HORIZON 2020
Information and Communication Technologies
Integrating experiments and facilities in FIRE+

Deliverable D3.2
Performance Test Tools 1st Iteration

Grant Agreement number:	**687884**
Project acronym:	**F-Interop**
Project title:	**FIRE+ online interoperability and performance test tools to support emerging technologies from research to standardization and market launch** **The standards and innovations accelerating tool**
Type of action:	**Research and Innovation Action (RIA)**
Project website address:	**www.finterop.eu**
Due date of deliverable:	**xxx**
Dissemination level:	**PU/CO**

This deliverable has been written in the context of the Horizon 2020 European research project F-Interop, which is supported by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.

Document properties

Responsible partner	EANTC AG
Author(s)/editor(s)	Eduard Bröse, Chika Ngwu
Version	0.2
Keywords	Test tool, Performance

Abstract

The deliverable D3.2 describes the current status of the Performance Testing Tool for the F-Interop Platform. The tool is developed within WP3 as the goal of the Task T3.1.

The Performance Testing Tool provides means to simulate a large number of CoAP-enabled IoT devices and generate traffic load in accordance with the desired profile. The tool also provides means to generate impairments on the test traffic and so simulate unfavourable network conditions. Further, the tool provides an estimation of power consumption based on the traffic volume.

The Deliverable contains detailed description of the Performance Testing Tool principles, design and documentation of its modules. Further we describe how the tool integrates with the F-Interop's framework and provide an overview of the instantiation and configuration processes occurring during a test session.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ABC	Attribute Based Credential
CA	Consortium Agreement
CoAP	Constrained Application Protocol
ComSoc	Communications Society
DESCA	Development of a Simplified Consortium Agreement
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Tables
DNS	Domain Name System
DNSSec	Domain Name System Security Extensions
DPA	Data Protection Authorities
DPO	Data Protection Officer
EC	European Commission
ENISA	European Union Agency for Network and Information Security
ETSI	European Telecommunications Standards Institute
EU	European Union
FP7	Seventh Framework Programme
GA	Grand Agreement
GA	General Assembly
GPS	Global Positioning System
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communication Technologies
ID	Identifier
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IERC	European Research Cluster on the Internet of Things
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Intellectual Property Committee
IPM	IPR Monitoring and Exploitation Manager
IPR	Intellectual Property Rights
IPSEC	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standards Organization
ISP	Internet Service Provider
IT	Information Technology
ITU	International Telecommunication Union
KPI	Key Performance Indicator
LSPI	Legal, Security and Privacy Issues
MAC	Media Access Control
MSc	Master of Science
M2M	Machine to Machine
OASIS	Organization for the Advancement of Structured Information Standards
OECD	Organization for Economic Cooperation and Development

OS	Operating System
OSN	Online Social Network
PC	Project Coordinator
PCP	Partner Contact Person
PDPO	Personal Data Protection Officer
PERT	Program Evaluation Review Technique
PhD	Doctor of Philosophy
PM	Person Month
PMB	Project Management Board
PPR	Periodic Progress Report
PRAAT	Privacy Risk Area Assessment Tool
P&T	Post & Telecom
QoS	Quality of Service
RAND	Reasonable and Non Discriminatory
RFC	Request For Comments
R&D	Research & Development
SME	Small Medium Enterprise
SMS	Short Message Service
SO	Session Orchestrator
SOTA (or SoA)	State Of the Art
SSL	Secure Sockets Layer
TC	Technical Coordinator
TCP	Transmission Control Protocol
TL	Task Leader
TLS	Transport Layer Security
Tor	The Onion Router
TRL	Technology Readiness Level
UK	United Kingdoms
UN	United Nations
UNCTAD	United Nations Conference on Trade and Development
UPRAAT	Universal Privacy Risk Area Assessment Tool
URL	Uniform Resource Locator
US	United States
VoIP	Voice over Internet Protocol
WES	Women's Engineering Society
WiTEC	Women in science, Engineering and Technology
WoT	Web of Trust
WP	Work Package
WPL	Work Package Leader
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

1.1 About F-Interop

F-Interop is a Horizon 2020 European Research project, which proposes to extend the European research infrastructure (FIRE+) with online and remote interoperability and performance test tools supporting emerging technologies from research to standardization and to market launch. The outcome will be a set of tools enabling:

- Standardization communities to save time and resources, to be more inclusive with partners who cannot afford travelling, and to accelerate standardization processes;
- SMEs and companies to develop standards-based interoperable products with a shorter time-to-market and significantly lowered engineering and financial overhead.

F-Interop intends to position FIRE+ as an accelerator for new standards and innovations.

1.2 Deliverable Objectives

1.2.1 Work package Objectives

- Research and develop performance test tools.
- Research and develop tools for privacy risk assessment
- Research and develop spatial representing tools to support experimenters.
- Research and integrate testing tools with network virtualization technologies such as OpenFlow / OpenDayLight based SDN / NFV environments.

1.2.2 Task Objectives

Work: The main goal of this task is to provide the required software to enable remote online testing with respect to QoS, scalability and energy. This work is based on the results of WP1 T1.1 “Testing tools requirements and analysis”. The tools will enable to test and measure:

- The scalability of the implementation under test (IUT), by analysing its behaviour and reaction time under a growing rate of requests and/or growing number of connected nodes.
- The effect of impaired network conditions on the stability, scalability and response time of the IUT
- The energy consumption impact of the tested device under given traffic load.

For QoS and performance testing in SDN/NFV networks / testbeds, the particular goals are a detailed analysis of performance and QoS parameter in OpenFlow v1.3 (e.g. SLA parameters, load/ delay/jitter control), the integration of the module in the OpenDayLight network controller, and the definition and implementation of a northbound API to the network controller

Roles: EANTC AG will lead the task with the support of UL and DG.

Outcome: A tool for assessing the performance and scalability of the IUT.

1.2.3 Deliverable Objectives and Methodology

1.2.3.1 Deliverable Objectives

The deliverable D3.2 – Performance Testing tools 1st iteration is the description of the first release of the Performance Test tool as output of the Task 3.1. This report contains the first version of the F-interop Performance Test Tool and the required key enablers needed.

1.2.3.2 Deliverables Methodology

One of the key features of the F-Interop Platform is to enable Remote and Online communications among IoT components. In some scenarios, what F-Interop called Location Models, the public Internet will be used to exchange data. Therefore, it is essential to avoid sharing information that may be confidential or sensitive for the owner of connected resources (e.g., individual, company,

organization). There is a need to protect any traffic exchanged and to provide an acceptable level of security and privacy.

On the other hands, users may have different view of privacy and security meanings and they might want to carefully check what will be shared during a test session. Hence, the platform should provide tools to check if a certain privacy and security level is satisfied.

Following the guidelines defined in Deliverable D1.2 (Privacy by Design Report), a state of the art study regarding IoT privacy issues has been conducted. Based on that, we selected one of the emerging IoT protocols, CoAP to investigate vulnerabilities during a session of data exchange. As output of this first iteration, the Privacy Test Tool has been developed which implements algorithms to "flexible" detect privacy issues.

The Privacy Test Tool follows the generic F-INTEROP Test Tool Design, a common infrastructure used to encapsulate task/algorithms within the F-Interop platform. Such design is based on the concept of "F-Interop session". This is a set of actions that any user (called F-Interop-User) has to perform in order to execute Remote or Online tests in the F-Interop-Platform.

These steps are summarized in the Table 1 of the deliverable D1.1 which is attached below.

Table 1 - F-Interop Session

Step	Action	Description
0	FI-User authentication and authorization. IUT registration/identification	FI-User authenticates in a secure way (prior FI-User registration needed) in FI-Platform. FI-User needs to be authorized to use FI-Platform resources. FI-User identifies which IUT he/she will test (prior IUT registration needed).
1	Test suites discovery and selection	FI-User starts by discovering the available test suites and by selecting the one he/she wants to execute
2	Resource description	FI-User specifies/selects resources in the F-Interop Platform that are needed for his/her F-Interop session including the location models 1, testing tools, libraries, etc. During this phase FI-Platform may request information from FI-User or provide information to FI-User for a coherent selection of the required resources
3	Resource reservation	The resources selected in the previous step are actually reserved.
4	**Resource provisioning, configuration and session setup**	The instantiation of the F-Interop-Platform resources that fit best with the FI-User needs is done.
5	**Test execution**	The online F-Interop test campaign is launched and the selected (executable) test suites are executed against the IUTs.
6	Results analysis and report	Test execution information is analysed. The test results and verdicts are provided together with explanations in case of FAIL or INCONCLUSIVE verdicts or something wrong happened. A report can be provided under request in case for example the FI-User wants to apply for a certification/labelling program.
7	Session storage	Storage of the F-Interop session information (Session-id, User-id, FI-User's IUT-id, IUTs' version, test description, test version, testing tool, test log and results, etc.). This has to remain accessible beyond the F-Interop session for the involved parties

This document focuses specifically on the **Step 4 (Resource provisioning, configuration and session setup)** and **Step 5 (Test Execution)** of an F-Interop Session.

The Testing tool architecture has been discussed in details during the WP/integration meetings allowing agreement on several points including the initial implementation design and the interfaces/APIs/standards to be used among the various components.

The task is working on a modular and as generic as possible architecture, to easily extends the solution towards new protocols, new matching criteria and user customization and requirements.

As a consortium decision, the AMQP (Advanced Message Queuing Protocol) standard has being adopted as a technological driver to easily deliver data to any location. We selected RabbitMQ as a technical solution for that.

RabbitMQ is a messaging tool that implements AMQP and it enables differentiated ACL security/authorization profiles. AMQP bindings are available in several languages. They can be used for remote interaction with the IUTs. Moreover, orchestration and distinct channels can be used as solution for create isolation among different testing tools.

Figure 1 describes the complexities of doing test in an online and remote manner.

