5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief

Innovation Action – ICT-41-2020 - 5G PPP – 5G

Innovations for verticals with third party services

D2.5: 5G-EPICENTRE Experiment execution

Delivery date: October 2023

Dissemination level: Public

Project Title:	**5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-nativE Network Applications for public proTection and disaster RElief**
Duration:	**1 January 2021 – 31 December 2023**
Project URL	https://www.5gepicentre.eu/

This project has received funding from the European Union's Horizon 2020 Innovation Action programme under Grant Agreement No 101016521.

Document Information

Deliverable	D2.5: 5G-EPICENTRE Experiment execution
Work Package	WP2: Cloud-native 5G NFV
Task(s)	T2.4: Experiment coordination and lifecycle management
Type	Report
Dissemination Level	Public
Due Date	M34, October 31, 2023
Submission Date	M34, October 31, 2023
Document Lead	Almudena Díaz Zayas (UMA)
Contributors	Jorge Márquez Ortega (UMA) Apostolos Siokis (IQU)
Internal Review	Konstantinos Apostolakis (FORTH) Hamzeh Khalili (CTTC)

Document history

Version	Date	Changes	Contributor(s)
V0.1	17/07/2023	Initial deliverable structure	Almudena Díaz Zayas (UMA)
V0.2	15/07/2023	50% of the deliverable content	Almudena Díaz Zayas (UMA) Jorge Márquez Ortega (UMA)
V0.3	02/10/2023	90% of the deliverable content	Almudena Díaz Zayas (UMA) Jorge Márquez Ortega (UMA) Apostolos Siokis (Iquadrat)
V1.0	16/10/2023	Internal Review Version	Almudena Díaz Zayas (UMA)
V1.1	24/10/2023	1st version with suggested revisions	Hamzeh Khalili (CTTC)
V1.2	24/10/2023	2nd version with suggested revisions	Konstantinos Apostolakis (FORTH)
V1.3	29/10/2023	First revisions after internal review	Almudena Díaz Zayas (UMA) Jorge Márquez Ortega (UMA)
V1.5	31/10/2023	Quality check/review: formatting, typesetting and proof-reading.	Konstantinos Apostolakis (FORTH)
V2.0	31/10/2023	Final version for submission	Almudena Díaz Zayas (UMA)

Project Partners

Logo	Partner	Country	Short name
	AIRBUS DS SLC	France	**ADS**
	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	**NOVA**
	Altice Labs SA	Portugal	**ALB**
	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	**HHI**
	Foundation for Research and Technology Hellas	Greece	**FORTH**
	Universidad de Malaga	Spain	**UMA**
	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	**CTTC**
	Istella SpA	Italy	**IST**
	One Source Consultoria Informatica LDA	Portugal	**ONE**
	Iquadrat Informatica SL	Spain	**IQU**
	Nemergent Solutions S.L.	Spain	**NEM**
	EBOS Technologies Limited	Cyprus	**EBOS**
	Athonet SRL	Italy	**ATH**
	RedZinc Services Limited	Ireland	**RZ**
	OptoPrecision GmbH	Germany	**OPTO**
	Youbiquo SRL	Italy	**YBQ**
	ORamaVR SA	Switzerland	**ORAMA**

List of abbreviations

Abbreviation	Definition
5GTSM	5G Traffic Simulator Manager
API	Application Programming Interface
DB	Database
ExaaS	Experiments as a Service
GA	Grant Agreement
HSPF	Holistic Security and Privacy Framework
K8s	Kubernetes
KPI	Key Performance Indicator
MANO	Management and Orchestration
MQ(TT)	Message Queueing (Telemetry Transport)
N/A	Non Applicable
(C/V)NF	Network Function
NFV	Network Functions Virtualization
NS	Network Service
QoE	Quality of Experience
QoS	Quality of Service
UC	Use Case
UI	User Interface
WP	Work Package
ELCM	Experiment Lifecycle Manager
JSON	Java Script Object Notation
YAML	Yet Another Markup Lenguage
REST	Respresentational State Transfer

Executive summary

This deliverable is the result of the work carried out in Task 2.4 of the 5G-EPICENTRE project, where the main goal is to apply an experimentation methodology to test the performance of the solutions being experimented with, in order to assess their capabilities.

The deliverable describes the experimentation workflow used to perform the testing of network applications on the 5G-EPICENTRE platforms. The correct realization of the experimentation workflow is key to understand how to use the infrastructure to test applications. Moreover, the document also includes a detailed description of the software tools developed within the context of Task 2.4. The description of these components and their deployment will also enable testbed operators to understand and adopt/modify the experimentation methodology.

Based on the above, the main contributions of this deliverable are to describe the full experimentation workflow, the components developed to support it, and the instructions to deploy these components in new platforms, that would be interested in adopting this methodology.

Table of Contents

List of Figures

List of Tables

1 Introduction

A description of the elements involved in the execution of experiments will be presented in this deliverable. A detailed view of its operation and architecture (which is not available in other deliverables) will be presented.

The elements to be described are:

- **Experiment Coordinator:** Element in charge of the coordination of deployments and the execution of the different experiments. This element will act as a link between the Portal and the rest of the platform.
- **5G Traffic Simulator Manager:** This section will describe the element in charge of 5G traffic generation in the different testbeds associated to the platform. To do so, it will make use of the **remote iPerf agents** that will also be described in the same section.
- **Publisher:** The publisher is the element in charge of publishing the different measurements taken from the testbeds that make up the platform, as well as collecting and verifying the different KPIs generated by the experiments that are executed on them.

1.1 5G-EPICENTRE experimentation methodology

The experimentation methodology used in the 5G-EPICENTRE project is an extension of the methodology defined in the 5Genesis project[1]. The 5Genesis experiment framework was successfully applied to evaluate mission critical services. Moreover, the experimentation methodology is test case oriented, which is one of the features identified in Task 2.4 for the experiment coordinator.

The 5Genesis project has defined and implemented a methodology for conducting experiments at the platforms level. The 5Genesis experimentation workflow is defined as follows:

- **Design and preparation of the experiment.**
  - Identification of the Key Performance Indicators (KPIs) relevant for the use case and the network scenarios.
  - Definition of the test cases that will enable obtaining the measurements needed to compute the targeted KPIs and the parametrization of the scenarios.
  - Application under test is instrumentalized in order to report the needed measurements to obtain the targeted KPIs.
  - Implementation of the test cases and scenarios.
- **Definition of the experiment.** After the initial consultancy phase, the experimenter has a list of test cases and scenarios available in the testbed. The experimenter can choose and combine the test cases and scenarios in different experiments. Formal description of the experiment is contained inside an *Experiment Descriptor*, whose information is needed to configure the network and execute the test cases.
- **Experiment execution.** The desired/requested network configuration is applied, and the experiment is executed. The application under test as well as the network is under continuous monitoring. The execution of the experiment can be automated via the definition of automation scripts, or can be executed manually by the experimenters, or by the end users of the solutions.
- **Results and reporting.** Measurements collected are available in different formats that can be accessed for the report's elaboration and KPIs' calculations.

The key component of the 5Genesis experimentation methodology is the *Experiment Lifecycle Manager (ELCM)* [1], which has been published as an open-source component. This component has been adopted in the 5G-EPICENTRE project for the execution of experiments at the platform level. For the installation of the ELCM in each testbed it is necessary to install the OpenTAP[2] tool before, which allows the automation of the different scripts

[1] https://5genesis.eu/
[2] https://opentap.io/

executed in the testbed. For the purposes of the 5G-EPICENTRE architecture, the ELCM is considered part of the platform as a service infrastructure block. To integrate this component into the 5G-EPICENTRE architecture, the 5G-EPICENTRE Experiment Coordinator includes an interface to communicate with the ELCMs, deployed at the different testbeds.

After adopting this component, the experimentation workflow in 5G-EPICENTRE architecture is listed below. Figure 1 provides an overview of the 5G-EPICENTRE components that support this experimentation workflow.

