36.

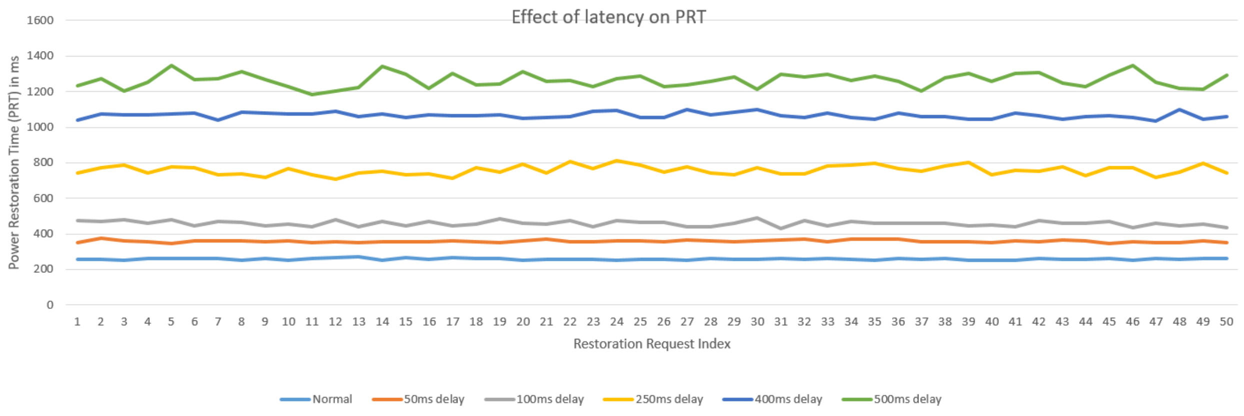


Figure 36: Effect of delay on the Power Restoration Time

As expected, additional delay increases the total PRT linearly. The various steps of the injected delay were 50ms, 100ms, 250ms, 400ms and 500ms. In order for the Vertical's condition of operation, PRT less than 1 second, to be valid, the testing and validation tools demonstrated that the maximum acceptable delay is less than 400ms. More than 400ms delay, and consequently more than 1 second PRT can cause power loss and damages to the nodes. Similarly, the impact of the increasing packet loss can be seen in Figure 37.

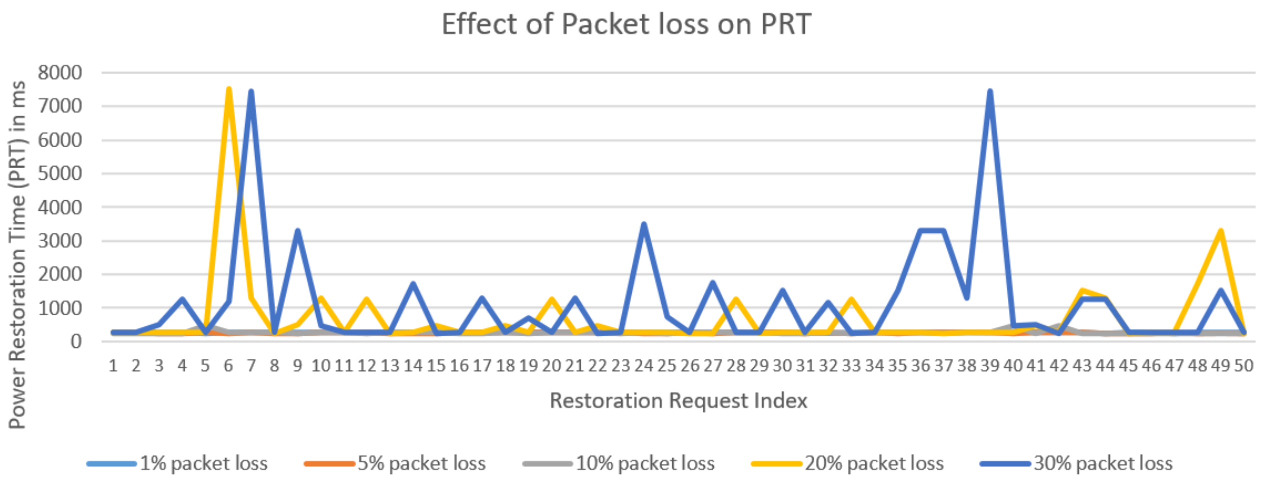


Figure 37: Effect of packet loss on Power Restoration Time

Packet loss up to 10% appears to have a reduced effect on the PRT. When that value reaches 20% and more, the impact is significantly more disruptive to the operation of the node since multiple lost packets disrupt the restoration process.

These two tests, while useful, show the impact of one network impairment on a selected Vertical KPI. The next step is to expand the functionalities of the testing tools and the impairment node integration, so as to be able to enable multiple impairment injection from the testing framework. In this way, more complex environments can be simulated and more complex tests, as well as use cases, can be supported.

6 Conclusions and next steps

When delivering an architecture definition document, only one should be the main conclusion, and only one should be the next move. As for the conclusion: we have effectively designed a model-based testing (and validation) architecture. As for the next steps: we are now in a position to develop such architecture.

Having stated this, which although obvious must still be highlighted, a good exercise is to evaluate how much the designed architecture complies with the initial requirements.

The main objective, as stated in section 1.3, was *the design of an overall framework enabling and supporting automated model-based testing and validation of 5G technologies, components and services*. Several requirements can be extracted from here, which will be listed later on. Also, with regards to the Testing-as-a-Service environment that the project is offering, its *ultimate goal is to allow for the automated execution and evaluation of test cases, based on high-level test plan descriptions of the intended experiments, and according to model-based testing principals*. These two definitions, plus the concepts of multi-site testing and monitoring, make up an already interesting list of requirements against which the designed architecture can be confronted:

1. Automation
2. Model-based
3. Testing capability
4. Validation capability
5. Monitoring capability
6. Benchmarking testing
7. Vertical's applications testing
8. High-level test plans
9. Multi-site testing

The following table reflects our evaluation on the designed architecture. Scores are given from 0 to 3, with the following meaning.

- 0: requirement not met
- 1: requirement partially met
- 2: requirement met, but there is room for some improvement
- 3: requirement perfectly met

Table 14: 5G EVE testing and validation framework support of requirements

Requirement	Score	Justification
Automation	2	First versions of the framework will not be fully automated, but it can be seen from the Deliverable's workflows that all operations during the Experiment Execution phase (which also includes the monitoring and the validation) are planned to be automated. "3" is not given because there are still many manual operations in the Experiment Design phase.
Model-based	3	All tests are based on blueprints that can be reused (or even combined) at the experimenter's will, trying to simplify the heterogeneous Verticals' scenarios. Strictly speaking, with this approach this is not a WP5 feature, but WP5 has adapted to blueprints, and will also produce test scripts (for different Test Case blueprints) applying the same philosophy.
Testing capability	3	The Experiment Execution Manager (EEM) drives all the testing capabilities and procedures, interfacing with WP3 modules (Runtime Configurator) to interact with local components at sites.

Validation capability	2	The Results Analysis and Validation (RAV) module drives all the validation capabilities and procedures, obtaining from the Data Collection Manager the required metrics, and receiving the trigger for validation from the EEM. “3” is not given because the validation conditions are still somehow simple.
Monitoring capability	3	WP5 has leveraged the powerful monitoring capabilities developed within WP3 and WP4, and has just adapted to those schemes.
Benchmarking testing	1	Tests with the objective of evaluating 5G technologies, architectures or components are perfectly possible in 5G EVE. “3” is not given because we are not yet in condition to validate all the required Network KPIs (see section 2.1.2).
Vertical’s applications testing	3	Tests with the objective of evaluating Verticals’ applications and services are perfectly possible in 5G EVE. Validation and visualization of Verticals’ KPI are even possible if they can inject their own metrics in the same data broker that is used for infrastructure monitoring.
High-level test plans	3	Elaboration of Test Cases based on high-level test plans provided by the incoming experimenters is a reality in 5G EVE, as demonstrated by the templates responses that participating Verticals have provided. These tools should be key to understand Verticals coming from ICT-19 projects.
Multi-site testing	3	Again, strictly speaking, the support of this requirement has not been fully solved in WP5, but in WP4. The Multi-site NSO is capable of deploying multi-site experiments; the Data Collection Manager can receive metrics from components no matter their location; and the Runtime Configurator can interact with all deployed elements. WP5 and WP4 have jointly defined the required interfaces so that, for WP5, there is no difference between multi-site or single-site testing.