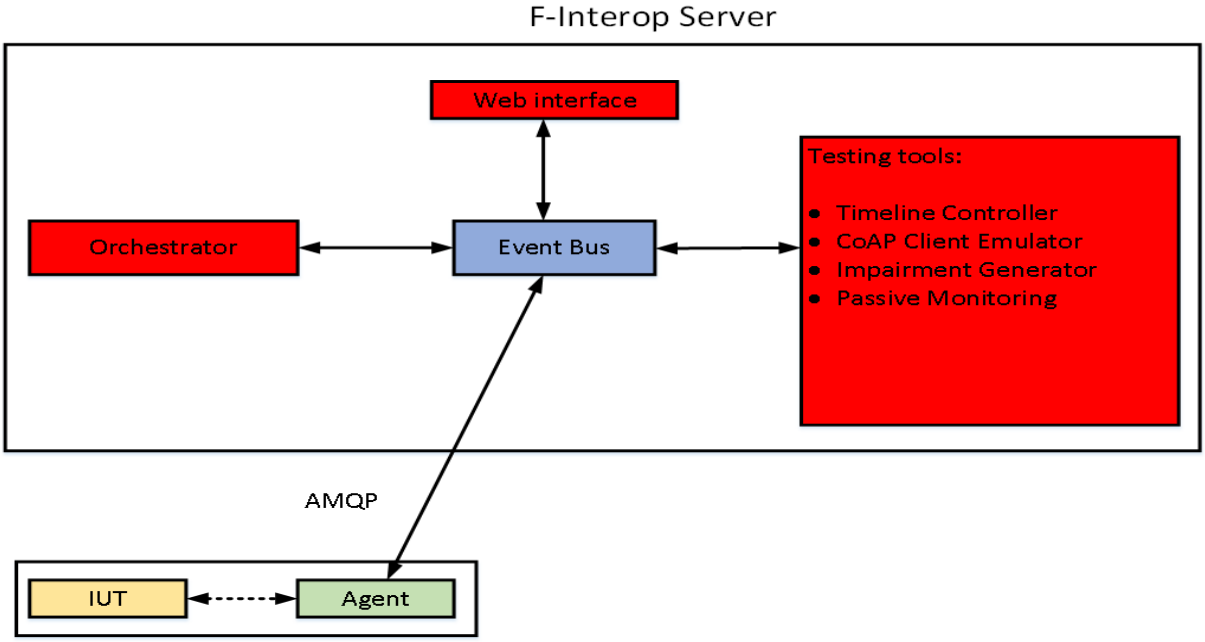

Figure 1 - F-Interop remote testing architecture

The following chapters aim to describe the architecture of the testing tool, its components and other auxiliary configuration used to execute a performance test.

2 Performance Test Tools Design

In the current implementation of the performance testing tool, we consider on only simple CoAP protocol emulation. The framework however is designed to make possible to easily integrate other protocols and/or functionalities in a generic way. The following sections describe the generic design of the framework and the functional modules of the performance testing tool (further: "subcomponents") without going into specific details of the functions provided by each module. This information is presented in detail in the next chapter 3 "Performance Test Modules Functionality."

2.1 Performance Test Tool Design

2.1.1 General Principles

The general goal of the performance test tool is to provide simulation of multiple simultaneously operating nodes (e.g. emulated CoAP clients), and generate traffic volume that would have detectable effect on the DUT when it reaches its performance limits.

In contrast to the interoperability and conformance testing, the test process is:

- Non-deterministic: the success of the emulated communication depends on the performance capacity of the network, the DUT and the tester itself. The result of the test is a collection of time-series measurements collected by various modules of the tester. The subsequent analysis of these results gives an estimation of the DUT's performance limits.
- Dynamically scalable: the performance test allows for dynamic adjustment of some parameters to simulate changing traffic load and network conditions. The test is driven by a profile defining how these adjustable parameters should change over time, so-called "timeline".

The performance test timeline is a part of the test configuration and consists of a number of time segments. In each segment, the change of each of the adjustable parameters is individually defined by their initial and final value in this segment and the type of interpolation between them.

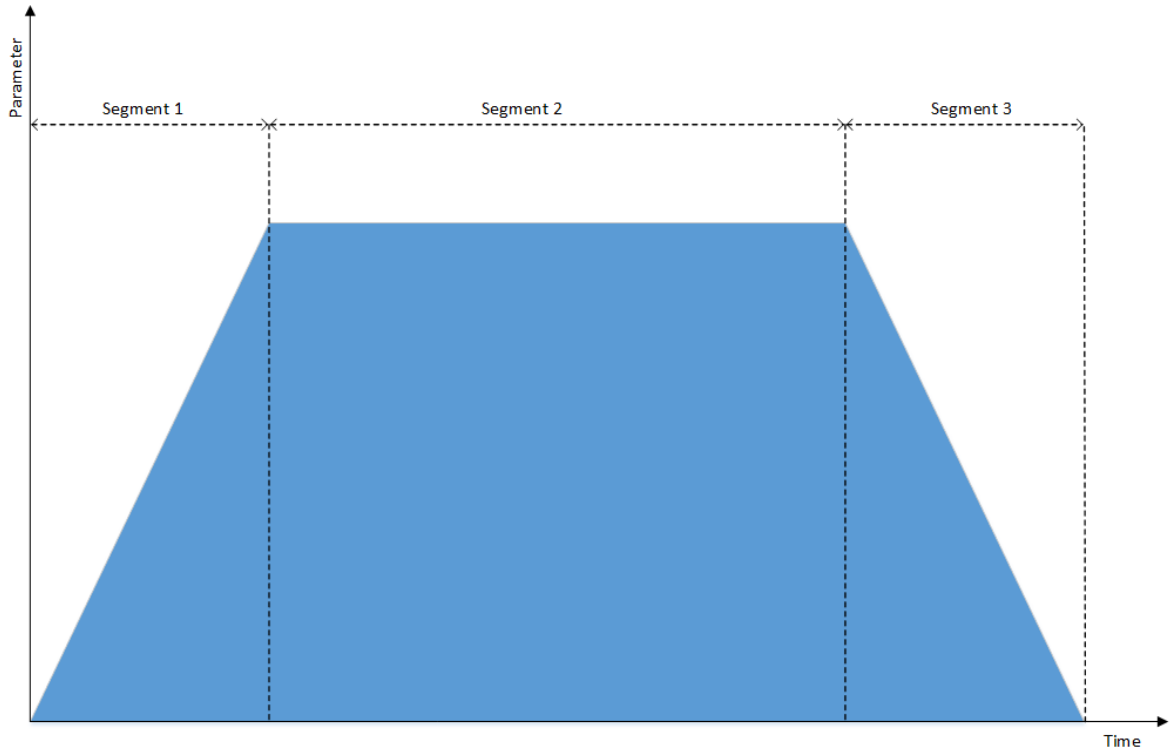

Figure 2 - Timeline Example

2.1.2 Generic Design

The performance test tool contains multiple components responsible for the different aspects of load generation and analysis. In the current implementation, these are:

- CoAP Client Emulation
- Impairment Generator
- Passive Monitoring

These components are described in detail in the following sections.

The Timeline Controller is the central component of the performance test tool that is responsible for initializing and controlling all other components during the test execution. The Timeline Controller interprets the timeline configuration, constantly calculates the current values for each of the adjustable parameters and sends them to the subcomponents under its control.

Per design, the Timeline Controller's function is generic and independent from the specific functions provided by the subcomponents. Each subcomponent provides identical interface to receive configuration or the parameter updates and to generate statistics. This way, the functionality of the test tool can be easily extended with new types of components (e.g. for different protocols) or by extending functionality of existing components without a need to modify the Timeline Controller or any other aspect of the framework.

2.1.3 Communication Bus

All performance test tool modules communicate with each other and the F-Interop framework via common message bus. F-Interop framework utilizes AMQP protocol for this purpose and defines a common message format. F-Interop framework uses RabbitMQ as the AMQP message broker.

When the test tool is instantiated by the F-Interop framework (specifically by the Session Orchestrator), the parameters necessary to connect to the common message bus are passed to the tool via environment variables:

- location of the AMQP message broker (host and port)
- AMQP vhost used by the session
- AMQP credentials (username and password)
- AMQP exchange name

F-Interop framework uses AMQP vhosts as the mechanism to isolate multiple test sessions that can exist within framework. All control plane communication between modules is performed within specified vhost only. Since access to each vhost is authorized with the individual credentials, applications running in the other test sessions have no access to it, ensuring the confidentiality of the test execution.

Each of the components within the performance test tool, including the Timeline Controller and all subcomponents is a separate process and maintains its own connection to the AMQP session.

2.2 Instantiation Process

The instantiation process of the performance test tool is mostly identical to the process of instantiation of other tools provided by the F-Interop platform e.g. interoperability testing tools.
An instance of the Performance Testing Tool Container is spawned by the SO when a test is about to start, indicated by the GUI

2.2.1 Performance Testing Tool Container

The performance testing tool is provided in the form of a Docker container. In order to be usable by the F-Interop framework, the container must be built and registered in the F-Interop's Docker repository under a name in specific format, in our case:

- testing_tool-performance-coap_client (as version-less name)
- testing_tool-performance-coap_client-v<VERSION> (with version specified)

The source code for the performance testing tool provides necessary Dockerfile configuration and a script for easy building and registering of the container.

In addition, the performance testing tool provides two configuration scripts for the Session Orchestrator itself:

- index.json: file containing all necessary metadata to describe the tool, including a tool-specific configuration UI template.
- supervisor.conf.j2 : a template for the Supervisor configuration, describing how the tool can be instantiated/launched

These two files are located separately in the F-Interop platform, under the path finterop/orchestrator/templates/f-interop/performance-coapclient/ in the Session Orchestrator deployment. Note that the in the future releases of the F-Interop platform, it is planned to access these files via the tool's container (as attached metadata), instead of accessing static files.

See the Annex 6.XXX for the detailed description of the Orchestrator configuration scripts.

2.2.2 Orchestration Process with Performance Test Tool

In the current implementation of the F-Interop framework, the orchestration of the test session is a complex communication process between the User interface (UI), Session Orchestrator (SO) and the Test Tool (TT). The goal of the orchestration is to launch the selected testing tool, provide it with configuration and parameters to access the common communication bus (AMQP session).