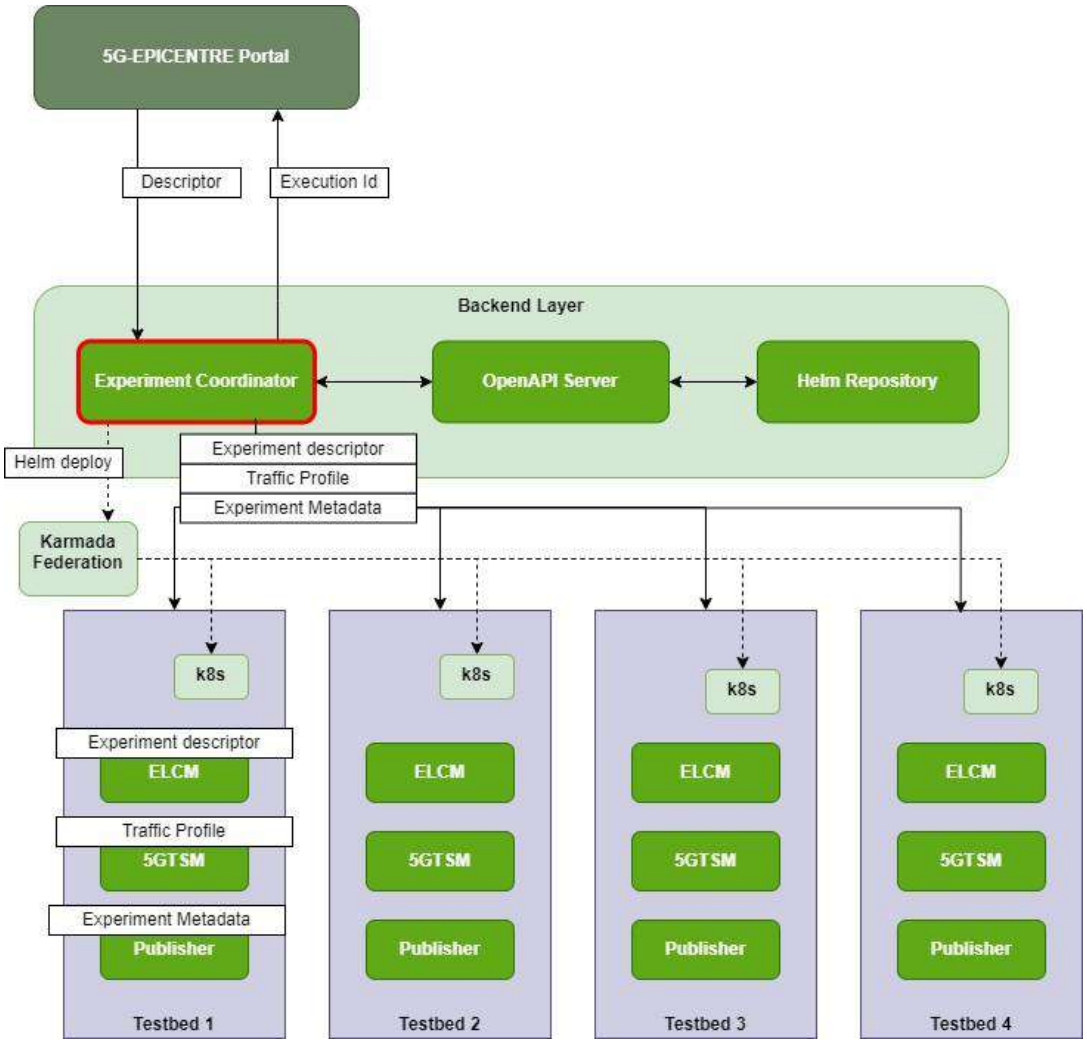

Figure 1: Components involved in the experiment execution

- **Step 1:** The experiment is defined at the 5G-EPICENTRE portal (see D3.2).
- **Step 2:** The 5G-EPICENTRE portal sends the experiment descriptor of the experiment to the Experiment Coordinator (described in Section 2). The information received is registered in the Experiment Coordinator, and the experiment metadata is generated. The experiment metadata includes the assigned Experiment Identifier and the Execution Identifier. These two identifiers are key to track the execution of the experiments and aggregate the measurements of the same experiment in each one of their repetitions.
- **Step 3:** The Experiment Coordinator is in charge of the deployment of the solutions under test. The images are downloaded from the Network Service Helm repository (see D4.3), and deployed via Karmada[3] on the Kubernetes (K8s) cluster of the platform chosen by the experimenter. An alternative option is for

[3] https://karmada.io/

Karmada to choose the better placement, in case there is not a specific experimentation requirement that involves its deployment on a platform that meets the requirement.

- **Step 4:** The Experiment Coordinator communicates with the 5G Traffic Simulation Manager (5GTSM, described in Section 3), to configure and initiate traffic generation in accordance with the traffic profile defined during the definition of the experiment.
- **Step 5:** The Experiment Coordinator populates the metadata of the experiment via the Publisher (described in Section 4). This step is needed to "stamp" all the measurements with the correct Experiment Id and Execution Id.
- **Step 6:** The experiment is executed, either in automated or manual execution option (depending on the experimenter preference indicated in the Portal at Step 1). Infrastructure-, network- and application-level measurements, are injected in the RabbitMQ queue defined as part of the 5G-EPICENTRE architecture (the format of the measurements is defined in D4.1). The 5G-EPICENTRE Analytics Engine (see D2.6) is in charge of the analysis of the results, and for sending them back to the Portal for KPIs' visualization.

The implementation details and the Application Programming Interfaces (APIs) offered by these components are introduced in subsequent sessions. Moreover, the instructions for the deployment of these components are provided, in order to anticipate other, external testbeds being integrated into the 5G-EPICENTRE ecosystem (after the conclusion of the project, as part of the platform exploitation plan – see D6.6 [6]).

1.2 Mapping of project's outputs

The purpose of this section is to map 5G-EPICENTRE Grant Agreement (GA) commitments, both within the formal Deliverable and Task description, against the project's respective outputs and work performed.

Table 1: Adherence to 5G-EPICENTRE's GA Deliverable & Tasks Descriptions

5G-EPICENTRE Task	Respective Document Chapters	Justification
T2.4: Experiment coordination and lifecycle management *"[…] With respect to the Experiment Coordinator, this Task will develop components for: i) experiment scheduling, where execution of the experiment planned by an experimenter through the Experiment Planner (T3.3) is orchestrated"*	Section 2.1.1 – Scheduler	Section 2 presents details of the internal operation of the Experiment Coordinator component, which in in charge of the management of the lifecycle of the experiment. Section 2.1.1 in particular, presents details of the internal component that handles planning of the experiment, and manages their execution at pre-run stage.
T2.4: Experiment coordination and lifecycle management *"[…] ii) experiment timeslot running process queue AI-manager, which will autonomously monitor availability of resources so as to determine which experiments will be ran on top of the infrastructure (including concurrent execution, if resources permit)"*	Section 2.1.2 – Resource availability	Section 2 presents details of the internal operation of the Experiment Coordinator component, which in in charge of the management of the lifecycle of the experiment. Section 2.1.2 in particular, presents details on the feasibility study check that is performed, which creates a waiting loop until

		the required resources to execute the experiment are available.
T2.4: Experiment coordination and lifecycle management *"[…] the Multi-container application composer, which will produce the necessary experiment configuration files that identify VNF containers to be instantiated during experiment execution".*	Section 2.1.3 – Experiment configuration	Section 2 presents details of the internal operation of the Experiment Coordinator component, which in in charge of the management of the lifecycle of the experiment. Section 2.1.3 presents the different configurations for correct operation of the experiments.
T2.4: Experiment coordination and lifecycle management *"[…]The Experiment Coordinator will also implement the execution of standardized test cases defined by standardization or defined by the project".*	Section 2.2.3 – Tasks	This Section describes 'Test Cases' as a list of tasks (minimum action that must be performed to deploy and execute an experiment) to be executed internally throughout different stages of experiment execution.
T2.4: Experiment coordination and lifecycle management *"[…] this Task will develop: i) an autonomous service manager, which aims an intelligent management and decommission of the necessary resources by specifying the network slices to be deployed and communicating with the layers below so as to ensure proper assignation of the resources"*	Section 2.2 – Implementation	Section 2.2 presents the execution logic of the Experiment Coordinator, elaborating on all internal processes that take place, from the moment the experiment descriptor is received, to the actual deployment and execution of the experiment.
T2.4: Experiment coordination and lifecycle management *"[…] ii) a dummy 5G traffic simulator, to emulate varying network traffic dynamics".*	Section 3 – 5G Traffic Simulator Manager	Section 3 presents the Traffic Simulator, developed to generate background traffic during the execution of the experiments.
	Section 4 – Publisher	Section 4 presents the details of the Publisher which is in charge of publishing the metadata of the experiments to include it in the measurements collected at the testbed level.

2 Experiment Coordinator

The Experiment Coordinator is the component in charge of coordinating the deployment and execution of the experiments on the platforms belonging to the 5G-EPICENTRE federation in the selected time slot. It is a centralized element, located at the backend layer of the 5G-EPICENTRE architecture, that receives the descriptions of the actions to be performed from the **Portal** (via an experiment descriptor exchange structure).

The descriptor is a JavaScript Object Notation (JSON) file, that contains all the needed information for the deployment of the use case, network and traffic configuration and the execution of the experiment.