As seen, we are getting very good (given by ourselves) grades at this stage of the project, where there is still a lot of time to incorporate to the framework more capabilities concerning the evaluation of Netwok KPIs, (which is where we also need to push in the next months). It also has to be noted, and the list of references is a good proof of that, that most of the work done in this WP has been started from scratch, given the very few previous initiatives of this type.

References

[1] 5G EVE Deliverable D5.1 “Disjoint testing and validation tools” – ICT-17-2018 5G EVE Project – 03/05/2019
[2] 5G EVE Deliverable D1.3 “5G EVE end-to-end facility reference architecture for Vertical industries and core applications” – ICT-17-2018 5G EVE Project – To be published in Dec. 2019
[3] 5G EVE Deliverable D1.4 “KPI collection framework” – ICT-17-2018 5G EVE Project – 28/06/2019
[4] Report ITU-R M.2410-0 (11/2017)
[5] 5G EVE Deliverable D4.1 “Experimentation Tools and VNF Repository” – ICT-17-2018 5G EVE Project – 31/10/2019
[6] 5G EVE Deliverable D3.2 “Interworking Reference Model” – ICT-17-2018 5G EVE Project – 28/06/2019
[7] 5G EVE Deliverable D3.3 “First implementation of the interworking reference model” – ICT-17-2018 5G EVE Project – 31/10/2019

Annex I: High-level test plan templates – available responses

Annex I.1: Experiment template for Utilities use case (Greece)

The following has been extracted from the template response for the Utilities use case in the Greek site.

Experiment Description:

The experiment will test the performance of the Utility Platform in a 5G network infrastructure in two different deployments. In first deployment the Utility Services are running on the Cloud, while on second deployment the Utility Services are running on the MEC. The experiment will examine the performance of the Platform/Infrastructure in case of power failures.

Expected Capabilities:

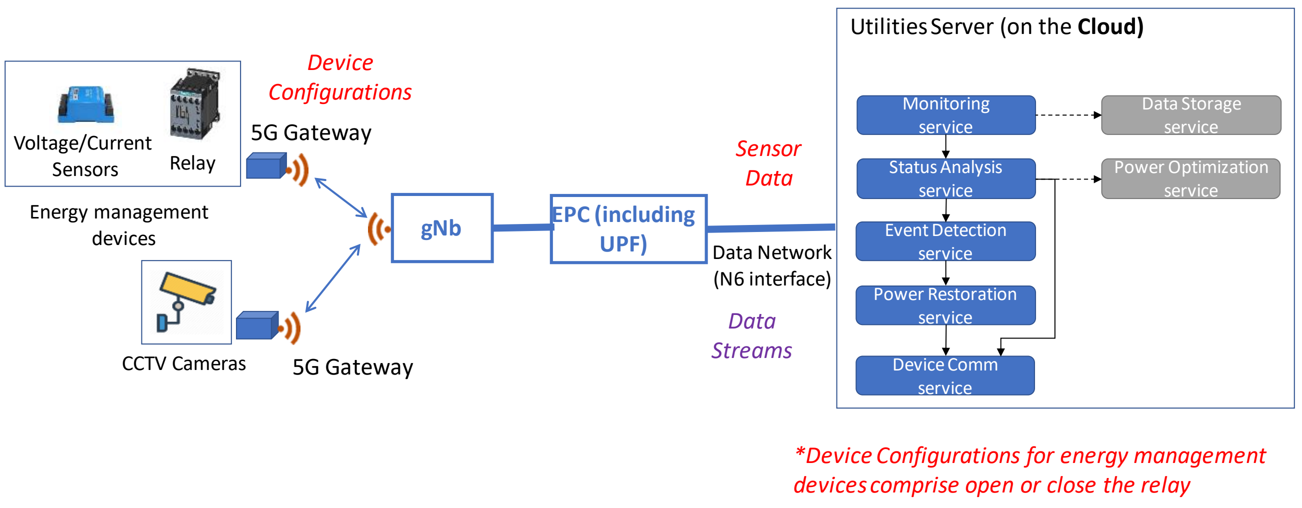
A high performance and mobile network connecting the Voltage/Current Sensors, the CCTV Cameras, the Replay Nodes, the 5G Gateways with the Utilities Servers. A very fast power recovery is expected in cases of power failures with low power restoration time thanks to the 5G infrastructure and MEC capabilities.

Table 15: Utilities UC experiment components

Component Name	Description	Deployment requirements
Voltage/Current Sensor	*Sensor measuring the values of voltage and current*	*Deployed in the distributed power grid infrastructure*
CCTV Camera	*Camera located in selected places of the power grid infrastructure*	*Deployed in the distributed power grid infrastructure*
Relay Node	*Node in the power grid infrastructure capable of relaying current to other nodes of the power grid infrastructure*	*Deployed in the distributed power grid infrastructure*
5G Gateway	*Node close to the sensors/cameras responsible to collect data from the sensors/cameras or send data to relay nodes. It communicates with the 5G RAN (acting as 5G terminal).*	*Deployed close to sensors, relay nodes and cameras to provide 5G connectivity*
Utilities ARTEMIS platform	*Cloud platform in which the Utilities Services are deployed*	*Deployed in a centralized server as Virtual Machine or in a distributed manner in the cloud continuum (edge/MEC, cloud)*

Experiment architecture:

Architecture 1: Utility Services on the Cloud (main scenario)



Architecture 2: Utility Services on the MEC (scenario to be investigated)

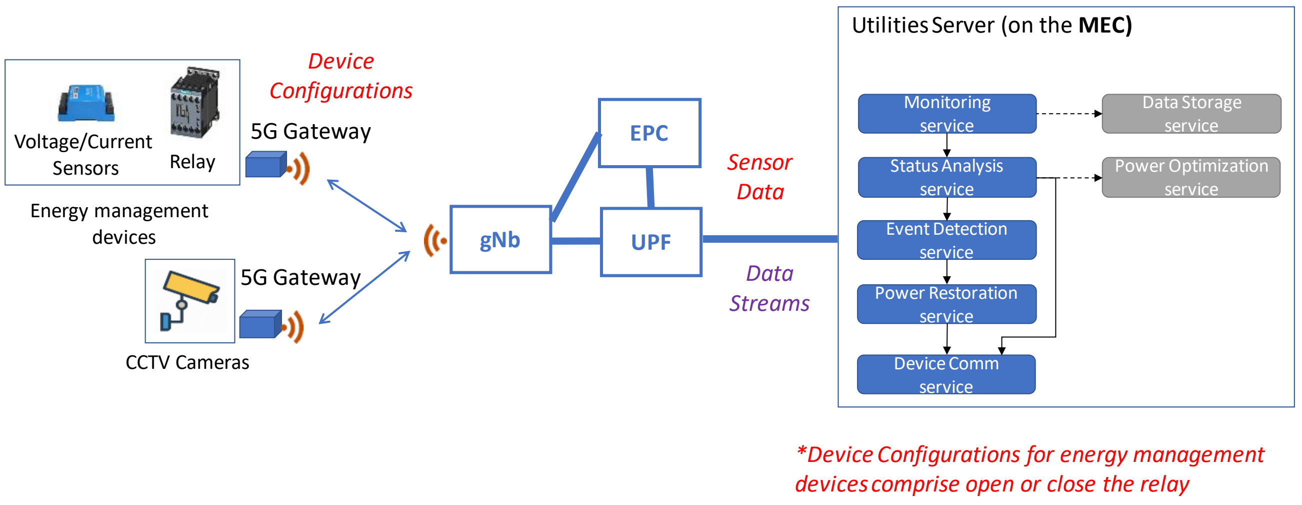


Table 16: Utilities UC meaningful KPIs

5G related KPIs	**Comments**	
RTT Latency	*Target: 20 – 100 msec, especially for the downlink (configurations)*	
Bandwidth	*Target: 15Mbps-50Mbps, especially for the uplink*	
Vertical KPIs	**Is it measurable during experimentation?**	**Can it be mathematically derived from 5G related KPIs?**
Power Restoration Time	*Yes, it is measured in the 5G Gateway node. Target: 1s*	*End-to-end latency plus an offset time of relay node's activation*
Power Restoration Decision Time	*Yes, it is measured in the Power Restoration service.*	*No*

Table 17: Utilities UC test plan

Test Case 1	
Test Name:	*Effect of Number of requests in RTT Latency*
Target KPI:	*RTT Latency*
Measurement Method:	*The RTT Latency will be measured by the 5G Gateway. It is the end-to-end latency between a power failure request and the power restoration response.*
Parameters:	*The number of requests will be set to 1, 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*RTT Latency between < 100 msec*