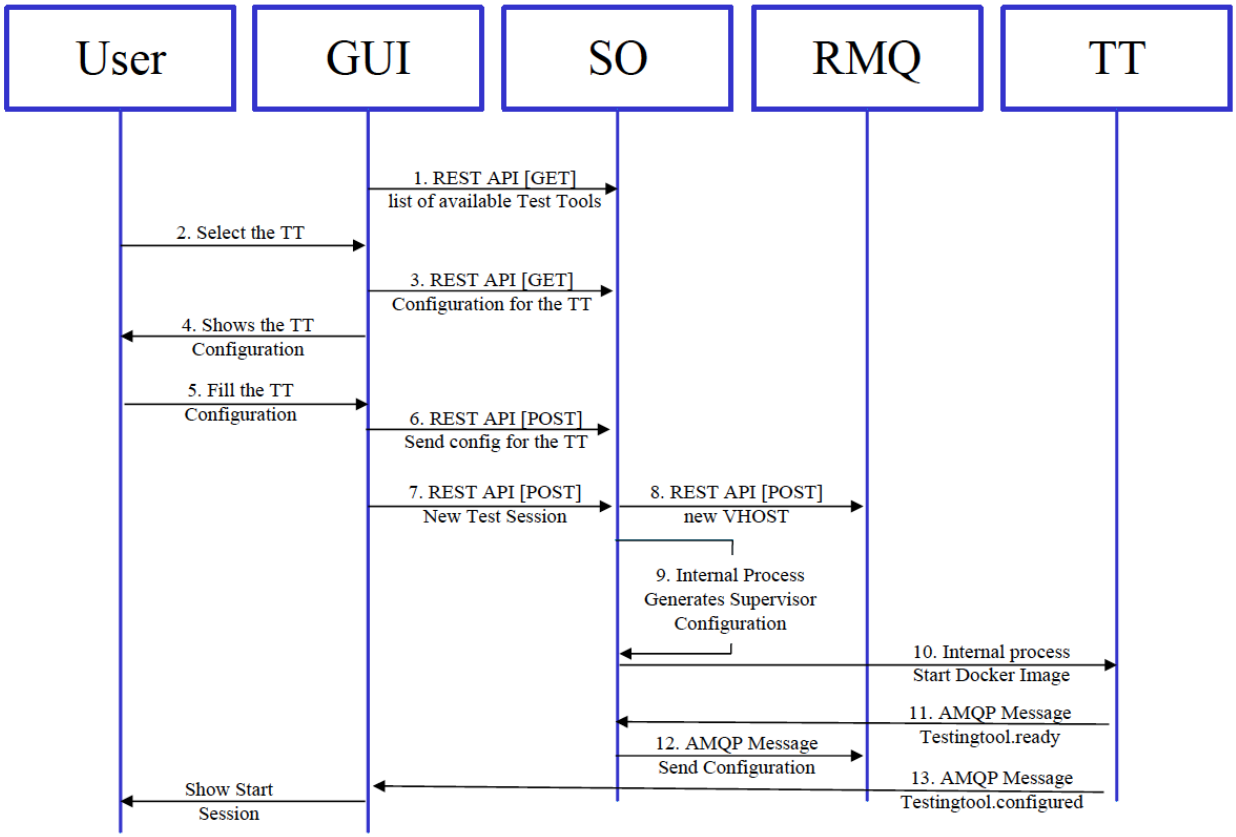

Figure 3 - Orchestration process of a Test Tool

In detail, the orchestration process involves following steps:

1. The UI queries the SO for the list of available tool types
2. The User selects the desired tool type (in our case - performance) and implementation (CoAP Client Emulation).
3. The UI queries the SO for the configuration file (index.json).
4. The UI generates the configuration page for the selected tool type using the UI template provided in the configuration file.
5. The User fills in the configuration values (static parameters and the timeline)
6. From the user's input, the UI generates a configuration file specific to the performance testing tool.

7. The UI requests the SO to create a new test session, using SO's REST API. The request includes the configuration file created by the tool's specific UI, which is passed to the SO opaquely.
8. The SO accesses the AMQP broker to create a new vhost for the session and sets the access credentials (generated randomly) and permissions. At this point, the common message bus for the test session is created and further communication between components occurs via AMQP until the session is about to be deleted.
9. The SO reads the Supervisor template (supervisor.conf.j2) provided for the selected tool and generates a Supervisor configuration file sufficient to launch the tool. The AMQP access information is included in this configuration in form of environment variables.
10. The Supervisor daemon running within the F-Interop framework interprets the new configuration file and launches the test tool - in our case, a Docker container is instantiated from the image "testing_tool-performance-coap_client". The necessary configuration is passed to the container via environment variables.
11. After the container has been launched, the tools may need some time to fully start up. When this process is complete, the test tool issues a message to the bus indicating its readiness to accept the test configuration.
12. The session orchestrator sends the configuration file from the initial request to create session (Step 7) to the test tool
13. The test tool verifies the configuration, and if it is acceptable, returns a message indicating that the tool is configured and is ready to perform the test. UI displays a button to start the test, accordingly

After these steps, the testing tool is successfully launched, configured and ready for the test execution upon request.

2.2.3 Session Teardown

After this point, the Session Orchestrator does not play an active role during the test execution. Only at the end of the session, the UI can request the teardown of the test session by sending an appropriate REST request to the Orchestrator.

When the session is to be removed, the Session Orchestrator removes the previously created Supervisor configuration, which causes the Supervisor daemon to terminate and remove the Docker container instance. The SO also removes the AMQP vhost, thus terminating the message bus that was used by the session.

2.3 Test Execution Process

After the test session with a performance test tool has been orchestrated and the test configuration sent, the test tool becomes ready for the execution, as signalled by testingtool.configured message to the GUI. At this point the GUI will display a button by pressing which the user can start the test. The test is started when the Timeline Controller receives testsuite.start message from the GUI.

From the previously received test configuration data, the Timeline Controller has a profile defining how each of the dynamically configurable test parameters should be modified during the test. Once the test is started, the Timeline Controller performs periodic updates for these dynamic variables.

At the moment of an update, the current test timestamp (seconds since test start) is used to locate the current timeline segment. For each dynamic parameter, the Timeline Controller interpolates between the initial and the final values of the segment to calculate value desired for this particular moment.

The values of dynamic parameters are then transmitted to the performance tool's components using performance.setvalues message via test session's AMQP bus. The components perform necessary adjustments:

- CoAP Client Emulation : creates or removes the emulated client nodes, and adjusts the total traffic rate
- Impairment Generator : adjusts the impairment settings by sending appropriate configuration commands to the NETEM subsystem
- Passive Monitor : adjusts the estimated energy consumption coefficient

During the test execution, the components of the performance testing tool collect various statistics, from their own protocol emulation (such as response time or protocol errors) and reported by the

operating system (impairment and traffic statistics). These statistics are collected in 1-second intervals and sent as performance.stats messages to the AMQP bus.

These messages are collected by the performance visualization component. The statistics are presented in real time graphically to the user and are also stored in a database for a retrospective analysis or review. The visualization component is provided by DG as part of Task T3.3 and described in detail in the Deliverable D3.4.

The test is terminated when the execution time reaches the end of the timeline. The Timeline Controller transmits the last update message with the final values of the last timeline segment, ensuring a consistent state at the end of the test. Then, the test termination is signalled to the GUI using testcoordination.testcase.finished followed by testcoordination.testcase.report.

The user also has the possibility to abort a running test. GUI provides an appropriate button and will send testcoordination.testsuite.abort. In such case, the Timeline Controller will terminate the execution immediately. As in the normal case, it will send out the final values for the timeline and report the termination of the test to the GUI.

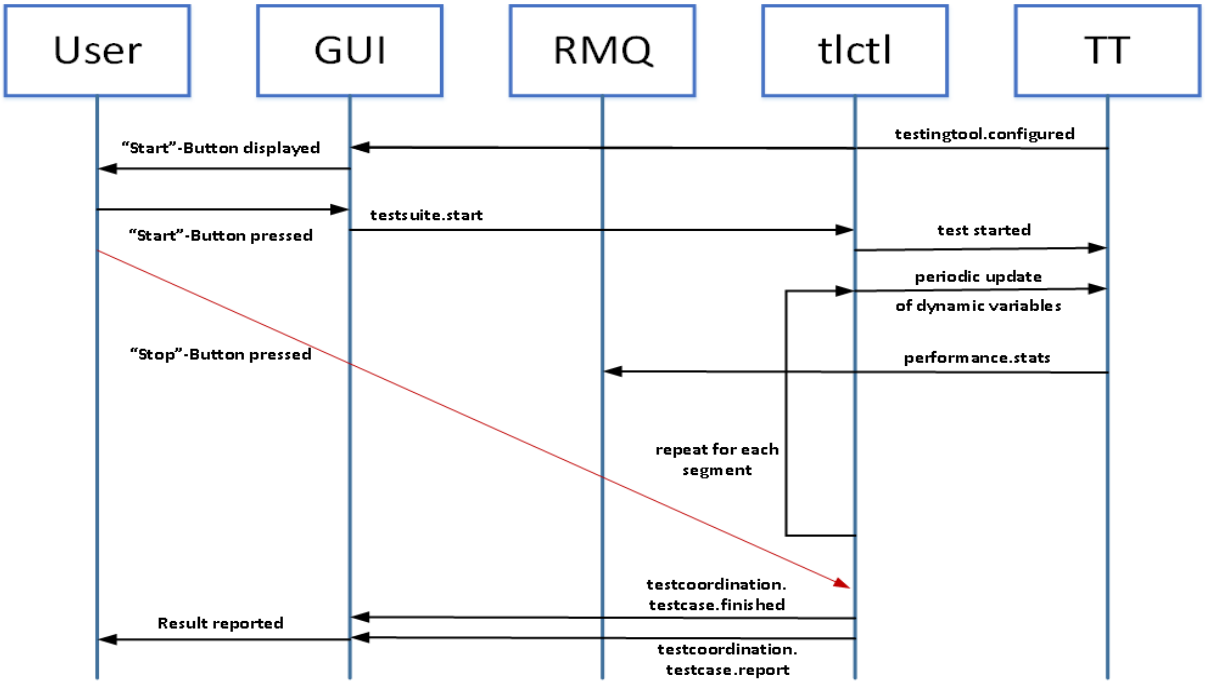

Figure 4 - Message flow during test execution

2.4 Communication between modules

The communication between all modules within a test session, including the internal communication between the Timeline Controller and the subcomponents occurs on the same AMQP bus created and assigned by the Session Orchestrator. The bus is established on a separate vhost, providing isolation of the test session from others and disallowing access for other users without proper credentials.