In order to understand the role of the Experiment Coordinator, it is important to elaborate on the format of the descriptor. The final version of this descriptor is outlined in Table 2, below:

Table 2: Experiment descriptor

Field	Type	Description
Application	string	Name of the application under test.
Automated	bool	Indicates whether the experiment will be run automatically or not.
ExclusiveExecution	bool	Indicates whether the execution is to be exclusive or not.
ExperimentType	string	Can either be “Standard” or “Custom”. Standard experiments are based on test cases.
Extra.Url	string	extra field used to for additional information required to executed the actions, for example, to indicate URL of the 5GTSM instance.
Extra.ServerProbes	Array<Object>	Field use for the configuration of the remote iperf agents servers.
Extra.ClientProbes	Array<Object>	Field use for the configuration of the remote iperf agents servers.
NSs	Array<string>	Name of the services to be deployed.
Parameters.Action	string	Can either be “deploy” or “delete”. Indicates if the experimenter wants to deploy, or delete the experiment in the descriptor.
Parameters.Filename	string	Name of helm file to be deployed.
Parameters.Testbed_id	integer	Identifier of the federated testbed.
Parameters.Namespace	string	Name of the federated k8s cluster where you want to deploy
Parameters.Netapp_id	string	Network application identifier

HSPF	string	Can either be “yes” or “no”. Indicates whether the Holistic Security and Privacy Framework network application (see D2.7) should be deployed alongside the current experiment vertical application microservices.
Parameters.HSPF_microservices	Array<string>	Indicates which microservices should have sidecars deployed for monitoring.
Parameters.HSPF env_vars	Array<string>	Additional field for the Holistic Security and Privacy Framework network application configuration.
Remote	bool	Unused field for 5G-EPICENTRE (expected by the ELCM, so needs to be added with a default value of *null*).
RemoteDescriptor	string	Unused field for 5G-EPICENTRE (expected by the ELCM, so needs to be added with a default value of *null*).
ReservationTime	Array<float>	An array of two float values representing UNIX timestamps, indicating start and end times (duration) of the experiment.
Scenario	string	Scenarios or list of scenarios available at the platform, if this field contains more than one scenario the test cases will be executed consecutively in the different scenarios.
Slice	string	Slices or list of slices available at the platform, if this field contains more than one slice test cases will be executed consecutively in the different scenarios.
TestCases	Array<string>	TestCases that will be executed by the ELCM.
UEs	Array<string>	UEs involved in the experiment.
Version	string	Descriptor version. At present it is set by default to "2.1.0".

Table 3 lists an example of a descriptor, in order to offer a clear view of its usage. This descriptor belongs to an experiment in which OneSource’s (ONE) Mobitrust application is deployed at the UMA (Malaga) platform, and there is no automation (*i.e.*, real ends users will use the application directly). With the information included in the descriptor, the Experiment Coordinator accesses the Network Service Repository’s OpenAPI server, which retrieves the Helm chart files from the Helm repository (see D4.3). Once the Helm chart files have been obtained, the Experiment Coordinator makes use of the **Karmada federation** (see D4.5), to deploy the use case to the selected testbed. The configuration of the background traffic (Extra field, in the descriptor) is also provided. This information is redirected to the 5GTSM, in which the experiment will be executed (the example concerns UMA platform iPerf configurations). The rest of the information is delivered to the local ELCM. The identifiers generated by the Experiment Coordinator, associated to the execution of the experiment (*i.e.*, execution id, experiment id, iteration id, *etc*.) are sent to the Publisher.

Table 3: Example experiment descriptor.

```
{
    “Application”: “Mobitrust”,
    “Automated”: no,
    “ExclusiveExecution”: true,
    “ExperimentType”: “Custom”,
    “Extra”: {
        “Url”: “http://172.31.253.233:5003/start”,
        “ServerProbes”: [{
            “origin”: “UE”,
            “userId”: “00”,
            “experiment_id”: “0”,
            “publish”: true,
            “device_id”: “iperf_client_01”,
            “request_body”: {
                “agent_id”: “remote_iperf_1”,
                “action”: “Start”,
                “parameters”: {
                    “-s”: “”,
                    “-u”: “”,
                    “-p”: 6010,
                }
            }
        }],
        “ClientProbes”: [{
            “origin”: “UE”,
            “userId”: “01”,
            “experiment_id”: “0”,
            “publish”: true,
            “device_id”: “iperf_client_02”,
            “request_body”: {
                “agent_id”: “remote_iperf_2”,
                “action”: “Start”,
                “parameters”: {
                    “-c”: “remote_iperf_1”,
                    “-p”: 6010,
                    “-u”: “”,
                    “-b”: “150M”,
                    “-t”: 300,
                    “-i”: 1,
                }
            }
        }],
```

```
    },
    "NSs": [],
    "Parameters": {
        "Action": "deploy",
        "Filename": "mobitrust.zip",
        "Testbed_id": 2,
        "Use_case_id": 0,
        "Namespace": "uma",
        "Netapp_id": "testing",
        "HSPF": "no",
        "HSPF_microservices": [],
        "HSPF_ env_vars": [],
    },
    "Remote": null,
    "RemoteDescriptor": null,
    "ReservationTime": [
        1688036400000,
        1702984381000,
    ],
    "Scenario": "Ideal",
    "Slice": "Default",
    "TestCases": [
        "Helm Agent"
    ],
    "UEs": [],
    "Version": "2.1.0",
}
```

Figure 2 also includes an overview of the information interchanged between the different components of the Experiment Coordinator, related with the execution of the experiments (default scenario is chosen, and default slice is used in this indicative example).

2.1 Design

The design of the Experiment Coordinator is based on the ELCM. The Experiment Coordinator expands the functionality offered by the ELCM to support the federation of different experimentation platforms under the same umbrella. Figure 2 provides an overview of the different modules that compose the Experiment Coordinator. The Experiment Coordinator offers a *northbound interface*, for experimenters to support experiment handling and metadata configuration. The *administration interface* provides internal access to the management of the tasks under execution. The *scheduler* enables the planning of the experiments. Finally, the *execution engine* is the core component of the Experiment Coordinator that supports the supervision of the execution of the experiments at the different platforms.

2.1.1 Scheduler

The Scheduler is the component that manages the execution of the experiments at the **Stage level**. The stages defined are:

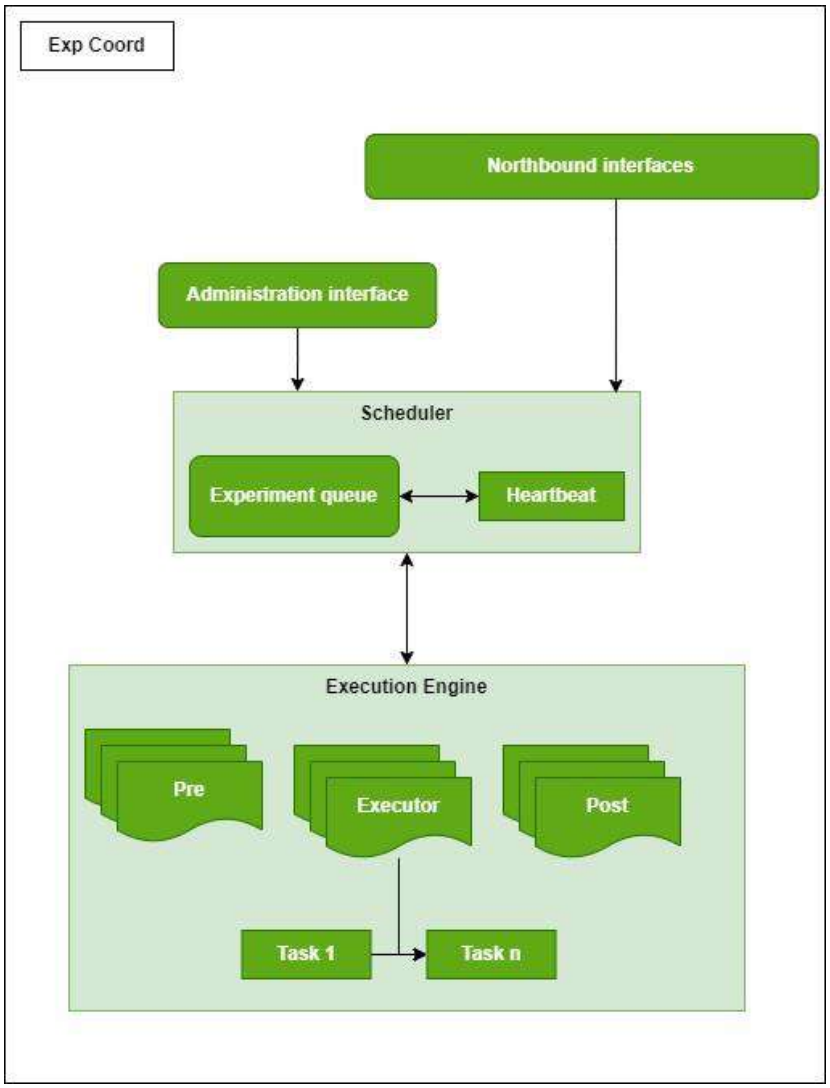

Figure 2: Experiment coordinator internal architecture

- **Pre-Run:** This stage includes the registration of the deployment request and configuration. A feasibility study check is also performed and a waiting loop is created for the required resources.
- **Run:** This stage is where the different tasks involved in the experiment are executed. It is in this stage where the communication with the 5GTSM takes place, and the deployment is performed in the indicated testbed. It is also where the Holistic Security and Privacy Framework (HSPF) network application is included (see D2.7 for more information) - if it has been requested.
- **Post-Run:** During this stage, the resources used by the experiment are released.