Test Case 2	
Test Name:	*Effect of Number of requests in Bandwidth*
Target KPI:	*Bandwidth*
Measurement Method:	*The Bandwidth will be measured by the 5G Gateway. It will be measured on the outgress interface for the uplink and on the ingress interface for the downlink.*
Parameters:	*The number of requests will be set to 1, 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Bandwidth (both on downlink and uplink) > 15Mbps*

Test Case 3	
Test Name:	*Effect of Number of requests in Power Restoration Time*
Target KPI:	*Power Restoration Time*
Measurement Method:	*The Power Restoration Time will be measured by the 5G Gateway. It is the end-to-end latency plus an offset time of relay node's activation.*
Parameters:	*The number of requests will be set to 1, 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Power Restoration Time < 1s*

Test Case 4	
Test Name:	*Effect of Number of requests in Power Restoration Decision Time*
Target KPI:	*Power Restoration Decision Time*
Measurement Method:	*The Power Restoration Decision Time will be measured by the Power Restoration service. It is the time between an arrival of a new request at the "Power Restoration Service" and the completion of this request. The time is measured as the time between the "request arrival ID" and "request completed ID" logs.*
Parameters:	*The number of requests will be set to 1, 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Power Restoration Decision Time < [VALUE STILL TO BE DETERMINED]*

Test Case 5	
Test Name:	*Effect of Grid Complexity in RTT Latency*
Target KPI:	*RTT Latency*
Measurement Method:	*The RTT Latency will be measured by the 5G Gateway. It is the end-to-end latency between a power failure request and the power restoration response.*
Parameters:	*The number of power grid nodes generated (emulated) will be set to 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*RTT Latency between < 100 msec*

Test Case 6	
Test Name:	*Effect of Grid Complexity in Bandwidth*
Target KPI:	*Bandwidth*
Measurement Method:	*The Bandwidth will be measured by the 5G Gateway. It will be measured on the outgress interface for the uplink and on the ingress interface for the downlink.*
Parameters:	*The number of power grid nodes generated (emulated) will be set to 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Bandwidth (both on downlink and uplink) > 15Mbps*

Test Case 7	
Test Name:	*Effect of Grid Complexity in Power Restoration Time*
Target KPI:	*Power Restoration Time*
Measurement Method:	*The Power Restoration Time will be measured by the 5G Gateway. It is the end-to-end latency plus an offset time of relay node's activation.*
Parameters:	*The number of power grid nodes generated (emulated) will be set to 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Power Restoration Time < 1s*

Test Case 8	
Test Name:	*Effect of Grid Complexity in Power Restoration Decision Time*
Target KPI:	*Power Restoration Decision Time*
Measurement Method:	*The Power Restoration Decision Time will be measured by the Power Restoration service. It is the time between an arrival of a new request at the "Power Restoration Service" and the completion of this request. The time is measured as the time between the "request arrival ID" and "request completed ID" logs.*
Parameters:	*The number of power grid nodes generated (emulated) will be set to 5, 10, 20, 50, 100, 200, 500, 1000*
Validation Conditions:	*Power Restoration Decision Time < [VALUE STILL TO BE DETERMINED]*

Annex I.2: Experiment template for Industry 4.0 use case (Spain)

The following has been extracted from the template response for the Industry 4.0 use case in the Spanish site. It must be noted that this is just an adapted (not updated) response from the one already included in D5.1, since the latter was provided over an old version of the template.

Experiment Description:

The experiment will test the virtualization of the control algorithms of AGVs. In the current state of the art the AGVs have an on-board PLC in charge of governing the internal control loop, which is collecting the information of the guiding sensors, taking the appropriate control decisions and generating the necessary signals to regulate the speed of the motors. During the experiment, this internal control loop will be relocated out of the AGV thanks to the 5G communication technologies.

Expected Capabilities:

A high performance and reliable mobile network connecting the AGVs to the platform with the virtual machines is required. The most challenging requirements to ensure the deployment of the solution in the future factories are the high reliability and the low latency and jitter.

Table 18: Industry 4.0 UC experiment components

Component Name	Description	Deployment requirements
AGV	*AGV where the slave PLC is running*	*220V to power a charging station* *220V to power a traffic station* *Test zone of 9mx4m*
Slave PLC	*PLC included in the AGV*	*Client device to connect to 5G infrastructure. 24Vdc powered or with some dc/dc converter (for example 24/5 if the client device is 5V powered.* *Static IP*
Master PLC	*Virtual PLC embedded in a Virtual Machine*	*Virtual machine to run Windows 7 (Service Pack 1 or higher) / 8 / 10 (32/64 Bit) + suitable PC hardware for the corresponding Windows platform*

Experiment architecture:

The AGV will have a Slave PLC on board. This PLC will collect the information from the sensors and physical inputs, this will be sent to the Virtual PLC. The Virtual PLC will process all this information, then it will take the appropriate control decisions and it will generate the right signals to control the motors of the AGV. This control signals will be sent by 5G communication to the Slave PLC, who will process them translating to real physical signals to command the motors.

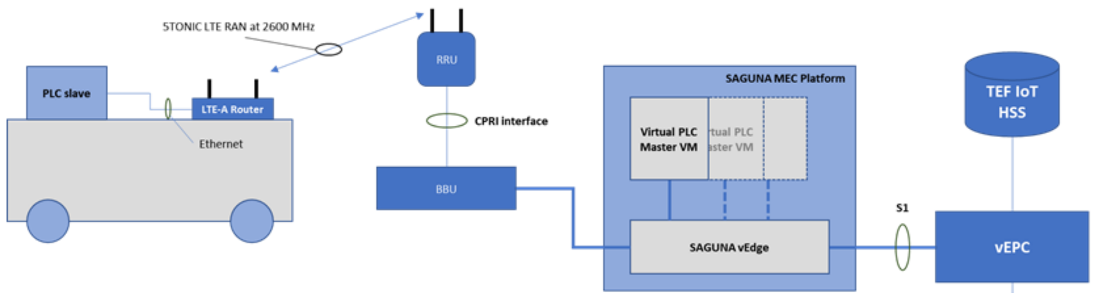


Table 19: Industry 4.0 UC meaningful KPIs

5G related KPIs	**Comments**	
Delay	*The delay will be the variable parameter in each of the test cases*	
Vertical KPIs	**Is it measurable during experimentation?**	**Can it be mathematically derived from 5G related KPIs?**
Navigation Level	*Yes, by the AGV*	*No mathematical formula known*
Consumption	*Yes, by the AGV*	*No mathematical formula known*
Time to lose the guide	*Yes, by the AGV*	*No mathematical formula known*

Table 20: Industry 4.0 UC test plan

Test Case 1	
Test Name:	*Effect of delay in Navigation Level*
Target KPI:	*Navigation Level*
Measurement Method:	*The Navigation Level will be measured by the AGVs. AGVs operate smoothly until a certain Navigation Level is reached*
Parameters:	*No external traffic.* *The delay will be fixed at 10ms, 15ms, 20ms… until the maximum acceptable delay. In each delay value, the AGVs will run 10 cycles.*
Validation Conditions:	*Delay will be considered acceptable until the Navigation Level goes beyond [VALUE STILL TO BE DETERMINED]*

Test Case 2	
Test Name:	*Maximum supported delay*
Target KPI:	*Time to lose the guide*
Measurement Method:	*When the delay is too high, the guides of the AGVs are lost, and they stop. A "guide lost" log is also sent.*
Parameters:	*No external traffic.* *The delay is increased until one AGV loses the guide*
Validation Conditions:	*Reception of "guide lost" log*

Test Case 3	
Test Name:	*Effect of delay in Consumption*
Target KPI:	*Consumption*
Measurement Method:	*The consumption will be measured by the AGVs. AGVs consumption is increased with the delay*
Parameters:	*No external traffic.* *The delay will be fixed at 10ms, 15ms, 20ms… until the maximum acceptable delay. In each delay value, the AGVs will run 10 cycles.*
Validation Conditions:	*Consumption will be considered acceptable until it goes beyond [VALUE STILL TO BE DETERMINED]*

Annex I.3: Experiment template for Media & Entertainment use case – Virtual Visit over 5G (France)

The following has been extracted from the template response for the Media & Entertainment use case in the French site.

Experiment Description:

The experiment will test the virtualization of some 5G components and especially the video 360° server. The main video use-cases typically deport the video server in the cloud. The main question is how far it can be deported especially when it corresponds to virtual Reality (VR) contents. This is the main expected objective of this experiment.

Expected Capabilities:

To prevent nausea when showing high quality VR video format, it is mandatory that the E2E transmission provides low latency and jitter, the throughput coming in background for the user's point of view (possibility to adapt and degrade a little bit the video format). Also, the buffer state at the receiver has a great QoE impact on the video transmission. So, the most challenging requirement to ensure the deployment of the solution and expect the capabilities is the low latency. However, it is crucial to know exactly where (at what layer) the latency value must be measured.