All messages follow the common format defined for the F-Interop platform in the API documentation [1]

```
- - - - - - - - - - - - - - - - - - - - -
Message routing key: control.session
 - - -
Message properties: {
    "content_type": "application/json",
    "message_id": "12404929-8974-4305-aebc-5676a5d97f04",
    "timestamp": 1498685266
}
 - - -
Message body: {
    "_api_version": "0.1.32",
    "_type": "testingtool.configured",
    "description": "Event Testing tool CONFIGURED"
}
- - - - - - - - - - - - - - - - - - - - -
```

Figure 5 - Example of the message format

The following table describes all messages in use by the performance testing tool.

Table 2 - Messages used by the Performance Test Tool

Message type	Direction	Description
performance.heartbeat	SC > TL	Periodic status reports from subcomponents to the Timeline Controller.
testingtool.ready	TL > UI,SO	Inform the UI and Orchestrator that the tool has been successfully instantiated and is ready for the configuration.
performance.configuration	SO > TL,SC	Provides test configuration to the performance testing tool
testingtool.configured	TL > UI	Inform the UI that the tool has been successfully configured and is ready to run
testcoordination.testsuite.start	UI > TL	Instruct the testing tool to start the test execution
performance.setvalues	TL > SC	Updates the dynamically adjustable parameters in the performance testing tool subcomponents
performance.stats	SC > VT	Transmit the performance test statistics to the visualization/result storage component
testcoordination.testcase.finished	TL > UI	Inform the UI that the test has been completed
testcoordination.testsuite.report	TL > UI	Inform the UI about the verdict on the complete test
testcoordination.testsuite.abort	UI > TL	Instruct the testing tool to abort the test execution

2.5 Test Tool Configuration

The complete test configuration is supplied to the performance testing tool via performance.configuration message on the AMQP bus. The message contains a JSON data block created by the UI or a configuration script. A full example of a configuration block can be seen in the Annex 6.1.4.

The general format of the message contains three major parts `static`, `initial` and `segments`:

```
config = {
    "static": {
        "<static-parameter-name>" : value,
```

```
        ...
    }
    "initial": {
        "<dynamic-parameter-name>" : value,
        ...
    }
    "segments": [
        {
            "name"     : "Name/Description of the timeline segment",
            "duration" : segment_duration_in_seconds,
            "values"   : {
                "<dynamic-parameter-name>" : [ final_value, interpolation_type ],
                ...
            }
        },
        ...
    ]
}
```

These sections have the following meaning and format as described below and in the accompanying parameter tables.

static: These configuration parameters define the basic configuration of the test and the simulated clients. They remain unchanged for the duration of the test.

initial: This section defines the initial parameters for the dynamically adjustable test parameters controlled by the timeline. They represent the starting values for the first segment of the timeline whereas following timeline segments take their initial values from the final values of the previous segment.

segments: A list of the timeline segments. Each segment is denoted with its symbolic name and duration. Further for each of the dynamically adjustable parameter it defines the final value at the end of the segment and the interpolation method between the initial and final value. The initial value for each parameter is derived from the final value of the previous segment, or from "initial" section for the first segment.

`static-parameter-name` and `dynamic-parameter-name` are strings uniquely defining parameter names described in the tables below. Non-mandatory dynamic parameters that the user is not interested in modifying during the test can be omitted in both `initial` and `segments` sections.

The following tables list the currently available static and dynamic configuration parameters provided by the tool.

2.5.1 Static Configuration Parameters

Table 3 - Static Configuration Parameters

Name	Type	Description
coapclient. dst_address	string	Address (IPv4, IPv6 or hostname) of the target CoAP Server. Used by both CoAP client emulation to direct the requests to, but also by the impairment generator and passive traffic monitoring to identify the test traffic streams
coapclient. dst_port	int	Port number of the target CoAP server.
coapclient. src_port	int	Base port number of the emulated CoAP clients. The requests generated by the emulated clients will appear to originate from ports src_port, src_port+1, src_port+2,...

coapclient. request_timeout	int	Request timeout in milliseconds. If the IUT
coapclient. request_sequence	sequence	Sequence of requests to be performed by each client. See below for a detailed description.
coapclient. adjust_rate	bool	Automatically reduce request rate if request failures are detected

The `coapclient.request_sequence` section has a special complex format:

```
config = {
    "static": {
        ...
        "coapclient.request_sequence" : [
            {
                "type": "<request_type>",
                "path": "<request_path>",
                "body": "<request_body>"
            },
            ...
        ],
        ...
    }
}
```

The sequence is a list of CoAP requests to be performed by the emulated CoAP clients. Each client instance will emit requests cyclically from this list and independently from each other. The requests are defined by the following parameters:

Table 4 - CoAP Request Parameters

Name	Type	Description
type	selection	Type of the request : "GET" or "POST"
path	string	Path of the request, e.g. "/.well-known/core"
body	string	Optional body content to be added to the request

2.5.2 Dynamic Configuration Parameters

The following table lists the dynamically adjustable parameters provided by the Performance TestingTool.

Table 5 - Dynamic Configuration Parameters

Name	Type	Description
coapclient. clients	int	Number of emulated client instances. When this dynamic parameter is modified, the tool will create or destroy emulated CoAP client instances accordingly to match the number. The instances are allowed to complete the currently running request.
coapclient. request_rate	float	Total rate of CoAP transactions initiated by the tool
impairment. delay	float	Delay in milliseconds added to the egress packets by the impairment generator

impairment. delay_variation	float	Delay variation added to the test traffic
impairment. delay_correlation	float	Probability of a packet's delay to correlate with the previous packet, in %
impairment. delay_distribution	selection	Type of delay distribution if delay variation is added. Possible values are: "uniform" : uniform random distribution "normal" : Gaussian distribution "pareto" : 80/20% relation of uniform/normal "paretonormal" : 25/75% relation of uniform normal
impairment. loss	float	Packet loss probability, in %
impairment. loss_correlation	float	Packet loss probability for packets following a dropped one. Allows simulation of loss bursts
impairment. duplicate	float	Probability of packet duplication, in %
impairment. corrupt	float	Probability of packet corruption, in %
monitoring. energy_coefficient	float	Coefficient for energy consumption estimation (µWh per byte)

The definition for each timeline segment provides following parameters:

Table 6 - Dynamic Configuration Parameters

Name	Type	Description
name	string	Symbolic name / description of the time segment
duration	int	Duration of the time segment in seconds
The following parameters are available for each of the supported dynamic parameters independently:		
final_value	(specific for each parameter)	Final value of the parameter at the end of the time segment. Also serves as the initial value for the next segment.
interpolation_type	selection	Defines how the Timeline Controller interpolates between the initial and the final values during the time segment.

The value V_T at a specific time T since beginning of the segment with duration D is calculated as:

$V_T = V_0 + (V_F - V_0) * f(\frac{T}{D})$, where f is one of the interpolation functions:

"linear" : $f(x) = x$

"power2" : $f(x) = x^2$

"root2" : $f(x) = \sqrt{x}$

"constant" : $f(x) = 1$

2.5.3 Configuration with the F-Interop GUI

In order to generate the configuration for the performance test tool, F-Interop GUI provides a tool-specific configuration page. The page is created dynamically from a template provided with the tool. The template specifies the list of configuration parameters, their types and constraints to be presented

to the user. The dynamic page generation from the template allows for easy extension it in the future, if more configuration parameters are added to the tool.

An excerpt from the template provides an example of such definition, with the full template shown in Annex 6.1.3.

```
ui_information = [
    ...
    {
        "field_name": "coapclient.dst_port",
        "description": "Port number of the target CoAP server",
        "type": "int",
        "value_default": "5683",
        "value_min": "1",
        "value_max": "65535",
        "mandatory": true,
        "timeline": false
    },
    ...
    {
        "field_name": "impairment.loss",
        "description": "Packet loss probability, in %",
        "type": "float",
        "value_default": "0.0",
        "value_min": "0.0",
        "mandatory": true,
        "timeline": true
    },
]
```

After reading in the template, the GUI generates a HTML/Javascript page where the user can enter the configuration parameters. The configuration page includes dynamically resizable tables for the request sequence and the timeline with arbitrary number of rows respective columns.

In case of the timeline definition, the dynamically adjustable parameters that the user is not interested in changing can be disabled.

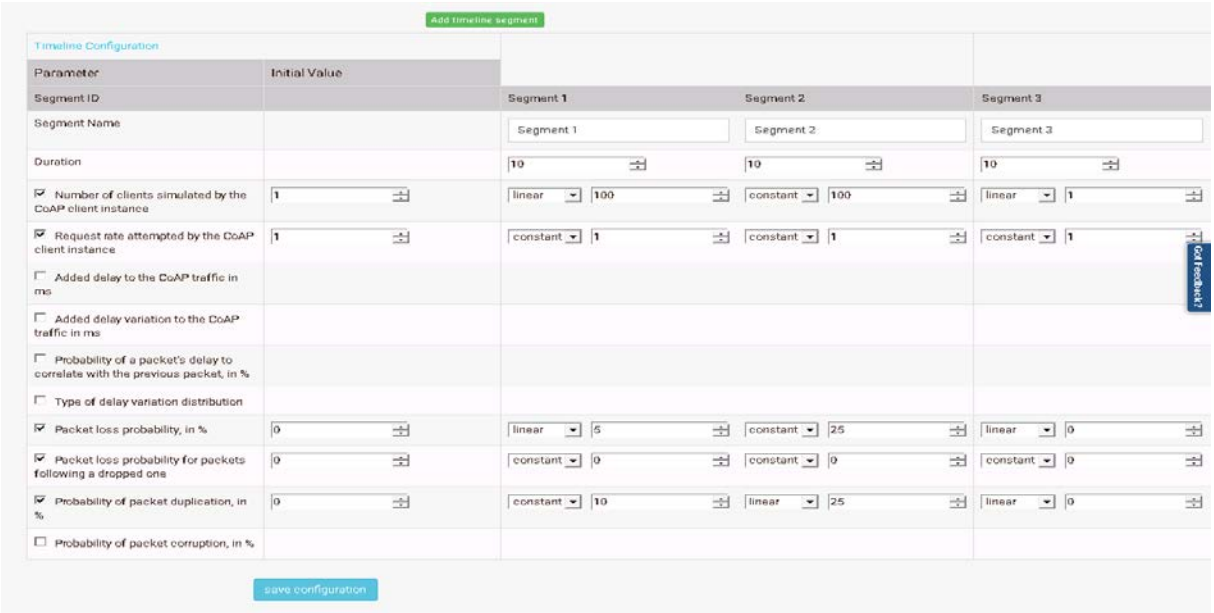

Figure 6 – Example of Timeline with disabled parameters

3 Performance Test Tool Modules Functionality

This section describes the individual components of the performance testing tools in more detail. For each module, we describe their detailed function, interfaces, configurable parameters and statistics provided by the module.