All experiments are kept in an internal execution queue, where they transition from one stage to another. When an experiment enters one of the execution stages, it is handled by an independent **Executor**, which will execute tasks one after another until they are completed, or an error is detected. The Experiment Coordinator can respond to events in an asynchronous manner, by receiving them through a REST API. However, the transition between the different stages of an experiment execution is coordinated by a background thread know as **Heartbeat**. This thread periodically triggers a status check of each execution from active experiments (*i.e.*, on any of the Run stages), triggering the transition to the next stage when an Executor has finished, or making them as erroneous, or user-cancelled, when necessary. The use of the Heartbeat thread provides a good balance between performance and parallel execution of different experiments.

2.1.2 Resource availability

One of the tasks of the Pre-Run stage is to check if it is feasible to run the experiment. If it is not feasible, a passive wait will be performed, until resources are released. As can be seen in Figure 3, the Experiment Coordinator ensures that each experiment can be safely executed on the platform. It is also capable of handling the execution of experiments with a **first come-first-served** system, which means that when two experiments are waiting for the same resource, the Experiment Coordinator will give priority to the one that has been waiting the longest.

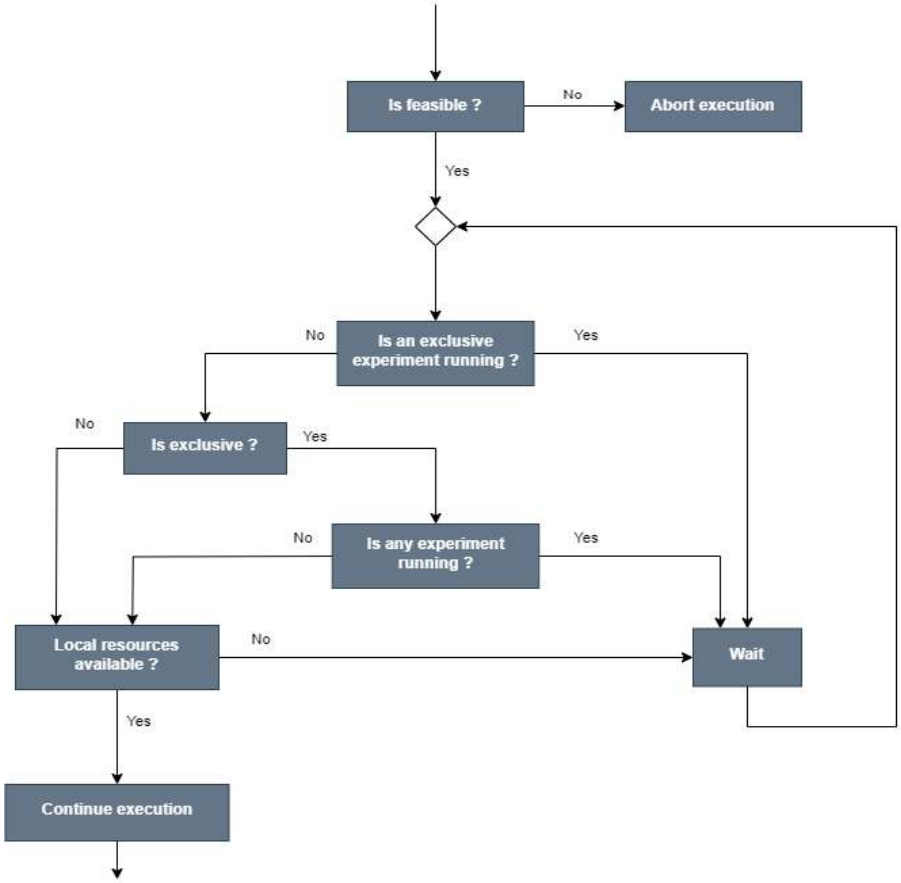

Figure 3: Flowchart on the feasibility of running experiments.

2.1.3 Experiment configuration

When deploying the Experiment Coordinator, it is necessary to establish certain configurations for correct operation. These configurations can be established in the **config.yml** file, created by the application when it is instantiated for this purpose. The application will use this file to read (among other things) the credentials established for the different elements. Some of the most important credentials that may be established are the following:

- **OpenAPI:** The internal Network Service Repository module in charge of the communication with the Helm repository. The established credentials allow access to it. Both this module and the repository are explained in more detail in D4.3.
- **RabbitMQ:** By establishing the credentials to access the RabbitMQ queue, the Experiment Coordinator can publish a message in the queue, indicating the beginning and end of an experiment identified with its metadata.
- **Publisher:** With the publisher credentials, the Experiment Coordinator can change the experiment id of the messages published by an experiment in execution, in order to facilitate its identification.

2.2 Implementation

This Section will describe details about the logic and implementation of the Experiment Coordinator, from reception of the descriptor, to the deployment and execution of the experiments. The implementation is done using classes according to the object-oriented programming model with the Python programming language.

2.2.1 *ExperimentRun* class

This class is responsible for storing updated information about the execution of an experiment. It also contains information about the Pre-run, Run and Post-run stages of the experiment. When a request is received, the **Scheduler** will create a new instance of the *ExperimentRun* class, identified by a unique id. As part of the parameters required for the creation of the *ExperimentRun* class, it is necessary to provide all the configuration data involved in the experiment. This data known as **Params** will be shared with the *PreRunner*, *Executor* and *PostRunner* classes, which are the classes in charge of the stages with the same names. Thanks to this parameter sharing, the communication between the different stages is facilitated. Two of the most important values of the 'Params' variable, are the **Descriptor**, which is received from the Portal, and contains relevant information about the experiment deployment, 5G traffic generation and experiment identification metadata; and the **configuration**, which is obtained during the different phases of the experiment.

The minimum logic for executing threads in parallel is found in the **Child** class. This class includes functionalities for the management of the different threads, where the execution of methods is possible, as well as the creation - and destruction of temporary folders.

2.2.2 ExecutorBase class

The **ExecutorBase** class extends the Child class. This class provides extra functionality that is in common between all execution stages (Pre-run, Run and Post-run). It includes information about when the Executor was created, when it was stated and finished, and the list of messages that have been generated. These messages are stored separately from the logs and are an efficient method to follow the execution process. For example, a new message will be generated when the Executor starts processing a new task.

All stages of execution perform a series of **Tasks**. In the case of the Pre-run and Post-run stages, this list of tasks is static and the same for all experiments. The tasks executed by the Executor are grouped in 'Test Cases' and are accessible through the 'RunTask' variable.

2.2.3 Tasks

A task is the minimum action that must be performed to deploy and execute an experiment. It is possible to delegate its execution to an external entity. For example, a task can be used to execute a script in OpenTAP, that will take some measurements, executing the script through commands by which parameters are passed. The task must be executed for as long as necessary, but only one task can be executed at a time.

Like the *ExperimentRun* class, *Tasks* receive the list of necessary parameters obtained from the Descriptor. These parameters define the behaviour of the task and can assume a condition to execute a given task or not. Some examples of tasks defined in the Experiment Coordinator for the deployment and execution of experiments are:

- **Run.CompressFile:** Generates a Zip file containing all the specified files and folders.
- **Run.TrafficSimulator:** Defines the metadata and initiates traffic generation on the selected remote iPerf agents through the corresponding 5GTSM.
- **Run.CliExecute:** Executes a script or command through the command line.
- **Run.DeployExperiment:** Performs the deployment of an experiment defined in the Portal. It publishes start and end of experiment execution messages and updates the Publisher with the generated execution id.

2.3 Northbound and Southbound interfaces

This Section will show the relevant interfaces offered by the different, final (*i.e.*, superseding prior deliverables) endpoints of the Experiment Coordinator.

2.3.1 Experiment Run

Endpoint used by the Portal for requesting the execution of experiments (Table 4). The content of this request includes the Descriptor file content (see also Table 2), with the necessary information for the deployment and execution of the experiment.

Table 4: API endpoint used to request the start the execution of tasks.