Table 21: Media & Entertainment UC experiment components

Component Name	Description	Deployment requirements
HMD	*Head Mounted Display: HTC Vive or other type.*	*For autonomous HMD, USB3 connection to the 5G modem* *For no autonomous HMD, WiFi connection to the video PC processing, the latter connected to 5G modem by USB port*
HTTP local 360° video server	*Video server for VR transmission. It will be hosted in VM.*	*Based on linux Ubuntu 18.04 with Nginx (HTTP server), Nodejs A-Frame (three.js, dash.js).* *220V power supply*
Windows 10 server	*Server used in no-autonomous HMD for video processing*	*220V power supply if no-autonomous HMD. USB3 connection from the 5G modem and to the HMD.*
Router/server	*Server for virtual router and analytics. Used to extract flows and measure KPIs*	*Based on linux Ubuntu 18.04. Router (DHCP server). Bitrate and latency graph tools and wireshark …* *220V power supply*

Experiment architecture:

The experiment architecture is depicted below. As illustrated, the video server is put behind a router and can be deported in the cloud or locally. The video server is embedded in a server that can be hosted in a VM (Linux 18.04). The video stream passed through the 5G network (based on EPC OAI) and the OAI RAN eNodeB that performs the BBU processing before sampling via the software defined radio (SDR) equipment (USRP). Then, the transmitted signal is received by the 5G modem (5G COTS equipment). When using no autonomous HMD, the 5G modem is connected via USB3 to a PC that makes the video processing and then transmits the video stream via USB3 to the HMD for the video scene restitution. In case of autonomous HMD, HMD is connected in WiFi to the video server.

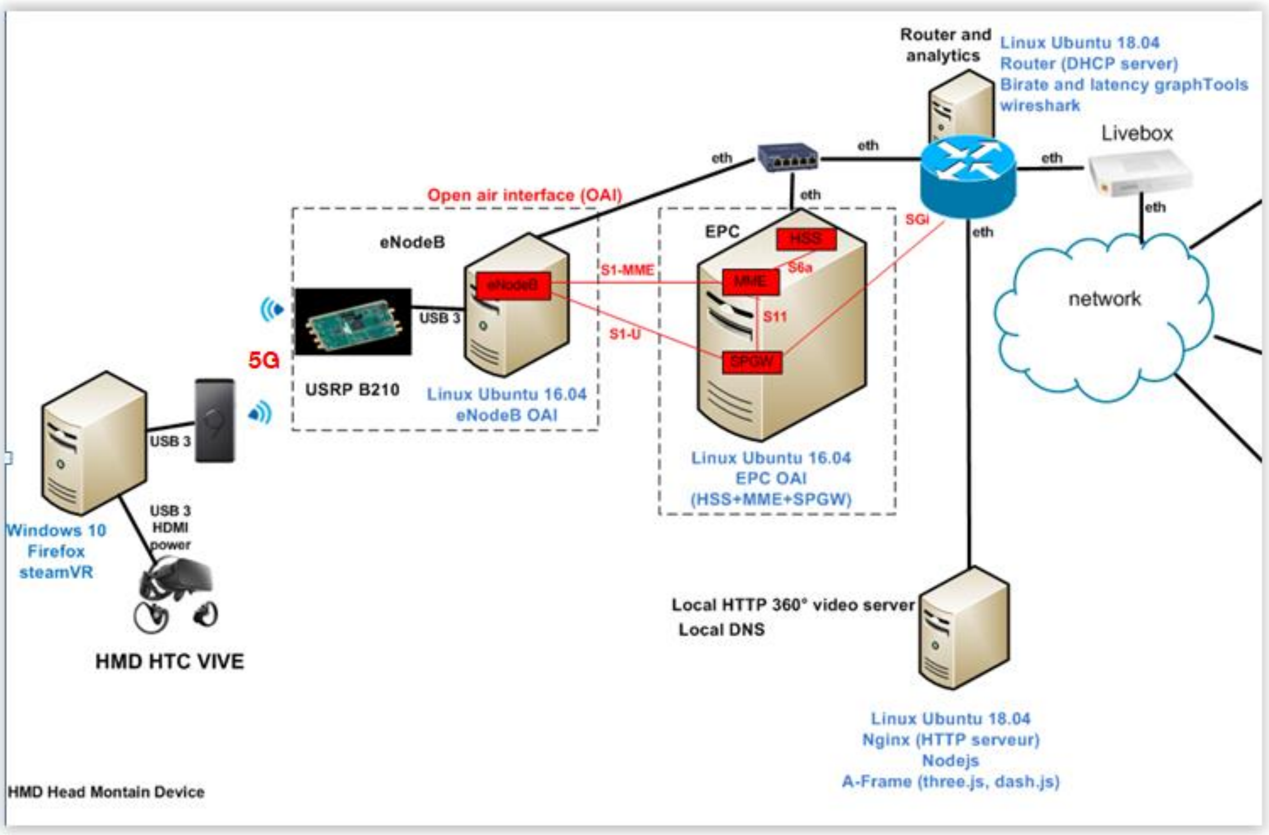


Table 22: Media & Entertainment UC meaningful KPIs

5G related KPIs	**Comments**	
Delay and throughput	*The delay and throughput will be the variable parameters in each of the test cases*	
Vertical KPIs	**Is it measurable during experimentation?**	**Can it be mathematically derived from 5G related KPIs?**
Nausea Level	*Yes, by the user carrying the HMD*	*No mathematical formula known*
Video stream loss: buffer state	*Yes, by probe implemented at the HMD video player API*	*No mathematical formula known*

Table 23: Media & Entertainment UC test plan

Test Case 1	
Test Name:	*Effect of delay in Nausea Level*
Target KPI:	*Max time of supporting the video scene (without nausea)*
Measurement Method:	*The Nausea Level will be measured by the user (felt could be different for each user). Video stream operates smoothly until a certain Nausea Level is reached*
Parameters:	*Unique video traffic (identical scene) whatever the video server deport.* *The delay will be measured and fixed at different values (to be defined regarding the levels where it is measured) until the maximum acceptable delay. In each delay value, at fixed throughput, the measurement will be made for different users.*
Validation Conditions:	*Delay value will be considered acceptable until the Nausea Level goes below [VALUE STILL TO BE DETERMINED]*

Test Case 2	
Test Name:	*Effect of throughput in Nausea Level*
Target KPI:	*Min throughput for supporting VR video 360° (without nausea)*
Measurement Method:	*The Nausea Level will be measured by the user (felt could be different for each user). Video stream operates smoothly until a certain Nausea Level is reached*
Parameters:	*Unique video traffic (identical scene) whatever the video server deport.* *The throughput will be measured and fixed at different values (to be defined regarding the levels where it is measured) until the maximum acceptable throughput. In each throughput value, at fixed delay measured in test case1, the measurement will be made for different users.*
Validation Conditions:	*Throughput value will be considered acceptable until the Nausea Level goes below [VALUE STILL TO BE DETERMINED]*

Test Case 3	
Test Name:	*Effect of delay in buffer state: QoE impact*
Target KPI:	*Max time of supporting the video scene: saccade of images or black screen (video loss)*
Measurement Method:	*The QoE Level (buffer occupancy) will be measured by the users acceptance about video quality (felt could be different for each user). Video stream operates smoothly until a certain acceptance Level is reached*
Parameters:	*Unique video traffic (identical scene) whatever the video server deport.* *The delay will be measured and fixed at different values (to be defined regarding the levels where it is measured) until the maximum acceptable delay. In each delay value, at fixed throughput, the buffer measurement will be made and the QoE evaluated for different users.*
Validation Conditions:	*Delay value will be considered acceptable until the buffer occupancy Level goes above [VALUE STILL TO BE DETERMINED]*

Test Case 4	
Test Name:	*Effect of throughput in buffer state: QoE impact*
Target KPI:	*Minimum throughput of supporting the video scene: saccade of images or black screen (video loss)*
Measurement Method:	*The QoE Level (buffer occupancy) will be measured by the users acceptance about video quality (felt could be different for each user). Video stream operates smoothly until a certain acceptance Level is reached*
Parameters:	*Unique video traffic (identical scene) whatever the video server deport.* *The throughput will be measured and fixed at different values (to be defined regarding the levels where it is measured) until the minimum acceptable throughput. For each throughput value, at fixed delay, the buffer measurement will be made and the QoE evaluated for different users.*
Validation Conditions:	*Throughput value will be considered acceptable until the buffer occupancy Level goes above [VALUE STILL TO BE DETERMINED]*

Annex I.4: Experiment template for Smart Tourism use case (Spain)

The following has been extracted from the template response for the Smart Tourism use case in the Spanish site.