3.1 Timeline Controller

The timeline controller connects all submodules and allows a dynamic approach to performance tests. It verifies that the test tools have started via heartbeat messages sent by the modules.
These heartbeat messages also allow ensuring that all test tools are configured and ready for a test session. In case a test tool should stop working, due to error, misconfiguration or other means, the timeline controller will detect and log this incident.

3.2 CoAP Client Emulation

The CoAP Client Emulation is the test tool that allows emulating CoAP-conformant traffic. The tool is component is based on the aiocoap Python library. Based on the configuration it will emulate multiple independent CoAP Clients, periodically sending CoAP requests to the IUT and collecting the statistics on the responses respective errors.

Table 6 - CoAP Client Emulation Statistics

Name	Units	Description
coapclient.rtt_min	ms	Minimum, average and maximum transaction response time, i.e. time between sending a request and receiving a response or error.
coapclient.rtt_avg	ms	
coapclient.rtt_max	ms	
coapclient.req_attempt	trans./s	Rate of attempted transactions
coapclient.req_success	trans./s	Rate of successful transactions
coapclient.req_failed	trans./s	Rate of failed transactions, i.e. an error response was received from the server
coapclient.req_timeout	trans./s	Rate of requests timed out, i.e. a response was not received within defined timeout interval

3.3 Impairment Generator

The Impairment Generator component provides means to simulate impaired network conditions for the test traffic. The Impairment Generator is based on the NETEM ("NETwork EMulator") module of the QoS subsystem in Linux kernel and supports all impairment methods, parameters and statistics provided by it. The component serves as a wrapper for NETEM and will generate appropriate commands to NETEM kernel module when the impairment parameters change.

Table 7 - Impairment Statistics

Name	Units	Description
impairment.bytes	bytes	Total number of bytes affected by impairment profile
impairment.packets	packets	Total number of packets affected by impairment profile
impairment.drops	packets	Number of packets explicitly dropped
impairment.overlimits	packets	Number of packets dropped due to bandwidth limits
impairment.requeues	packets	Number of packets reordered

impairment.dups	packets	Number of packets duplicated
impairment.corruptions	packets	Number of packets corrupted

3.4 Passive Monitoring

The Passive Monitoring component is responsible for the collection of traffic statistics of the CoAP test traffic at the layers 2/3/4, i.e. raw packet and byte counts, whereas the CoAP emulation is concerned with the statistics at the application layer. The module also provides the statistics on the estimated power consumption of the IUT, when provided a conversion coefficient.

Table 8 - Passive Monitoring Statistics

Name	Units	Description
passivemonitoring.packetsin	packets	Number of ingress packets
passivemonitoring.packetsout	packets	Number of egress packets
passivemonitoring.bytesin	bytes	Number of ingress bytes
passivemonitoring.bytesout	bytes	Number of egress bytes
passivemonitoring.coap_confirm	packets	Total number of CoAP Confirm packets
passivemonitoring.coap_non-confirm	packets	Total number of CoAP Non-Confirm packets
passivemonitoring.coap_acknowledge	packets	Total number of CoAP Acknowledge packets
passivemonitoring.coap_reset	packets	Total number of CoAP Reset packets
passivemonitoring.estimated_energy	µW	Estimated energy consumption

4 Future Developments

As the project evolves with time, we strive to create a Performance Testing Tool that can fit the needs of both F-Interop platform and users.
Therefor

- Abort/restart of test tools on error or misconfiguration
- Verify Test Tool configuration
- Improve integration with other testing tools
- Improve stability and speed
- Bug fixing and Improvements

The ability to abort the current test due to errors is an important task for the timeline controller.
If any error/s occur that might decrease the reliability of the Test Tool, the test need to be aborted and restarted to ensure the best possible results.
In case that a submodule stops working the timeline controller will make sure that this submodule is restarted as fast as possible to ensure a perfect test procedure.

The validation of the configuration will play a major role in this.
Most errors that cause the decrease in reliability are caused by misconfiguration.
Verifying that the configuration corresponds to the required parameters will ensure a smooth start of the test.

Improving the integration with other test tools will fortify the stability of the F-Interop platform as one, was well as providing a unified user interface and superior user experience.

5 References

[1] F-Interop API Documentation - http://doc.f-interop.eu/?amqp#messages-format

6 Annex

6.1 Configuration Files

6.1.1 Supervisor configuration file

supervisor.coapclient.conf:

```
unix_http_server]
file=/tmp/supervisor.sock   ; (the path to the socket file)

[supervisord]
logfile=/tmp/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation backups;default 10)
loglevel=info                ; (log level;default info; others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default supervisord.pid)
nodaemon=false               ; (start in foreground if true;default false)
minfds=1024                  ; (min. avail startup file descriptors;default 1024)
minprocs=200                 ; (min. avail process descriptors;default 200)

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL  for a unix socket

[program:tlctl]
command = sh -c "python3.6 -m tlctl"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/tlctl-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5
environment=
   FINTEROP_PERF_MODULE_NAME="tlctl",
   FINTEROP_PERF_SET="impctl,coapclient,pmctl"

[program:pmctl]
command = sh -c "python3.6 -m pmctl"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
```

```
stdout_logfile = /var/log/pmctl-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5
environment=
   FINTEROP_PERF_MODULE_NAME="pmctl"

[program:impctl]
command = sh -c "python3.6 -m impctl"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/impctl-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5
environment=
   FINTEROP_PERF_MODULE_NAME="impctl"

[program:coapclient]
command = sh -c "python3.6 -m coapclient"
autorestart=false
stopsignal=INT
stopasgroup=true
loglevel=debug
redirect_stderr=true
stdout_logfile = /var/log/coapclient-stdout.log
stdout_logfile_maxbytes = 10MB
stdout_logfile_backups = 5
environment=
   FINTEROP_PERF_MODULE_NAME="coapclient"
```

6.1.2 Dockerfile

Dockerfile.coapclient:

```
FROM ubuntu:16.04
MAINTAINER broese@eantc.de

RUN apt-get update -y -qq && apt-get -y -qq install software-properties-common
RUN add-apt-repository ppa:jonathonf/python-3.6
RUN apt-get update -y -qq && apt-get -y -qq install python3.6
RUN apt-get -y -qq install python3-dev python3-setuptools python3-pip supervisor nano sudo libpcap-dev less
RUN python3.6 -m pip install --upgrade pip
RUN python3.6 -m pip install pika aiocoap requests
RUN ln -sf /usr/lib/x86_64-linux-gnu/libpcap.so.1.* /usr/lib/libpcap.so.1

# environment
```

```
ENV PATH="/finterop_perf:$PATH"
ENV FINTEROP_PERF_DEBUG 0
ENV FINTEROP_PERF_THREADS 4

# software
ADD VERSION .
ADD ./finterop_perf /finterop_perf
WORKDIR /finterop_perf

# launch processes
CMD supervisord -c supervisor.coapclient.conf
```