Run API	
Method	POST
Endpoint	/api/v0/run
Request Headers	None
Request Body	See Table 2.
Response	**200** The response contains the execution Id in the format: {“ExectionId": int}
	404 Indicates that the connection to the Experiment Coordinator has not been possible.
	500 Indicates that there was an error in the input format.

2.3.2 Experiment Cancel

Endpoint for cancelling the execution of an experiment identified by an execution Id (Table 5).

Table 5: API endpoint for cancelling the execution of tasks in the Experiment Coordinator.

Cancel API	
Method	GET
Endpoint	/execution/<id>/cancel
Request Headers	None
Request Body	None
Response	**200** This code indicates that the experiment has been successfully cancelled.
	404 Indicates that the connection to the Experiment Coordinator has not been possible.
	500 Indicates that there was an error in the input format.

2.3.3 Experiment Descriptor

Endpoint for returning the Descriptor file associated with an experiment identified by the given execution id (Table 6).

Table 6: API endpoint used to obtain the descriptor file associated with an experiment.

Descriptor API	
Method	GET
Endpoint	/execution/<id>/descriptor
Request Headers	None
Request Body	None
Response	**200** Returns the experiment descriptor in JSON format.
	404 Indicates that the connection to the Experiment Coordinator has not been possible.
	500 Indicates that there was an error in the input format.

2.3.4 Experiment Logs

Endpoint for returning the logs associated with an experiment, identified by its execution Id (Table 7).

Table 7: API endpoint used to obtain the logs of all stages.

Logs API	
Method	GET
Endpoint	/execution/<id>/logs
Request Headers	None
Request Body	None

Response	**200** Returns the logs of the different stages in the format:
	{
	“Status”: str
	“PreRun”: <LogInfo>
	“Executor”: <LogInfo>
	“PostRun”: <LogInfo>
	}

	404 Indicates that the connection to the Experiment Coordinator has not been possible.
	500 Indicates that there was an error in the input format.

2.4 Deployment

This Section includes a guide for the deployment of the application on a K8s-based platform. For the containerization of the Experiment Coordinator, an example Dockerfile can be found in Table 8, that can be used to create the containers with Docker[4].

[4] https://www.docker.com/

Table 8: Dockerfile example for Docker container creation of the Experiment Coordinator.

```
FROM python:3.10.13-slim-bookworm
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 5001
CMD ["./start.sh"]
```

Once the container has been created, a YAML file should be generated for the creation of the deployment in K8s. An example of such a YAML file can be found in Table 9.

Table 9: Example YAML file of an Experiment Coordinator deployment in a K8s cluster.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: exp-coord
labels:
apps: exp-coord
spec:
replicas: 1
selector:
matchLabels:
app: exp-coord
template:
metadata:
labels:
app: exp-coord
spec:
containers:
- name: exp-coord
image: repository_name/5gepicentre:exp-coord
ports:
- containerPort: 5001
```

Finally, the deployment must be implemented using the **kubectl**[5] command line tool. This tool is used for the management and use of a cluster based on k8s.

[5] https://kubernetes.io/docs/reference/kubectl/

3 5G Traffic Simulator Manager

The 5G Traffic Simulator Manager (5GTSM) is the module in charge of traffic generation in the network at the time of running an experiment. Traffic generation is based on the **iPerf** [6], that will be executed by **remote iPerf agents**. Thanks to these elements, the 5GTSM can generate traffic between several agents acting as client/server pairs with the parameters required for the generation of the desired scenario.

3.1 Traffic generation

For traffic generation a POST request containing a JSON file with the necessary data for the configuration of the agents involved in traffic generation, is sent to a 5GTSM endpoint by the Experiment Coordinator (The 5GTSM is an element deployed in each of the testbeds).

The 5GTSM instance will compare the IDs of the agents received for traffic generation among those it has stored and will configure theses agents with the indicated parameters. Once the agents are configured, it will start traffic generation using the iPerf tool for the time indicated in the configuration. Figure 4 shows the workflow communication between the Experiment Coordinator and the traffic agents via the 5GTSM:

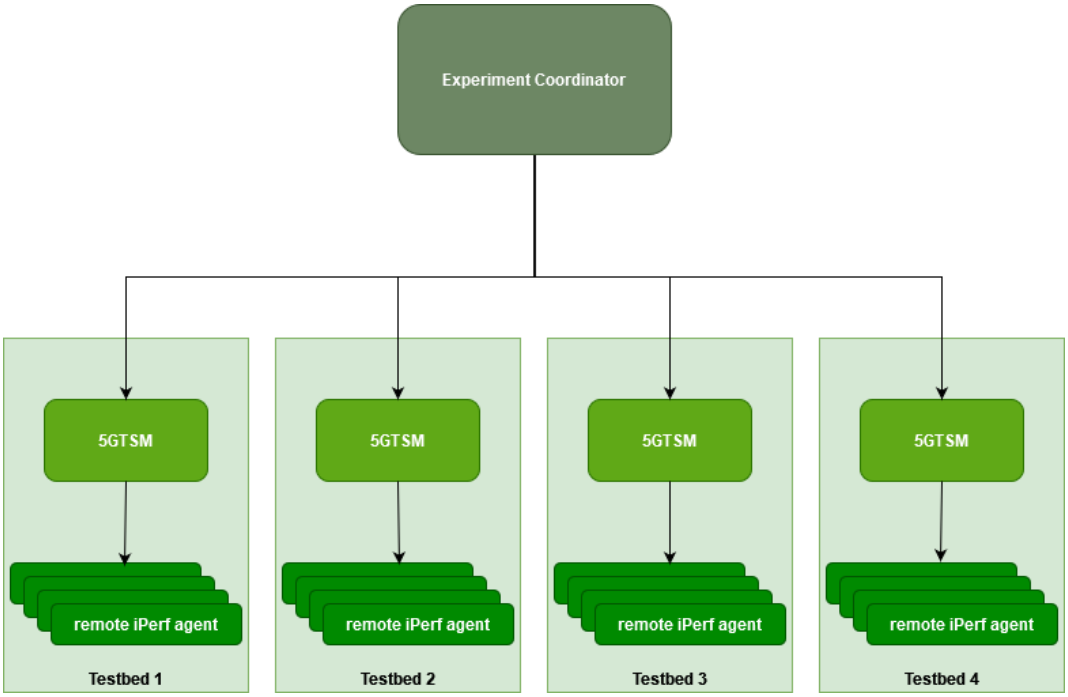

Figure 4: Life cycle of traffic generation

3.1.1 Remote iPerf Agents

The remote iPerf agent is the agent used to run the iPerf tool. This agent receives from the 5GTSM the configuration parameters and runs the iPerf tool. The results of the iPerf tool are captured by the same remote iPerf agent and published in the 5G-EPICENTRE MQTT queue in real time.

3.2 Implementation

Both the 5GTSM and the remote iPerf agents offer a REST API programmed in Python for their configuration and control.

[6] https://iperf.fr/

3.2.1 Implementation of the 5GTSM

The 5GTSM has several endpoints where it receives the instructions to be carried out. These are explained in more detail in Section 3.3. As a general overview of the behaviour of the traffic generation feature, the 5GTSM receives a JSON file via a POST request to the **/start** endpoint whose content is shown in Table 10. The data included in the JSON file is used for the configuration of the remote iPerf agents.

Table 10: Configuration of a remote iPerf agent

```
{
    "netapp_id": "String",
    "origin": "String",
    "UserId": "String",
    "experiment_id": "String",
    "publish": "boolean",
    "request_body":
        {
            "agent_id": "String",
            "action": "Start/Stop",
            "parameters": {
                "-s": "",
                "-u": "",
                ...
             }
        }
}
```

The file contains the metadata necessary to identify the experiment to which the traffic is applied. The *"request_body"* field contains the data to identify the agent and the *"parameters"* sub-field contains the necessary flags for the configuration of the iPerf tool.

Once the JSON file is received, it goes through a verification, to check that it is in the correct format. If the verification is successful, the 5GTSM checks that it has stored the value assigned to the "*agent_id"* sub-field in its JSON file for that purpose. If the agent_id exists in that file, the 5GTSM will send the configuration to the corresponding remote iPerf agent.

3.2.2 Implementation of the Remote iPerf agents

A remote iPerf agent is in charge of both the generation of traffic through the iPerf tool, and the publication of the results obtained by this tool in an MQTT queue.

The remote iPerf agent supports both iPerf 2.x and 3.x versions, as well as Linux and Windows operating systems. To specify the version of the iPerf tool to be used, as well as establish the credentials of the MQTT queue where the results are to be published, there is a configuration YAML file ("*config.yml"*) that must be configured before the execution of the remote iPerf agents.

For the execution of the iPerf tool, the agents use the Python library ***subprocess***[7], which in turn contains the ***popen*** library. Thanks to this library it is possible to execute commands from the Python application with the desired parameters. The execution of such a command will be done asynchronously using execution threads.

Once the iPerf tool is running, regular expressions are used to read the output data from the iPerf tool. This data will be published in the corresponding MQTT queue using the Python ***paho***[8] library. Table 11 shows an example of the messages that the remote iPerf agent publishes in the MQTT queue, when it is generating traffic.

Table 11: Example of the results published by the remote iPerf agent in the MQTT queue