Experiment Description:

The experiment aims at evaluation what benefits 5G brings to applications that take advantage of VR technology and real time 360 degree video to enhance the experience of participants in events or conventions. These applications will be run on top of a 4G network first, and then on top of a 5G network to compare the performance.

Expected Capabilities:

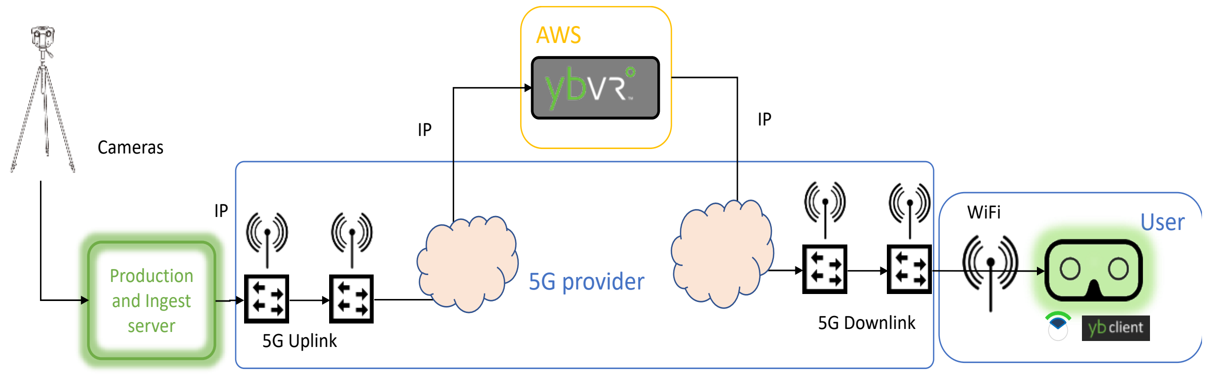
A test environment where tests can be executed over 4G or 5G environments as required. High bandwidth in the 5G solution is a must, as is a high-speed Internet connection for transmission of video flows (4K/8K) from AWS. It must also be possible to emulate additional delay (to simulate congestion) in the network.

Table 24: Smart Tourism UC experiment components

Component Name	Description	Deployment requirements
YBVR VoD server	*Video server:* • *Extreme FoV adaptation with extra short GOP (100ms) 60 fps* • *Single bitrate to avoid influence of ABR*	*To be provided by YBVR from AWS*
VR handset	*Video client device*	*To be provided by YBVR*
Video feed	*For local video ingestion in the uplink test*	*To be provided by YBVR*

Experiment architecture:

<u>*Architecture 1: Immersive experience, for live participation in events*</u>



Architecture 2: Virtual tickets, for on-demand participation in events

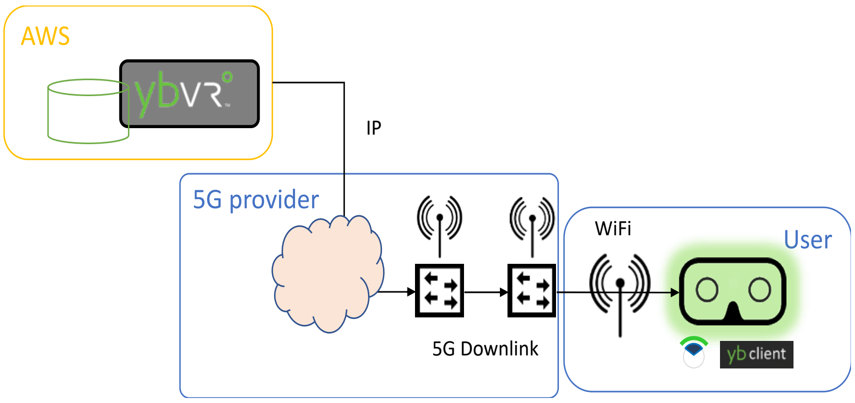


Table 25: Smart Tourism UC meaningful KPIs

5G related KPIs	Comments	
Download bitrate	*At least, consistent 50Mbps are expected*	
Upload bitrate	*Target would be consistent 200Mbps*	
Vertical KPIs	**Is it measurable during experimentation?**	**Can it be mathematically derived from 5G related KPIs?**
Aggregated viewport adaptation time	*Yes*	*No mathematical formula known*
Aggregated time to right frame (TtRF)	*Yes*	*No mathematical formula known*
Camera changes during video playback	*Yes*	*No mathematical formula known*

Table 26: Smart Tourism UC test plan

Test Case 1	
Test Name:	*Effect of delay in viewport changes*
Target KPI:	*Aggregated viewport adaptation time, TtRF*
Measurement Method:	*YBVR player simulate viewport changes and compare requested and reproduced viewport timing*
Parameters:	*The video source location will be exclusively in AWS.* *The mobile network will change from 4G to 5G, for comparison.* *Scenario of Architecture 2. Additional delay to be included (+50ms for congestion or +150ms for severe congestion)*
Validation Conditions:	*There exist specific guideline graphics. Examples:* *Viewport change speed: 50% of viewports under 200ms delay; 90% under 350ms; 99% under 500ms* *TtRF: 500ms or less during the entire playback; 200ms or less for 95% of the playback*

Test Case 2	
Test Name:	*Effect of delay in instant camera changes*
Target KPI:	*Camera changes, TtRF*
Measurement Method:	*YBVR player simulate camera changes and compare requested and reproduced viewport timing*
Parameters:	*The video source location will be exclusively in AWS.* *The mobile network will change from 4G to 5G, for comparison.* *Scenario of Architecture 2. Additional delay to be included (+50ms for congestion or +150ms for severe congestion)*
Validation Conditions:	*5G should get at least 50% more camera changes than 4G.*

Test Case 3	
Test Name:	*Downstream distribution*
Target KPI:	*Download bitrate*
Measurement Method:	*VoD content uses special encoding settings: High quality VR video (up to 50Mbps of bitrate); two cameras: 360 degrees and 180 degrees; single bitrate.* *YBVR player will meter regularly download bitrate*
Parameters:	*The video source location will be exclusively in AWS.* *The mobile network will change from 4G to 5G, for comparison.* *Scenario of Architecture 2. Additional delay to be included (+50ms for congestion or +150ms for severe congestion)*
Validation Conditions:	*5G should consistently hit the top bitrate mark of 50Mbps. The same could be achieved in 4G, although some fluctuations could happen*

Test Case 4	
Test Name:	*Upstream distribution*
Target KPI:	*Upload bitrate*
Measurement Method:	*VoD content uses special encoding settings: High quality VR video (up to 50Mbps of bitrate); two cameras: 360 degrees and 180 degrees; single bitrate.* *YBVR player will meter regularly upload bitrate*
Parameters:	*The YBVR platform will be exclusively in AWS.* *The mobile network will change from 4G to 5G, for comparison.* *Scenario of Architecture 1. Additional delay to be included (+50ms for congestion or +150ms for severe congestion)*
Validation Conditions:	*5G should consistently hit the top bitrate mark of 200Mbps. The same could not be achieved with 4G.*

Annex I.5: Experiment template for Transport use case (Spain)

The following has been extracted from the template response for the Transport use case in the Italian site.

Experiment Description:

The Transport vertical use cases can be tested applying for a tracking and recognition service experiment where an end user, equipped with a 5G terminal, moves around a 5G covered area performing different transportation modes (e.g. walking, biking, driving a vehicle or using public transportation means) and sending the collected data from the mobile device sensors (mainly GPS, accelerometer, magnetometer and gyroscope) to a backend platform where the real-time data are elaborated to estimate the end user transportation mode.

Expected Capabilities:

A high reliable and low-latency mobile network connecting the 5G terminal to the platform with the virtual machines hosting the tracking and recognition services is required. The most challenging requirements to ensure the deployment of the above described solution in the future smart cities are the high reliability and the low latency.