6.1.3 F-Interop GUI template

index.json:

```
{
  "_type": "testsuite.manifest",
  "testing_tool": "CoAP Client emulation for performance testing",
  "version": "0.0.1",
  "test_type": "performance",
  "protocols_under_test": ["CoAP"],
  "protocols_info": [
    {
      "protocol": "CoAP_CORE",
      "specification_id":"RFC7252",
      "specification_ref": "https://tools.ietf.org/html/rfc7252",
      "test_description_id": "TD_COAP_CORE_PERF"
    }
  ],
  "underlying_supported_protocol": ["ipv6","ipv4","udp"],
  "agent_names": [],
  "iut_roles": ["coap_server"],
  "available_location_models:": ["loc_mod_A_single_user", "loc_mod_F_single_user"],

  "ui_information":[
    {
      "field_name": "Testing Tool Description",
      "type": "text",
      "value": [
        "Performance testing tool simulating CoAP clients"
       ]
    },
    {
      "field_name": "coapclient.dst_address",
      "description": "Address (IPv4, IPv6 or hostname) of the target CoAP server",
      "type": "text",
      "value_default": "californium.eclipse.org",
      "mandatory": true,
```

```
  "timeline": false
},
{
  "field_name": "coapclient.dst_port",
  "description": "Port number of the target CoAP server",
  "type": "int",
  "value_default": "5683",
  "value_min": "1",
  "value_max": "65535",
  "mandatory": true,
  "timeline": false
},
{
  "field_name": "coapclient.src_port",
  "description": "Base port number of the emulated clients",
  "type": "int",
  "value_default": "10000",
  "value_min": "1",
  "value_max": "65535",
  "mandatory": true,
  "timeline": false
},
{
  "field_name": "coapclient.request_timeout",
  "description": "Request timeout in milliseconds",
  "type": "int",
  "value_default": "1000",
  "value_min": "0",
  "mandatory": true,
  "timeline": false
},
{
  "field_name": "coapclient.request_sequence",
  "description": "Sequence of requests to be performed by each client",
  "type": "request_sequence",
  "value_default": [ ],
  "mandatory": true,
  "timeline": false
},
{
  "field_name": "coapclient.adjust_rate",
  "description": "Automatically reduce the request rate if request failures are detected",
  "type": "boolean",
  "value_default": false,
  "mandatory": true,
  "timeline": false
},
{
```

```
  "field_name": "coapclient.clients",
  "description": "Number of clients simulated by the CoAP client instance",
  "type": "int",
  "value_default": "1",
  "value_min": "1",
  "value_max": "65535",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "coapclient.request_rate",
  "description": "Request rate attempted by the CoAP client instance",
  "type": "float",
  "value_min": "0.1",
  "value_default": "1.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.delay",
  "description": "Added delay to the CoAP traffic in ms",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.delay_variation",
  "description": "Added delay variation to the CoAP traffic in ms",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.delay_correlation",
  "description": "Probability of a packet's delay to correlate with the previous packet, in %",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.delay_distribution",
  "description": "Type of delay variation distribution",
```

```
  "type": "selection",
  "value":[ "uniform", "normal", "pareto", "paretonormal" ],
  "value_default": "uniform",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.loss",
  "description": "Packet loss probability, in %",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.loss_correlation",
  "description": "Packet loss probability for packets following a dropped one",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.duplicate",
  "description": "Probability of packet duplication, in %",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "impairment.corrupt",
  "description": "Probability of packet corruption, in %",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
  "mandatory": true,
  "timeline": true
},
{
  "field_name": "monitoring.energy_coefficient",
  "description": "Coefficient for energy consumption estimation (uWh per byte)",
  "type": "float",
  "value_default": "0.0",
  "value_min": "0.0",
```

```
      "mandatory": true,
      "timeline": true
    }
  ],
  "ansible_playbook": "ansible/main.yml",
  "owner": "F-Interop",
  "mantainer": "Eduard Broese",
  "mantainer_email": "broese@eantc.de"
}
```


6.1.4 Performance Test Tool configuration example


```
{
  "status": "closed",
  "tests": [],
  "users": [
    "X0AJ3Y9K",
    "f-interop"
  ],
  "configuration": {
    "segments": [
      {
        "duration": "10",
        "values": {
          "coapclient.clients": [
            "10",
            "linear"
          ],
          "impairment.delay_variation": [
            "0.0",
            "constant"
          ],
          "impairment.corrupt": [
            "0.0",
            "constant"
          ],
          "impairment.delay_correlation": [
            "0.0",
            "constant"
          ],
          "coapclient.request_rate": [
            "1.0",
            "constant"
          ],
          "impairment.delay": [
            "0.0",
            "constant"
          ],
          "impairment.duplicate": [
            "0.0",
            "constant"
          ],
          "impairment.loss_correlation": [
            "0.0",
            "constant"
          ],
          "impairment.loss": [
            "0.0",
            "constant"
```

```
    ],
    "impairment.delay_distribution": [
      "uniform",
      "constant"
    ]
  },
  "name": "Segment 1"
},
{
  "duration": "10",
  "values": {
    "coapclient.clients": [
      "10",
      "constant"
    ],
    "impairment.delay_variation": [
      "0.0",
      "constant"
    ],
    "impairment.corrupt": [
      "0.0",
      "constant"
    ],
    "impairment.delay_correlation": [
      "0.0",
      "constant"
    ],
    "coapclient.request_rate": [
      "1.0",
      "constant"
    ],
    "impairment.delay": [
      "0.0",
      "constant"
    ],
    "impairment.duplicate": [
      "0.0",
      "constant"
    ],
    "impairment.loss_correlation": [
      "0.0",
      "constant"
    ],
    "impairment.loss": [
      "0.0",
      "constant"
    ],
    "impairment.delay_distribution": [
      "uniform",
      "constant"
    ]
  },
  "name": "Segment 2"
},
{
  "duration": "10",
  "values": {
    "coapclient.clients": [
      "1",
      "linear"
```

```
      ],
      "impairment.delay_variation": [
        "0.0",
        "constant"
      ],
      "impairment.corrupt": [
        "0.0",
        "constant"
      ],
      "impairment.delay_correlation": [
        "0.0",
        "constant"
      ],
      "coapclient.request_rate": [
        "1.0",
        "constant"
      ],
      "impairment.delay": [
        "0.0",
        "constant"
      ],
      "impairment.duplicate": [
        "0.0",
        "constant"
      ],
      "impairment.loss_correlation": [
        "0.0",
        "constant"
      ],
      "impairment.loss": [
        "0.0",
        "constant"
      ],
      "impairment.delay_distribution": [
        "uniform",
        "constant"
      ]
    },
    "name": "Segment 3"
  }
],
"initial": {
  "coapclient.clients": "1",
  "impairment.delay_variation": "0.0",
  "impairment.corrupt": "0.0",
  "impairment.delay_correlation": "0.0",
  "coapclient.request_rate": "1.0",
  "impairment.delay": "0.0",
  "impairment.duplicate": "0.0",
  "impairment.loss_correlation": "0.0",
  "impairment.loss": "0.0",
  "impairment.delay_distribution": "uniform"
},
"static": {
  "coapclient.dst_address": "californium.eclipse.org",
  "coapclient.request_sequence": [
    {
      "body": "",
      "path": "/.well-known/core",
      "type": "GET"
```

```
      },
      {
        "body": "",
        "path": "/.well-known/core",
        "type": "POST"
      },
      {
        "body": "",
        "path": "/.well-known/core",
        "type": "GET"
      },
      {
        "body": "",
        "path": "/.well-known/core",
        "type": "POST"
      }
    ],
    "coapclient.dst_port": "5683",
    "coapclient.request_timeout": "1000",
    "coapclient.adjust_rate": false,
    "coapclient.src_port": "10000"
  }
 },
 "processes_status": [
  {
    "now": 1507129778,
    "group": "5c3c4e11-cba1-4daf-8ef6-dd98fcd2fd5c",
    "description": "pid 21131, uptime 5 days, 0:49:48",
    "pid": 21131,
    "stderr_logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-reference_iut-coap_client-coapthon-v0.1-stderr.log",
    "stop": 0,
    "statename": "RUNNING",
    "start": 1506694790,
    "state": 20,
    "stdout_logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-reference_iut-coap_client-coapthon-v0.1-stdout.log",
    "logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-reference_iut-coap_client-coapthon-v0.1-stdout.log",
    "exitstatus": 0,
    "spawnerr": "",
    "name": "5c3c4e11-cba1-4daf-8ef6-dd98fcd2fd5c|reference_iut-coap_client-coapthon-v0.1"
  },
  {
    "now": 1507129778,
    "group": "5c3c4e11-cba1-4daf-8ef6-dd98fcd2fd5c",
    "description": "",
    "pid": 16503,
    "stderr_logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-testing_tool-stderr.log",
    "stop": 1507129778,
    "statename": "STARTING",
    "start": 1507129778,
    "state": 10,
    "stdout_logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-testing_tool-stdout.log",
    "logfile": "/home/f-interop/finterop/finterop/orchestrator/logs/5c3c4e11-cba1-4daf-8ef6-
dd98fcd2fd5c-testing_tool-stdout.log",
    "exitstatus": 0,
    "spawnerr": "",
```

```
      "name": "5c3c4e11-cba1-4daf-8ef6-dd98fcd2fd5c|testing_tool"
    }
  ],
  "iuts": [
    {
      "execution_mode": "user-assisted",
      "version": "1",
      "role": "coap_server",
      "location": "user-facilities",
      "owner": "urn:publicid:IDN+finterop+user+ngwucibs",
      "id": "urn:publicid:IDN+finterop:ngwucibs+node+resource"
    },
    {
      "execution_mode": "reference-iut",
      "version": "1.3.1",
      "role": "coap_client",
      "location": "central-server-docker",
      "owner": "urn:publicid:IDN+finterop+authority+sa",
      "id": "reference_iut-coap_client-coapthon-v0.1"
    }
  ],
  "testing_tools": "http://orchestrator.f-interop.eu:8181/tests/f-interop/performance-coapclient"
}
```