```
{
    "experiment_id": "01",
    "testbed_id": "1",
    "scenario_id": "scenario",
    "use_case_id": "use case",
    "netapp_id": "opentap_uma",
    "category": "experiment",
    "data": [
        {
            "type": "throughput",
            "unit": "Mbps",
            "origin": "UE",
            "timestamp": 1693335608373,
            "value": 101
        },
            "type": "jitter",
            "unit": "ms",
            "origin": "UE",
            "timestamp": 1693335608373,
            "value": 0.054
        },
        {
            "type": "packet loss",
            "unit": "%",
            "origin": "UE",
            "timestamp": 1693335608373,
            "value": 41
        },
        ...
```

3.3 Northbound and Southbound interfaces

This Section will show the interfaces offered by the different endpoints that are exposed by the 5GTSM.

[7] https://docs.python.org/3/library/subprocess.html
[8] https://eclipse.dev/paho/

3.3.1 Start API

An endpoint that executes an iPerf command with its parameters on the iPerf agent identified by ID in the request body (in JSON format). This is intended to be the normal way to start a server/client on a remote agent (Table 12).

Table 12: API endpoint to start the configuration of a remote iPerf agent.

Start API	
Method	POST
Endpoint	/start
Request Headers	None
Request Body	**netapp_id** [String] OPTIONAL OPTIONAL Id for netapp identification.
	origin [String] REQUIRED Origin of the measures. Can be one of {"UE", "RAN", "5GC", "EPC", "main data server", "edge"}.
	userId [String] REQUIRED User identification.
	experiment_id [Number] REQUIRED Experiment identifier provided by the Experiment Coordinator.
	publish [Boolean] REQUIRED Indicates whether the generated traffic measurements should be published in the RabbitMQ queue, or not.
	request_body [Object] REQUIRED The request body contains the data necessary to identify and configure the agents. **- agent_id** [String] REQUIRED Identifier of the agent involved in traffic generation. **- action** [String] REQUIRED Action to be performed by the agent. **- parameters** [Object] REQUIRED This section includes the agent configuration parameters based on the iPerf tool. There will be one entry for each configuration flag. A possible example of configuration parameters could be: "parameters": { "-c": "Probe2A", "-p": 6004,

	"-u": "", "-b": "150M", "-t": 1200, "-i": 1 }
Response	**200** The response will contain a JSON Object with metadata and iPerf parameters for traffic generation.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.3.2 Stop API

Endpoint for stopping an iPerf server running on a remote agent, identified by a dictionary with keyword "agent_id" and value "ID" (Table 13).

Table 13: API endpoint to stop the execution of a remote iPerf agent identified by agent_id

Stop API	
Method	POST
Endpoint	/stop
Request Headers	None
Request Body	**userId** [String] REQUIRED User identification.
	request_body [Object] REQUIRED The request body contains the necessary to identify the agent to be stopped. **- agent_id** [String] REQUIRED Identifier of the agent to be stopped. **- action** [String] REQUIRED Action to be performed by the agent.
Response	**200** The Agent specified by the "agent_id" keyword has stopped executing. The response will contain a JSON with the necessary information for the identification of the agent to be stopped.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.3.3 Add iPerf agent API

Endpoint for adding the ID of a remote agent. Identified by a given ID, IP and port (Table 14).

Table 14: API endpoint for adding a remote iPerf agent to a 5GTSM.

Add iPerf agent API	
Method	POST
Endpoint	/add_iperf_agent
Request Headers	Content-Type: application/x-www-form-urlenconded
Request Body	**agent_id** [String] REQUIRED Agent identification.
	agent_ip [String] REQUIRED Agent IP.
	agent_port [Integer] REQUIRED Agent port.
Response	**200** The Agent specified by the "agent_id" keyword has been added to the 5GTSM. The response will contain the word "Done!" to confirm that the operation was successful.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.3.4 Delete iPerf agent API

Endpoint for deleting the ID of a remote agent. Identified by a given ID (Table 15).

Table 15: API endpoint for deleting a remote iPerf agent.

Delete iPerf agent API	
Method	POST
Endpoint	/delete_iperf_agent
Request Headers	Content-Type: application/x-www-form-urlenconded
Request Body	**agent_id** [String] REQUIRED Agent identification.
Response	**200** The Agent specified by the "agent_id" keyword has been deleted to the 5GTSM. The response will contain the word "Done!" to confirm that the operation was successful.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.3.5 Retrieve API

Endpoint for retrieving the results of the previous execution (Table 16). Returns a JSON reporting the success of the retrieval, a message and list of Results (dictionary with parsed results).

Table 16: API endpoint to retrieve a dictionary containing the results of a given agent since its last execution.

Retrieve API	
Method	GET
Endpoint	/retrieve
Request Headers	
Request Body	**agent_id** [String] REQUIRED Agent identification.
	host[String] REQUIRED Identifies the agent's ip and port with the format, "ip:port".
Response	**200** The operation has been successful and returns a dictionary with the results set of the last execution of the dictionary.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.3.6 Retrieve probes API

Endpoint for retrieving a list of all agents in the 5GTSM (Table 17).

Table 17: API endpoint to retrieve a dictionary containing all the remote iPerf agent ids associated to a 5GTSM

Retrieve Probes API	
Method	GET
Endpoint	/retrieve_probes
Request Headers	None
Request Body	None
Response	**200** Return a list of all agents stored in the 5GTSM.
	404 Indicates that the connection to the 5GTSM has not been possible.
	500 Indicates that there was an error in the input format.

3.4 Deployment

This Section provides a guide for the deployment of the application on K8s-based platforms. Example Dockerfiles can be found in Table 18 and Table 19, that can be used to create the containers for the 5GTSM and the remote iPerf agents, respectively, with Docker.

Table 18: Dockerfile to containerize the 5GTSM in a Docker container.

```
1  FROM python:3.9-alpine
2  WORKDIR ./src
```

```
COPY requirements.txt .
RUN python -m venv venv
RUN venv/bin/pip install -r requirements.txt
COPY app app
COPY config config
COPY REST REST
COPY iperf-hosts.json .
COPY boot.sh ./
ENV  PYTHONPATH=${PYTHONPATH}:./src
RUN chmod +x boot.sh
EXPOSE 5003
ENTRYPOINT ["./boot.sh"]
```

Table 19: Dockerfile to containerize a remote iPerf agent in a Docker container.

```
FROM ubuntu:20.04
WORKDIR ./src
COPY requirements.txt .
RUN apt-get update
RUN apt-get install python3.9 -y
RUN apt install apt-utils -y
RUN apt-get install iperf -y
RUN apt-get install iperf3 -y
RUN apt-get install expect -y
RUN apt-get install python3-venv -y
RUN python3 -m venv venv
RUN venv/bin/pip install -r requirements.txt
COPY iperfExecutor iperfExecutor
COPY app.py ./
COPY boot.sh ./
COPY config.yml ./
ENV PYTHONPATH=${PYTHONPATH}:./src
RUN chmod +x boot.sh
EXPOSE 6006
EXPOSE 6005
ENTRYPOINT ["./boot.sh"]
CMD ["6005"]
```

Once the containers have been created, a YAML file should be generated for the deployment in K8s. Examples of such YAML files can be found in Table 20 and Table 21, below:

Table 20: YAML file of 5GTSM deployment in K8s cluster.

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
name: 5gtsm
labels:
app: 5gtsm
spec:
replicas: 1
selector:
matchLabels:
app: 5gtsm
template:
metadata:
labels:
app: 5gtsm
spec:
containers:
- name: 5gtsm
image: repository_name/5gepicentre:5gtsm
ports:
- containerPort: 5003
```

Table 21: YAML file of remote iPerf agents deployment in K8s cluster.

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: remote-iperf
labels:
app: remote-iperf
spec:
replicas: 1
selector:
matchLabels:
app: remote-iperf
template:
metadata:
labels:
app: remote-iperf
spec:
containers:
- name: remote-iperf
image: repository_name/5gepicentre:remote-iperf
ports:
- containerPort: 6006
```

Finally, the deployment must be implemented using the **kubectl** tool. For the communication between the different deployments it is possible to create **services**. These services will expose IPs for the different components that can be accessed between them. An example of a service for the 5GTSM is shown in Table 22.

Table 22: YAML file of a service for container communication in a K8s cluster.