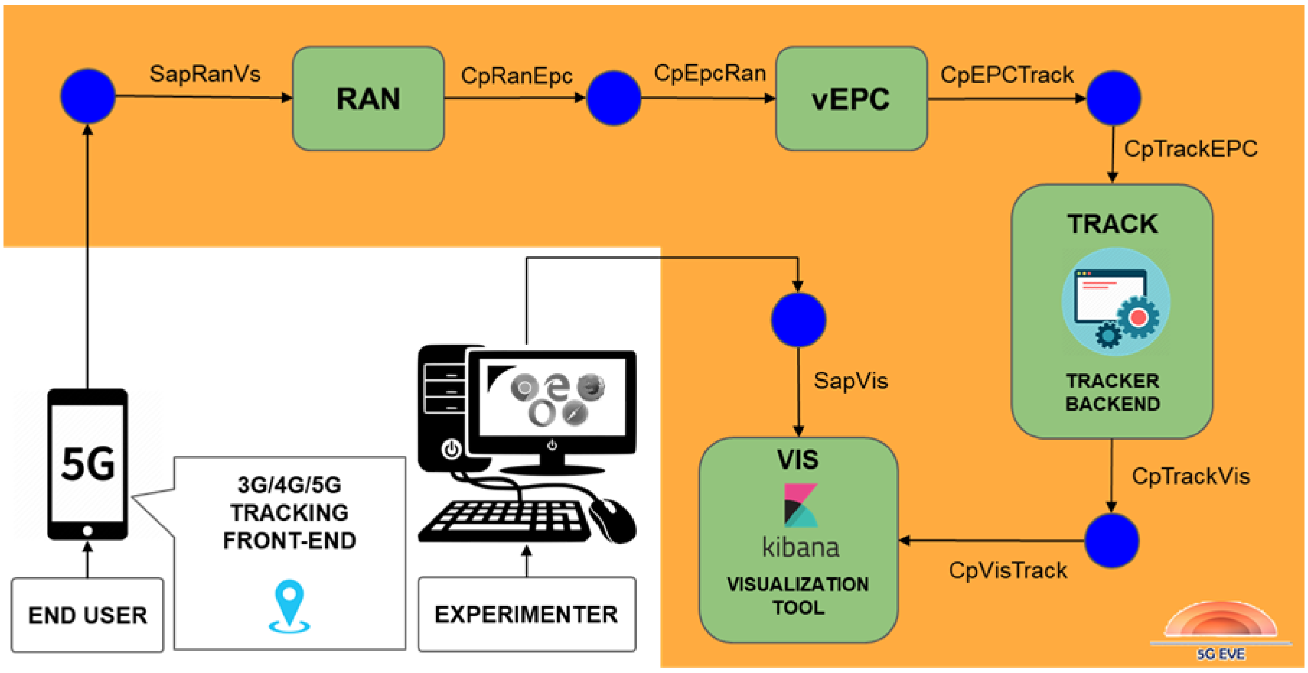
Table 27: Transport UC experiment components

Component Name	Description	Deployment requirements
User Equipment (UE)	*User 5G terminal (smartphone)*	*User terminal to connect to 5G infrastructure.*
Android App	*App collecting mobility data from smartphone sensors.*	*User terminal equipped with mobility sensors (e.g., GPS, magnetometer, accelerometer, gyroscope, ...)*
Backend	*Virtual machine hosting tracker backend and recognition backend services.*	*Virtual machine to run Linux Ubuntu + required software to run the service (e.g., php, python, anaconda)*
Visualization tool	*Virtual machine hosting data visualization tool.*	*Virtual machine to run Linux Ubuntu + ELK Stack.*

Experiment architecture:

While end users feed the recognition service with real-time data, the experimenter visualizes the statistics and related reports on the estimated transportations modes.

Tracking service experiment architecture:



Recognition service experiment architecture:

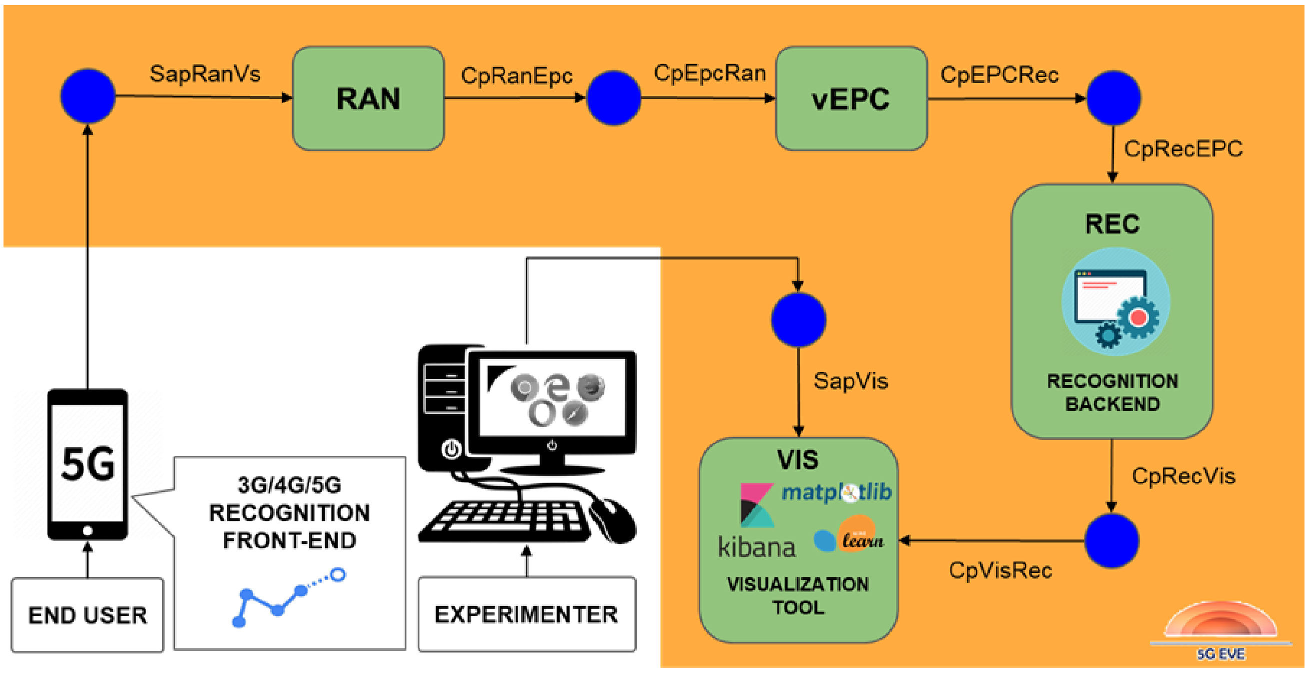


Table 28: Transport UC meaningful KPIs

5G related KPIs	Comments
Delay	*Network delay affects the transportation mode estimation*
Availability	*5G MEC IT resources availability affects the transportation mode estimation*

Vertical KPIs	Is it measurable during experimentation?	Can it be mathematically derived from 5G related KPIs?
Tracking response time	*Yes, by visualization tool (Kibana)*	*No mathematical formula known*
Memory usage	*Yes, by visualization tool (Kibana)*	*No mathematical formula known*
Recognition delay[4]	*Yes, by backend (Recognition service)*	*Proportional to network delay*

Table 29: Transport UC test plan

Test Case 1	
Test Name:	*Effect of number of connected devices on tracking response time performed at MEC*
Target KPI:	*Tracking response time*
Measurement Method:	*The tracking response time will be measured by the Kibana visualization tool: it measures the time spent to evaluate the selected amount of data and visualize them*
Parameters:	*No external traffic.* *Number of users: from 1 to hundreds*
Validation Conditions:	*Tracking response time <= 200ms*

Test Case 2	
Test Name:	*Effect of number of connected devices on MEC memory usage*
Target KPI:	*Memory usage*
Measurement Method:	*The memory usage will be measured by the Kibana visualization tool: it measures the percentage of used memory over the total available memory*
Parameters:	*No external traffic.* *Number of users: from 1 to hundreds*
Validation Conditions:	*Memory usage < 90%*

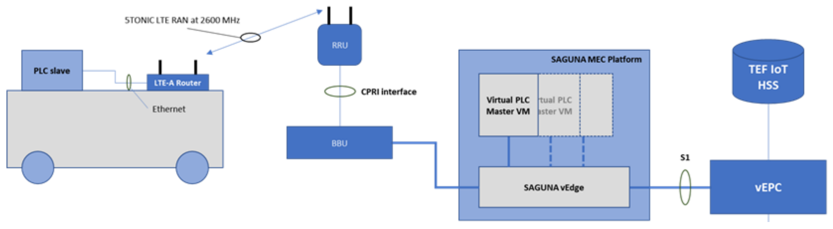
Test Case 3	
Test Name:	*Effect of network delay and availability on transportation mode recognition capability*
Target KPI:	*Recognition delay*
Measurement Method:	*The recognition delay will be measured by the backend recognition service: it measures the time needed to estimate the correct transportation mode, when a transportation mode occurs*
Parameters:	*No external traffic.* *Number of users: from 1 to hundreds* *Delay: 1ms, 10ms, 100ms, 1s*
Validation Conditions:	*The impact of network delay is less than 10% of the recognition delay*