6.2 Example of a Test Session

TBD: example of the GUI walkthrough

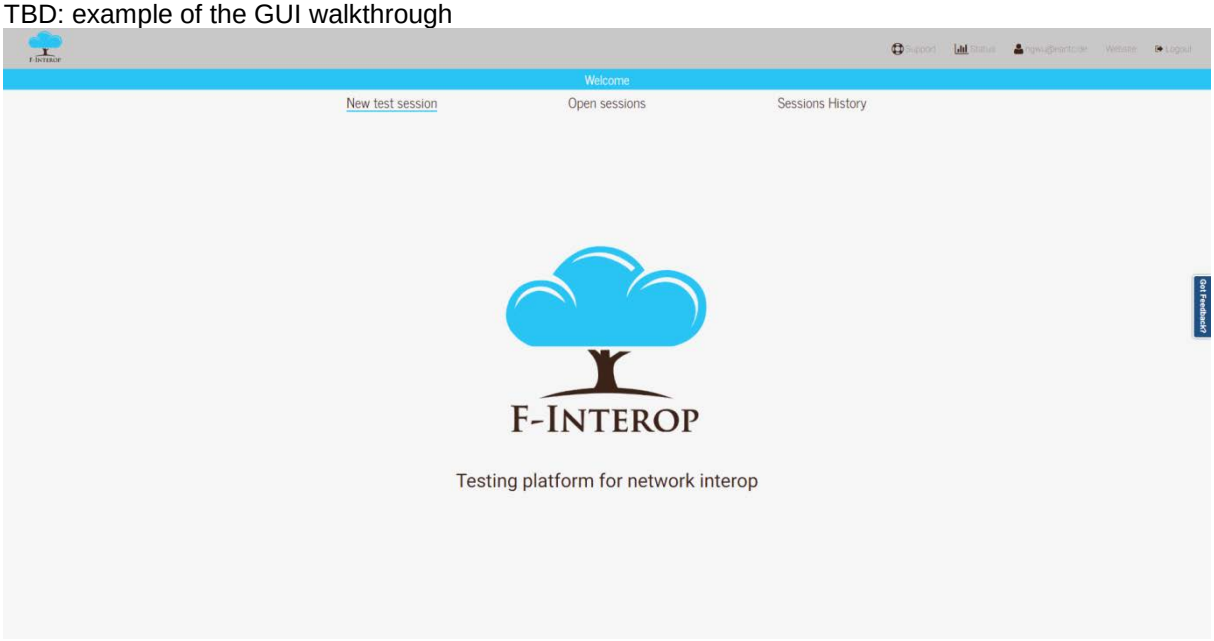

Figure 7 – New Test Session

Start by creating a new Test Session

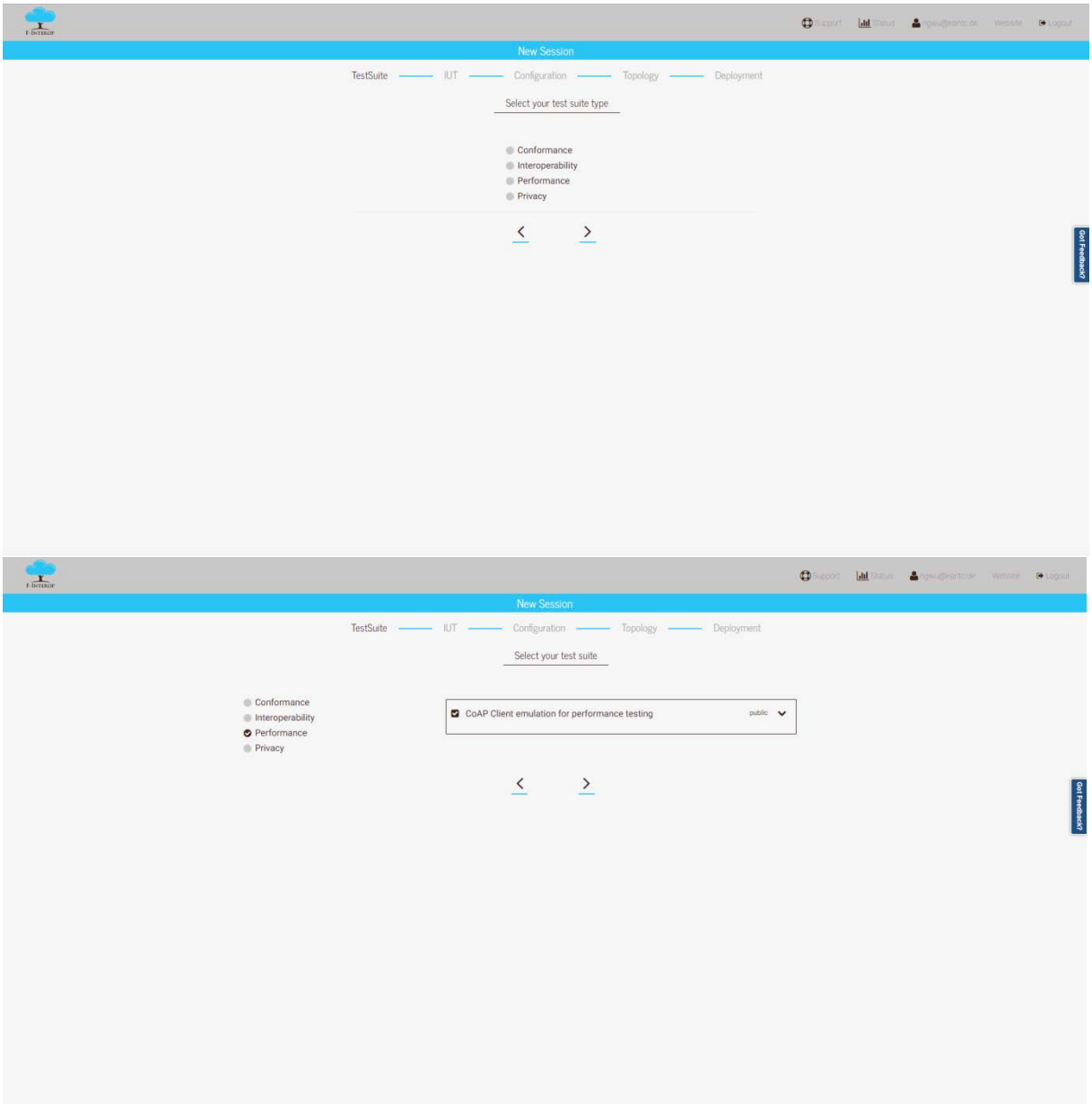

Figure 8 – Test Selection

Select “Performance” and choose CoAP client emulation for performance testing.

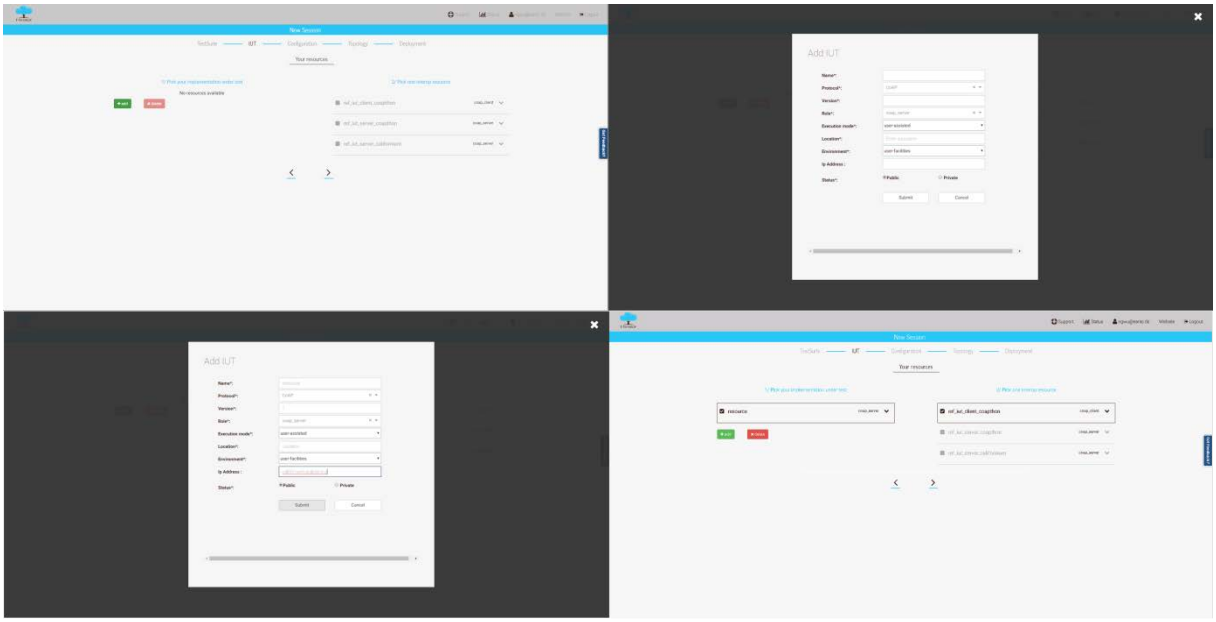

Figure 9 – Resource Management

In case this should be your first test, you'll need to add a resource to your IUT list.

Resources are saved across the test and are bound to your account.

Simply click the green "add" button on the left and a window will appear.

This window allows you to provide the information about your IUT.

First off you'll need to add a name for the resource, the protocol (CoAP-only at the moment) and the version of the protocol.

The role is set to coap_server per default, as your IUT serves as CoAP server.

Execution mode allows you to choose between "user-assisted" and "automated-iut", in this case we'll go with "user-assisted".

The provided location is used for the spatial representation map.

For environment we'll use the "user-facilities" and add an IP-Address to reach the IUT.

Finally you can choose between a public status and a private status.

Hit "Submit" to save your resource.

The window will close, notifying you that the changes were saved.

Select your IUT and the "ref_iut_client_coapthon" option and proceed with the configuration.