```
apiVersion: apps/v1
kind: Service
metadata:
name: epicentre-service
spec:
selector:
spec:
app: 5gtsm
type: LoadBalancer
ports:
- protocol: TCP
port: 5003
targetPort: 5003
```

4 Publisher

The Publisher is the component in charge of supervising the interchanged messages for reporting the measurements, adding into these messages the metadata information required to link the measurements with the application under test, the testbed, the experiment and the specific execution of the experiment. The Publisher makes use of queues based on a **RabbitMQ Message Broker**, which in turn uses the Message Queuing Telemetry Transport (**MQTT**) protocol. These queues are classified by a *topic*, which is identified by a routing key. The Publisher's actions depend on subscribing to a queue that shares a common topic (*e.g.*, *topic*, with routing key *"application")* the application under test will report the measurement in this queue. The Publisher, which is subscribed to this queue, will read each message that passes through it, will check that the format is correct, and will fill in the necessary metadata for experiment identification. It will then publish the revised messages in a queue with different topic (routing key *"publisher"*), where the rest of the components of the 5G-EPICENTRE architecture can subscribe and read the messages in the correct format.

The Publisher includes an interface to Prometheus to retrieve the relevant information about the infrastructure. This information is also published into the *"publisher"* topic queue. The JSON format that the messages published in the RabbitMQ queue must follow, and that the Publisher must check and fill in with the relevant metadata, is shown in Table 23. Figure 5 provides an overview of the role of the Publisher in the 5G-EPICENTRE architecture.

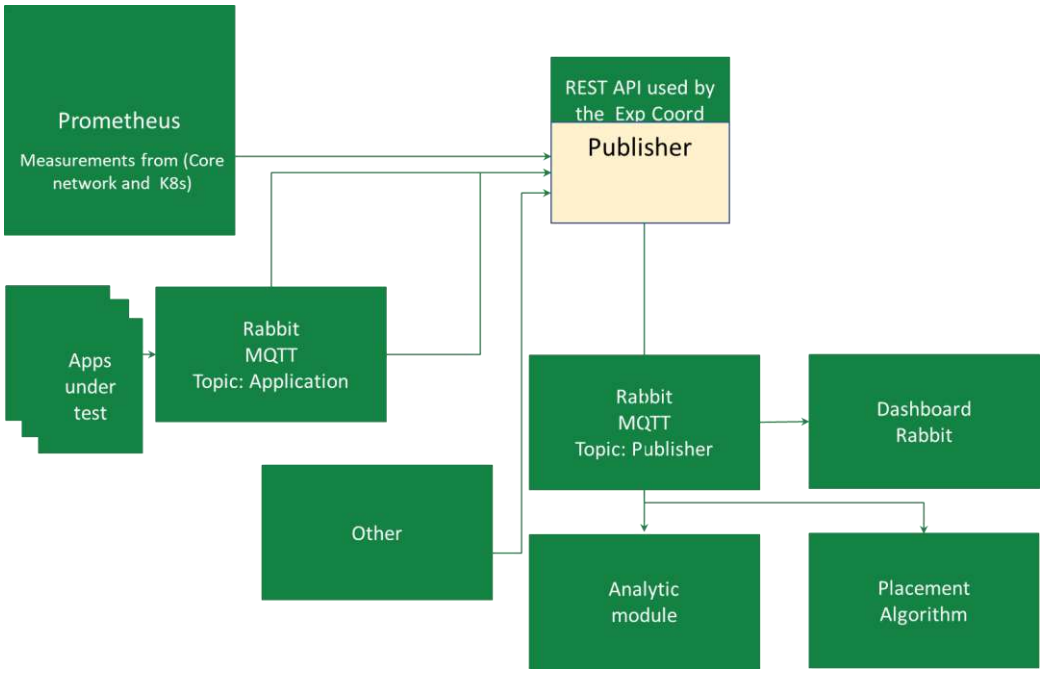

Figure 5: Publisher component in the context of the 5G-EPICENTRE architecture

Table 23: Publisher message format

```
{
    "$id": "https://example.com/arrays.schema.json",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "description": "The JSON schema for all kind of metrics",
    "category": ["5g_network", "nfv_mano", "vnf_chain", "experiment"], MANDATORY
    "testbed_id": ["integer"], MANDATORY, filled by the Testbed (UMA: 1, ALB: 2, CTTC: 3, HHI: 4)
    "scenario_id": ["integer"], OPTIONAL, filled by the Testbed
    "use_case_id": ["integer"], OPTIONAL, filled by the Testbed
```