[4] This vertical KPI is still to be defined and implemented

Annex II: Low-level test plan templates – available responses

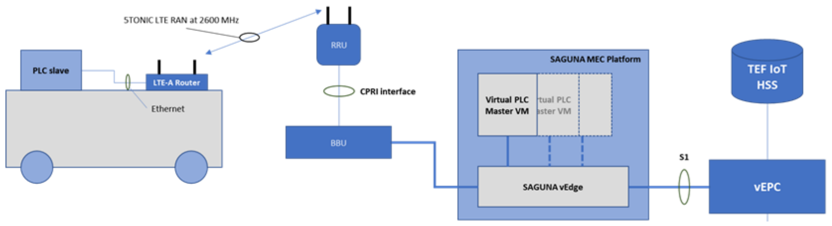
Annex II.1: Experiment template for Industry 4.0 use case (Spain)

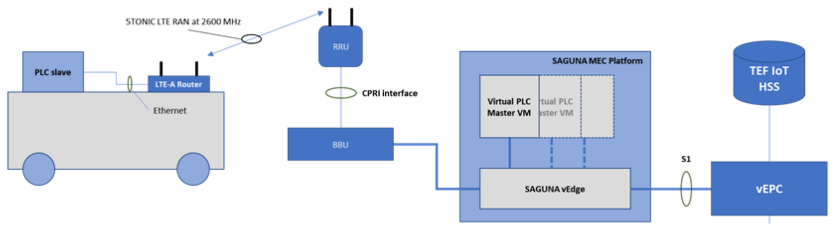
The following has been extracted from the template response for the Industry 4.0 use case in the Spanish site.

Test Case 1		
Test Name:		*Effect of latency in Navigation Level*
Test Objective:		*Infrastructure metrics: "latency"* *Application metrics: "navigation_level" (obtained from Vertical's components, not from traffic tools)* *Target KPI: "navigation_level"* *In this case, the KPI that wants to be validated coincides with a metric that is provided by an ASTI component. The latency is the variable parameter, so it is also required to monitor this metric.*
Test Prerequisites:		*The "navigation_level" is measured by the AGV, and sent to the Master PLC as part of their communication. A mechanism will have to be implemented to propagate this metric to the Kafka bus. Options:* *a) Capture traffic between AGV and Master PLC, decode it, and publish in Kafka (this would require an additional element not provided by ASTI)* *b) Check if Master PLC can publish these values in Kafta* *The AGVs need specific space to deploy the circuit and associated components (e.g. battery charger) → this limits the hosting sites to 5TONIC*
Required Capabilities:		*In the hosting site we will need:* • *Compute capacity at MEC to host the Master PLC, a Windows based VM (requirements to be determined)* ○ *Local compute resources should be controlled by the local NFVO, so that instantiation can be triggered by the Multi-site NSO* • *A component emulating latency, under the control of the Runtime Configurator* • *Full 5G network: access + EPC*
Sub-Case 1	**Test Topology:**	 5TONIC LTE RAN at 2600 MHz PLC slave LTE-A Router Ethernet RRU CPRI interface BBU SAGUNA MEC Platform Virtual PLC Master VM tual PLC aster VM SAGUNA vEdge S1 TEF IoT HSS vEPC *The AGV counts with a router to connect to the 5G network. The Master PLC is deployed at MEC, so the latency emulator must be placed somewhere between the AGV and the MEC.*
	Test Variables:	*The test variable is the latency. It must be noted that the additional latency introduced by the latency emulator is not the latency E2E. It is not trivial to measure this latency, since the Slave PLC at the AGV is not designed to do it. Either we can connect an additional device to the same router, or we should*

		characterize the initial latency, and add it to the emulated one when results are presented.
	Test Procedure:	***Deployment**: both the Master PLC and the latency emulator should be deployed real time as part of this phase.* ***Procedure**: after deployment, the AGV should start running (to be checked if this can be done automatically)* *Initial emulated "latency" = 0ms; "step" = 5ms* *Loop*:* *run experiment for 3 minutes* *latency = latency + step* **The ideal would be to execute the loop until the target KPI passes the validation threshold. However, since the validation cannot be done in real time at this stage, the number of runs of the loop is something to be determined. We might also provide a manual mechanism to stop the test, as it is evident to experimenters when the AGV is working far beyond the correct performance conditions.* ***Monitoring**: it is mandatory to receive the "navigation level" from the AGV continuously from the start of the experiment. This shall be visualized together with the E2E latency.*
	Expected results:	*Once the loop is finished, all the "navigation level" values will be compared with the specified threshold (still to be defined). If all the values are lower than the threshold, the result will be PASS, if at least one value is higher, the result will be FAIL.* *The test report should include the graphics "navigation level" versus "latency", plus the final result (PASS/FAIL), and the latency value in which the experiment fails, if this is the case.*

Test Case 2	
Test Name:	*Maximum supported latency*
Test Objective:	*Infrastructure metrics: "latency"* *Application metrics: "guide_lost" (obtained from Vertical's components, not from traffic tools; it is a log that appears in the Master PLC)* *Target KPI: "time_to_lose_the_guide"* *The truth about the KPI that wants to be validated is that it is a "latency" value. The name "time to lose the guide" means that only a specific latency value is desired: the one at which the "guide lost" signal appears in the system (which is also when the AGV stops). The latency is the variable parameter, so it is also required to measure and store this metric.*
Test Prerequisites:	*The "guide_lost" signal is sent to the Master PLC, where it appears as a log. A mechanism will have to be implemented to announce the appearance of this log to the Kafka bus. Options:* *a) Capture traffic between AGV and Master PLC, decode it, and publish in Kafka the message where the guide_lost signal is sent* *b) Check if Master PLC can publish this message in Kafta* *The AGVs need specific space to deploy the circuit and associated components (e.g. battery charger) → this limits the hosting sites to 5TONIC*
Required Capabilities:	*In the hosting site we will need:*

		• *Compute capacity at MEC to host the Master PLC, a Windows based VM (requirements to be determined)* ○ *Local compute resources should be controlled by the local NFVO, so that instantiation can be triggered by the Multi-site NSO* • *A component emulating latency, under the control of the Runtime Configurator* • *Full 5G network: access + EPC*
	Test Topology:	 STONIC LTE RAN at 2600 MHz RRU PLC slave LTE-A Router Ethernet CPRI interface BBU SAGUNA MEC Platform Virtual PLC Master VM tual PLC ster VM SAGUNA vEdge S1 TEF IoT HSS vEPC *The AGV counts with a router to connect to the 5G network. The Master PLC is deployed at MEC, so the latency emulator must be placed somewhere between the AGV and the MEC.* *Capabilities and Topology in Test Cases 1 and 2 are totally the same, so the same vertical blueprint can be used.*
	Test Variables:	*The test variable is the latency. It must be noted that the additional latency introduced by the latency emulator is not the latency E2E. It is not trivial to measure this latency, since the Slave PLC at the AGV is not designed to do it. Either we can connect an additional device to the same router, or we should characterize the initial latency, and add it to the emulated one when results are presented.*
Sub-Case 1	**Test Procedure:**	***Deployment****: both the Master PLC and the latency emulator should be deployed real time as part of this phase.* ***Procedure****: after deployment, the AGV should start running (to be checked if this can be done automatically)* *Initial emulated "latency" = 0ms; "step" = 5ms* *Loop*:* *run experiment for 3 minutes* *latency = latency + step* **The ideal would be to execute the loop until the "guide_lost" signal appears. However, since the validation cannot be done in real time at this stage, the number of runs of the loop is something to be determined. We might also provide a manual mechanism to stop the test, as it will be evident to experimenters when the AGV stops completely.* ***Monitoring****: the KPI is a specific latency value, the one experienced when the application metric appears. Since this is a single occurrence event, it makes little sense to visualize it. The latency could be presented, however: it will provide zero real information to the experimenter, but it will help them realize that the experiment is running (they will see changes in the latency).*
	Expected results:	*The result of the experiment will be a "time to lose the guide" value, which will be the latency value experienced in the scenario when the "guide_lost" signal is generated. There is not a comparison to be done in this scenario that determines the PASS/FAIL condition.* *The test report should include the "time to lose the guide" only, as in this case, including a graphic with the latency will not be useful.*