```
    "experiment_id": ["integer"], OPTIONAL, filled by the Testbed
    "netapp_id": ["string"], OPTIONAL, filled by the Use Case APP
    "device_id": ["string"], OPTIONAL, filled by the Use Case APP
    "data": [  Array of data blocks, filled by actual data source (NetApp or Testbed)
        {
            "type": ["string"], MANDATORY
            "timestamp": ["long"], MANDATORY, as UTC POSIX timestamp, in milliseconds as a long
            "origin": ["UE", "RAN", "5GC", "EPC", "main data server", "edge"], MANDATORY
            "unit": ["gbps", "mbps", "kbps2, "bps2, "s", "ms", "us", "ordinal", "rate", "percent-age",
                     "interval"], OPTIONAL
            custom fields (key-value pairs as from native metrics):
            "param1": ["integer", "float", "string", "bool"],
            ...
            "paramN": ["integer", "float", "string", "bool"]
            "device_id": ["string"]
        }
        ...
    ]
}
```

The Publisher's metadata completion process can be seen in Figure 6.

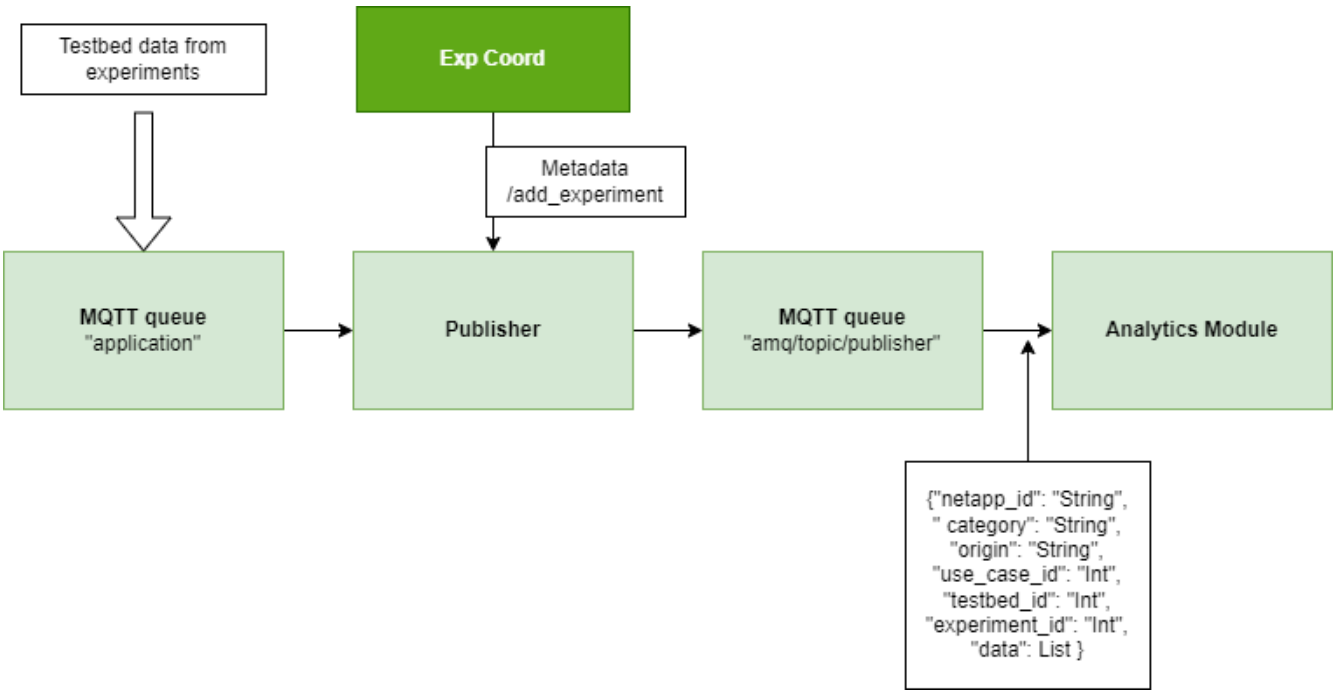

Figure 6: Publisher metadata completion process.

The process begins with the publication of data by the testbeds on which the experiments are being ran. The Publisher will passively "listen" to the data in the queue with topic *"application"*. The Publisher recognizes each published message. The format that the messages must follow is the format that is compatible with the Analytics Engine, so that measurements can be processed correctly.

Some considerations apply when filling in the metadata of the messages. The Publisher has metadata stored for the different use cases (in the *rsc/experiment.json* file). If the *"category"* field of the message contains the value *"experiment",* the value of the *"netapp_id"* field will be considered to identify the experiment and fill it with the metadata stored. If the *"category"* field does not have a value other than *"experiment"*, or if the *"netapp_id"* value is not stored in the Publisher, then the message will be filled with the standard metadata set in the *DefaultConfig* file. Any message accepted by the Publisher will be automatically published in the queue identified with the routing key *"amq/topic/publisher"*.

The Publisher will also connect to any Prometheus instance that has been configured and monitor the data periodically (see D1.4). The measurements collected from the Prometheus instance will be published in the RabbitMQ queue with routing key *"amq/topic/publisher"* for further analysis.

4.1 Implementation

The Publisher implementation is based on a REST API developed in Python. A **Flask**[9] server has been created for this purpose and is used for its execution.

4.1.1 Monitoring to the MQTT queue

To monitor and publish messages from the RabbitMQ queue that are published by the testbeds, it is first necessary to configure access to the queue. For this purpose the *DefaultConfig* configuration file in the *Helper* directory is used. In the file, in addition to the default messages related to the experiments, there are the credentials for access to the RabbitMQ queue, and to the relevant Prometheus instance.

The *Listener* class is used to monitor the messages of the RabbitMQ queue. This class makes use of the **Paho** library created for RabbitMQ queue management. Thanks to this library, it is possible to make use of the different callback functions, that are launched when a message is published in the queue.

4.1.2 Publication of messages in the MQTT queue

For the publication of processed messages from the RabbitMQ queue, certain considerations apply. The rules for publishing messages are included in the **Publisher** class. Rules to be considered when filling in the message metadata are defined by the *"category"* field of the message. If the value of this field is *"experiment"*, then it is mandatory to indicate the *"netapp_id"* field and value in order to identify the predefined experiment in the *rsc/experiment.json* file (as described above). In case the value of *"category"* is *"nfv_mano"* or *"vnf_chan"*, then the experiment metadata is defined in the message itself. Finally, if the value is *"5g_network"*, then it is not mandatory to indicate the metadata.

For the publication of messages, an execution thread is used for each message which will allow several messages to be published simultaneously while continuing to monitor the queue for new messages.

4.1.3 Fetching of Prometheus measurements

The Prometheus instance data collection is configured in the *DefaultConfig* file. In this file the queries made to this instance are defined through calls to a REST API that the instances exposes. These measurements received from the Prometheus instance do not fit within a single message but the Publisher can detect this problem and publish the measurements in multiple messages.

Once the Prometheus instance has been configured in the Publisher, it will query its measurements periodically at an interval configured in the *DefaultConfig* file.

[9] https://flask.palletsprojects.com/en/3.0.x/

4.2 Northbound and Southbound interfaces

This Section will show the interfaces offered by the different endpoints exposed by the Publisher.

4.2.1 Add experiment API

This endpoint is used to send a JSON containing the metadata of a new experiment (Table 24). This metadata is stored by the Publisher. When the corresponding experiment results are received at the MQTT queue, they will be filled with this metadata.

Table 24: API endpoint for adding experiments to the Publisher

Add experiment API	
Method	POST
Endpoint	/add_experiment
Request Headers	None
Request Body	**netapp_id** [String] REQUIRED Required Id for netapp identification.
	origin [String] REQUIRED Origin of the measures. Can be one of {"UE", "RAN", "5GC", "EPC", "main data server", "edge"}.
	use_case_id [String] REQUIRED User identification.
	experiment_id [Number] REQUIRED Experiment identifier provided by the Experiment Coordinator.
	testbed_id [Number] REQUIRED Identifier of the testbed to which the experiment belongs.
	category [String] REQUIRED Category of messages, may be one between [“5g_network”, “nfv_mano”, “vnf_chain”, “experiment”]
Response	**200** The response contains a message indicating that the experiment has been added.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.2.2 Remove experiment API

This endpoint is used to remove an experiment from the list of stored experiments (Table 25).

Table 25: API endpoint for removing experiments to the Publisher

Remove experiment API	
Method	POST
Endpoint	/remove_experiment
Request Headers	None
Request Body	**netapp_id** [String] REQUIRED Required Id for network application identification.
Response	**200** The response contains a message indicating that the experiment has been removed.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.2.3 Publish API

This API endpoint expects a message containing all the mandatory fields and publishes it on the RabbitMQ queue. This endpoint is meant to be used to publish experiment results, and expects messages to be fully formatted, unless the experiment has been previously added by the Experiment Coordinator through the **/add_experient** endpoint (Table 26).

Table 26: Access point for publishing in the queue

Publish API	
Method	POST
Endpoint	/publish
Request Headers	None
Request Body	**netapp_id** [String] REQUIRED Required Id for network application identification.
	use_case_id [String] REQUIRED User identification.
	experiment_id [Number] REQUIRED Experiment identifier.
	testbed_id [Number] REQUIRED Identifier of the testbed to which the experiment belongs.
	category [String] REQUIRED Category of messages, may be one between [“5g_network”, “nfv_mano”, “vnf_chain”, “experiment”]
	Data [List]

	- **type** [String] REQUIRED - **timestamp** [Long] REQUIRED - **origin** [UE, RAN, 5GC, EPC, main data server, edge] REQUIRED - **unit** [gbps, mbps, kbps2, bps2, s, ms, us, ordinal, rate, percentage, interval] OPTIONAL - **paramN** [Int,Floar, String, Bool] OPTIONAL - **…** - **device_id** [String] the identification of the device.
Response	**200** The response contains a message indicating that the message has been published.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.2.4 Fetch metrics API

This endpoint starts fetching new metrics from the Prometheus instances (if they are available), until the time specified (Table 27).

Table 27: Endpoint for metrics fetch

Fetch metrics API	
Method	POST
Endpoint	/fetch_metrics
Request Headers	None
Request Body	**metrics** [List] REQUIRED Information about the metrics to be fetched for. - **query** [String] REQUIRED Name of the query to be performed. It must be a valid search as indicated in the documentation - **unit** [String] OPTIONAL Name of the unit - **origin** [UE, RAN, 5GC, EPC, main data server, edge] REQUIRED **end_time** [Time when the search ends. Expressed in Hour, Minute, Seconds] **interval** [Integer] REQUIRED Interval of metrics in seconds.
Response	**200** The response contains a message indicating that the fetch has been successful.
	404 Indicates that the connection to the Publisher has not been possible.
	500 Indicates that there was an error in the input format.

4.3 Deployment

This Section provides a guide for the deployment on a K8s-based platform. For the containerization of the Publisher, an example Dockerfile can be found in Table 28, that can be used to create the containers with Docker.

Table 28: Dockerfile to containerise the Publisher

```
FROM python:3.10.13-slim-bookworm
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN pip install wheel
RUN pip install -r requirements.txt
RUN chmod +x start.sh
EXPOSE 5050
ENTRYPOINT ["./start.sh"]
CMD ["5050”]
```

Once the containers have been created, a YAML file should be created for the creation of the deployment in K8s. Such an example YAML file can be found in Table 29.

Table 29: YAML file for the creation of the publisher deployment on a K8s platform

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: publisher
labels:
app: publisher
spec:
replicas: 1
selector:
matchLabels:
app: publisher
template:
metadata:
labels:
app: publisher
spec:
containers:
- name: publisher
image: repository_name/5gepicentre:publisher
ports:
- containerPort: 5050
```

Finally, the deployment must be implemented using the **kubectl** command line tool. This tool is used for the management and use of a cluster based on k8s.

5 Conclusion

This document is related to Task T2.4 “Experiment coordination and lifecycle management”. This deliverable presented details regarding the experiment coordination in the context of the 5G-EPICENTRE architecture.

The document described the experimentation methodology supported by the 5G-EPICENTRE architecture, and provided details regarding the implementation and deployment of the components involved in the execution of the experiment at the backend layer and at the testbed level. A special emphasis has been given to interfaces and deployment description, in order to provide a deep understanding of the experimentation methodology, and to ensure a successful adoption by new testbeds during the project exploitation phase.

The experimentation methodology adopted by the 5G-EPICENTRE project provides a flexible environment for executing experiments, supporting repeatable configurations and fully automated executions based on test cases, or experimentation definitions created with real users.

References

[1] García B., Gallardo, M. M., Merino, P., Eichhorn, F. & Koumaras, H. (2021, March). *Deliverable D3.16 Experiment Lifecycle Manager (Release B)*. 5GENESIS Project. Available at: https://5genesis.eu/wp-content/uploads/2021/04/5GENESIS_D3.16_V1.0.pdf