Test Case 3		
Test Name:		*Effect of latency in Consumption*
Test Objective:		*Infrastructure metrics: "latency"* *Application metrics: "consumption" (obtained from Vertical's components, not from traffic tools)* *Target KPI: "consumption"* *In this case, the KPI that wants to be validated coincides with a metric that is provided by an ASTI component. The latency is the variable parameter, so it is also required to monitor this metric.*
Test Prerequisites:		*The "consumption" is measured by the AGV, and sent to the Master PLC as part of their communication. A mechanism will have to be implemented to propagate this metric to the Kafka bus. Options:* *a) Capture traffic between AGV and Master PLC, decode it, and publish in Kafka (this would require an additional element not provided by ASTI)* *b) Check if Master PLC can publish these values in Kafta* *The AGVs need specific space to deploy the circuit and associated components (e.g. battery charger) → this limits the hosting sites to 5TONIC*
Required Capabilities:		*In the hosting site we will need:* • *Compute capacity at MEC to host the Master PLC, a Windows based VM (requirements to be determined)* ○ *Local compute resources should be controlled by the local NFVO, so that instantiation can be triggered by the Multi-site NSO* • *A component emulating latency, under the control of the Runtime Configurator* • *Full 5G network: access + EPC*
Sub-Case 1	**Test Topology:**	 5TONIC LTE RAN at 2600 MHz PLC slave LTE-A Router Ethernet RRU CPRI interface BBU SAGUNA MEC Platform Virtual PLC Master VM SAGUNA vEdge S1 TEF IoT HSS vEPC *The AGV counts with a router to connect to the 5G network. The Master PLC is deployed at MEC, so the latency emulator must be placed somewhere between the AGV and the MEC.* *Capabilities and Topology in Test Cases 1 and 3 are totally the same, so the same vertical blueprint can be used. In fact, the only difference between these two scenarios is the target KPI.*
	Test Variables:	*The test variable is the latency. It must be noted that the additional latency introduced by the latency emulator is not the latency E2E. It is not trivial to measure this latency, since the Slave PLC at the AGV is not designed to do it. Either we can connect an additional device to the same router, or we should characterize the initial latency, and add it to the emulated one when results are presented.*

	Test Procedure:	***Deployment**: both the Master PLC and the latency emulator should be deployed real time as part of this phase.* ***Procedure**: after deployment, the AGV should start running (to be checked if this can be done automatically)* *Initial emulated "latency" = 0ms; "step" = 5ms* *Loop*:* *run experiment for 3 minutes* *latency = latency + step* **The ideal would be to execute the loop until the target KPI passes the validation threshold. However, since the validation cannot be done in real time at this stage, the number of runs of the loop is something to be determined. We might also provide a manual mechanism to stop the test, as it is evident to experimenters when the AGV is working far beyond the correct performance conditions.* ***Monitoring**: it is mandatory to receive the "consumption" from the AGV continuously from the start of the experiment. This shall be visualized together with the E2E latency.*
	Expected results:	*Once the loop is finished, all the "consumption" values will be compared with the specified threshold (still to be defined). If all the values are lower than the threshold, the result will be PASS, if at least one value is higher, the result will be FAIL.* *The test report should include the graphics "consumption" versus "latency", plus the final result (PASS/FAIL), and the latency value in which the experiment fails, if this is the case.*

Annex III: Experiment Execution workflow; testing, validation and monitoring procedures

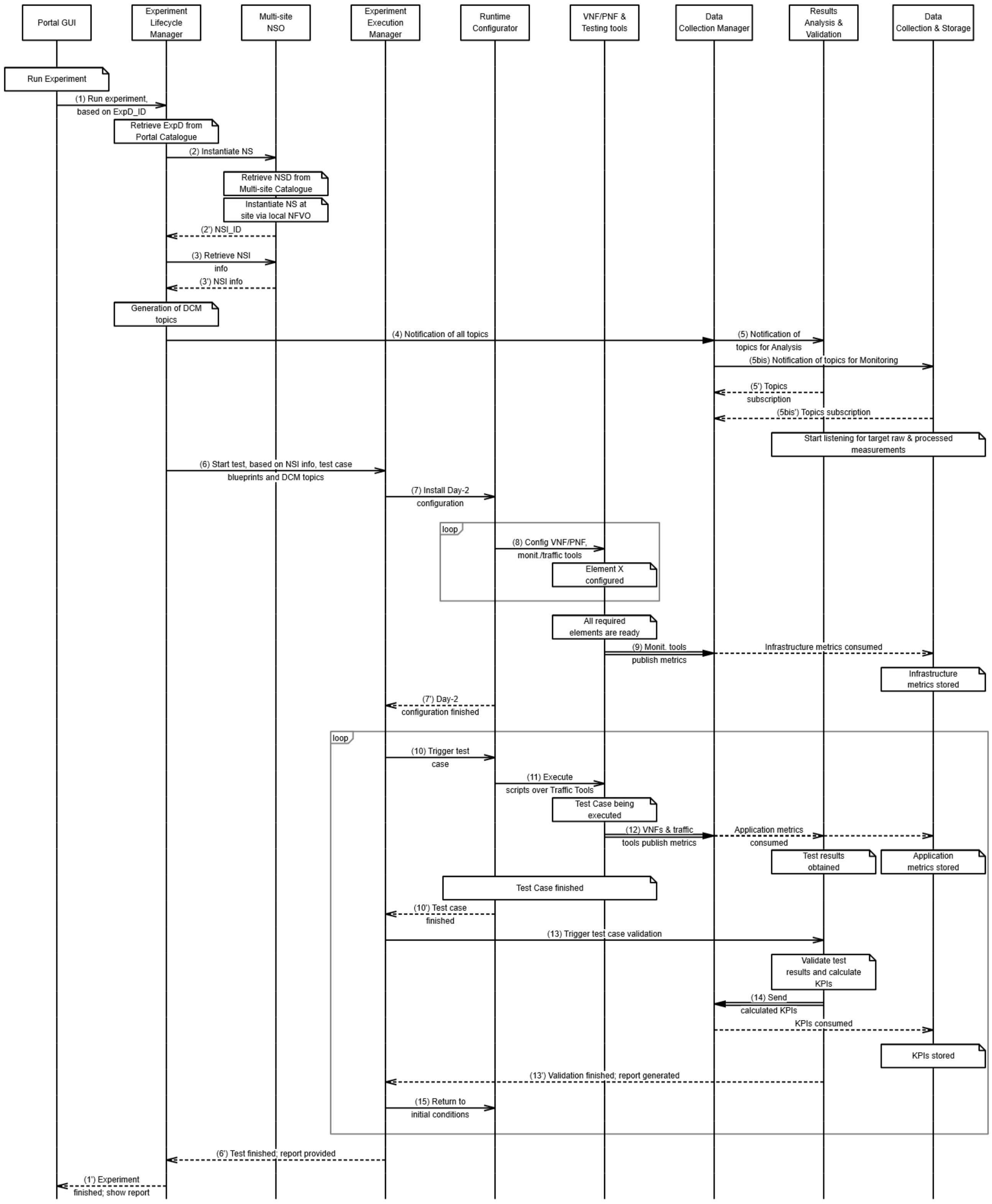

Portal GUI
Experiment Lifecycle Manager
Multi-site NSO
Experiment Execution Manager
Runtime Configurator
VNF/PNF & Testing tools
Data Collection Manager
Results Analysis & Validation
Data Collection & Storage
Run Experiment
(1) Run experiment, based on ExpD_ID
Retrieve ExpD from Portal Catalogue
(2) Instantiate NS
Retrieve NSD from Multi-site Catalogue
Instantiate NS at site via local NFVO
(2') NSI_ID
(3) Retrieve NSI info
(3') NSI info
Generation of DCM topics
(4) Notification of all topics
(5) Notification of topics for Analysis
(5bis) Notification of topics for Monitoring
(5') Topics subscription
(5bis') Topics subscription
Start listening for target raw & processed measurements
(6) Start test, based on NSI info, test case blueprints and DCM topics
(7) Install Day-2 configuration
loop
(8) Config VNF/PNF, monit./traffic tools
Element X configured
All required elements are ready
(9) Monit. tools publish metrics
Infrastructure metrics consumed
Infrastructure metrics stored
(7') Day-2 configuration finished
loop
(10) Trigger test case
(11) Execute scripts over Traffic Tools
Test Case being executed
(12) VNFs & traffic tools publish metrics
Application metrics consumed
Test results obtained
Application metrics stored
Test Case finished
(10') Test case finished
(13) Trigger test case validation
Validate test results and calculate KPIs
(14) Send calculated KPIs
KPIs consumed
KPIs stored
(13') Validation finished; report generated
(15) Return to initial conditions
(6') Test finished; report provided
(1') Experiment finished; show report

Figure 38: Experiment Execution; testing, validation and monitoring procedures