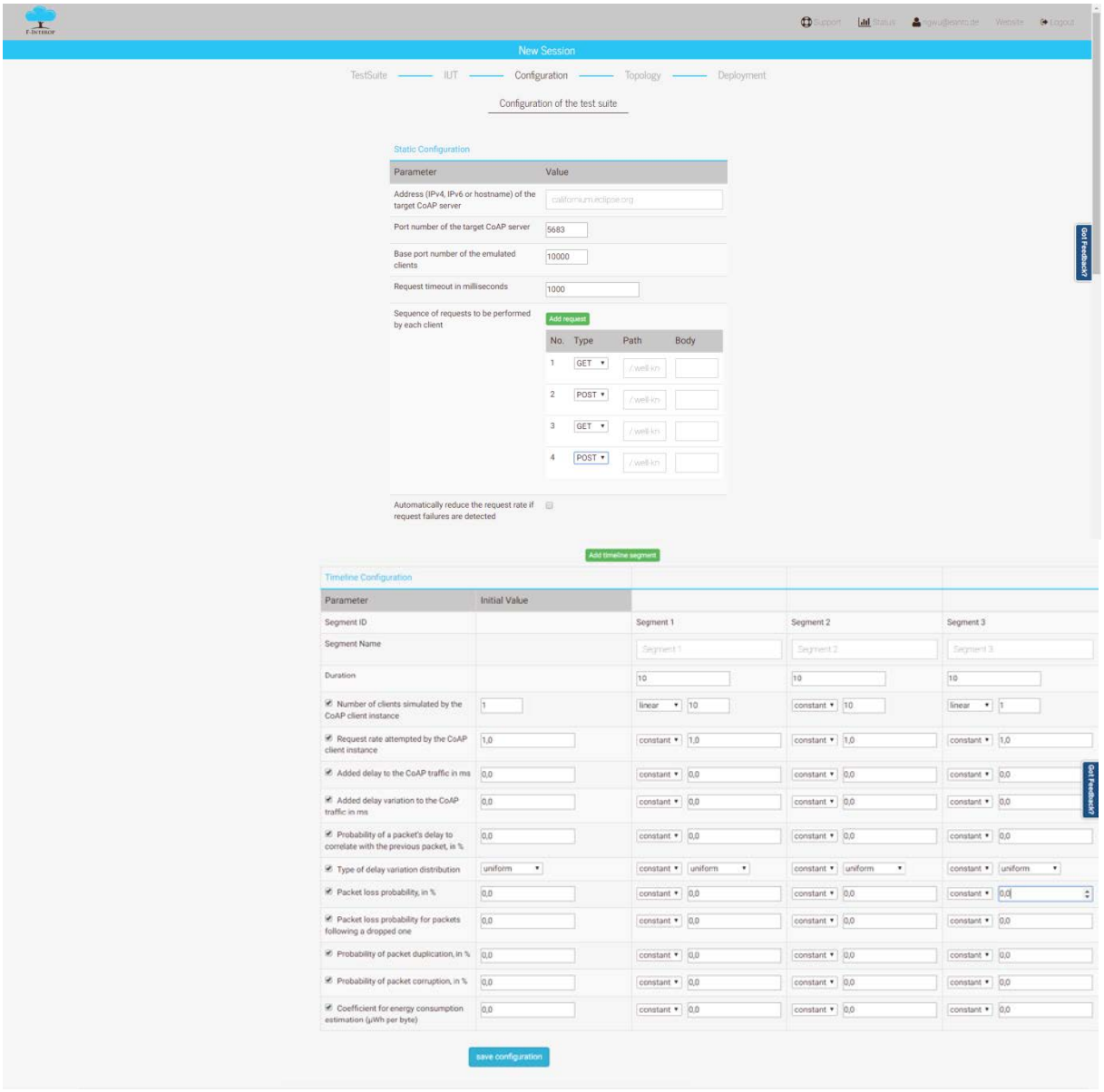

Figure 10 - Test Configuration

Provide the requested address and port number of your IUT and add the preferred requests, sent by the emulated clients.

Check the box if you want to automatically reduce the request rate if request failures are detected.

Next add some timeline segments, choose the name and length for each segment and provide the requested information for each parameter.

If you do not want to include a feature to your test, you can simply disable the row by deselecting the checkbox before the name.

Hit “save configuration” and proceed.

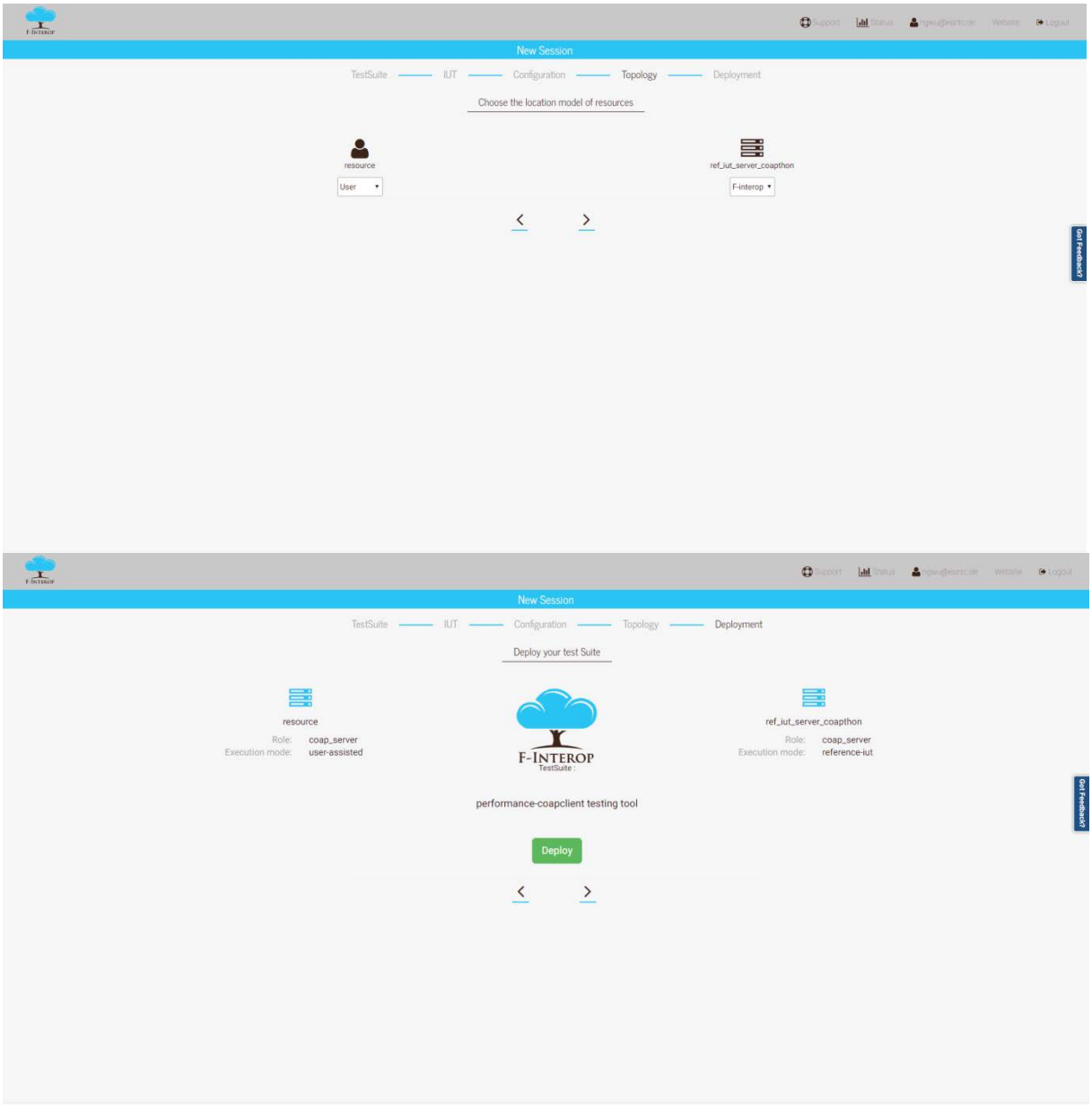

Figure 11 – Deploying the Test

Simply choose the location of the resources and proceed to deploy the test.

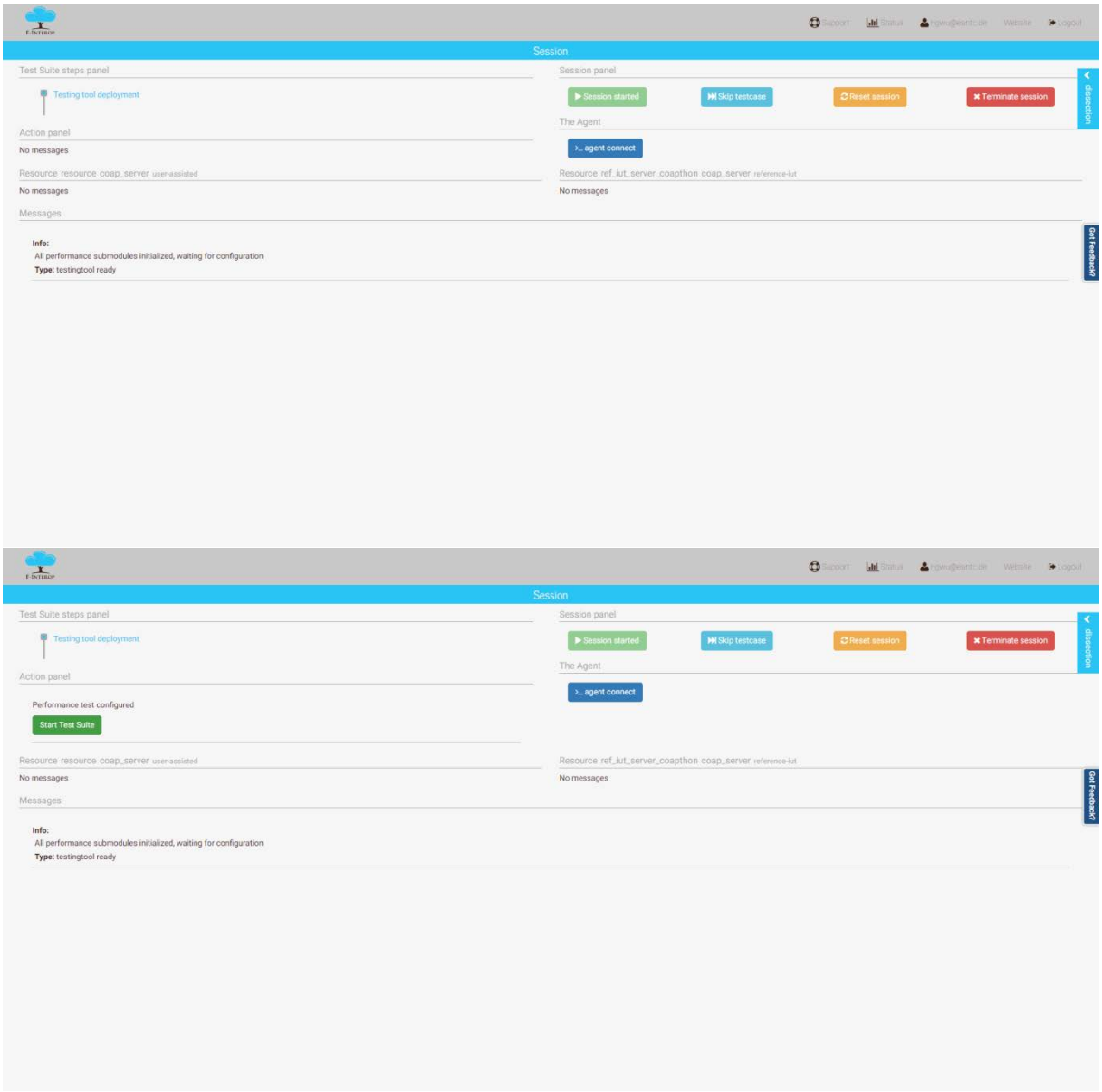

Figure 12 – Test Start

As soon as the container is created and the Testing Tools are configured, the UI will show a "Start Test Suite" button that will initiate the test upon being pressed.
The test can be cancelled any time by pressing the "Terminate session" button.

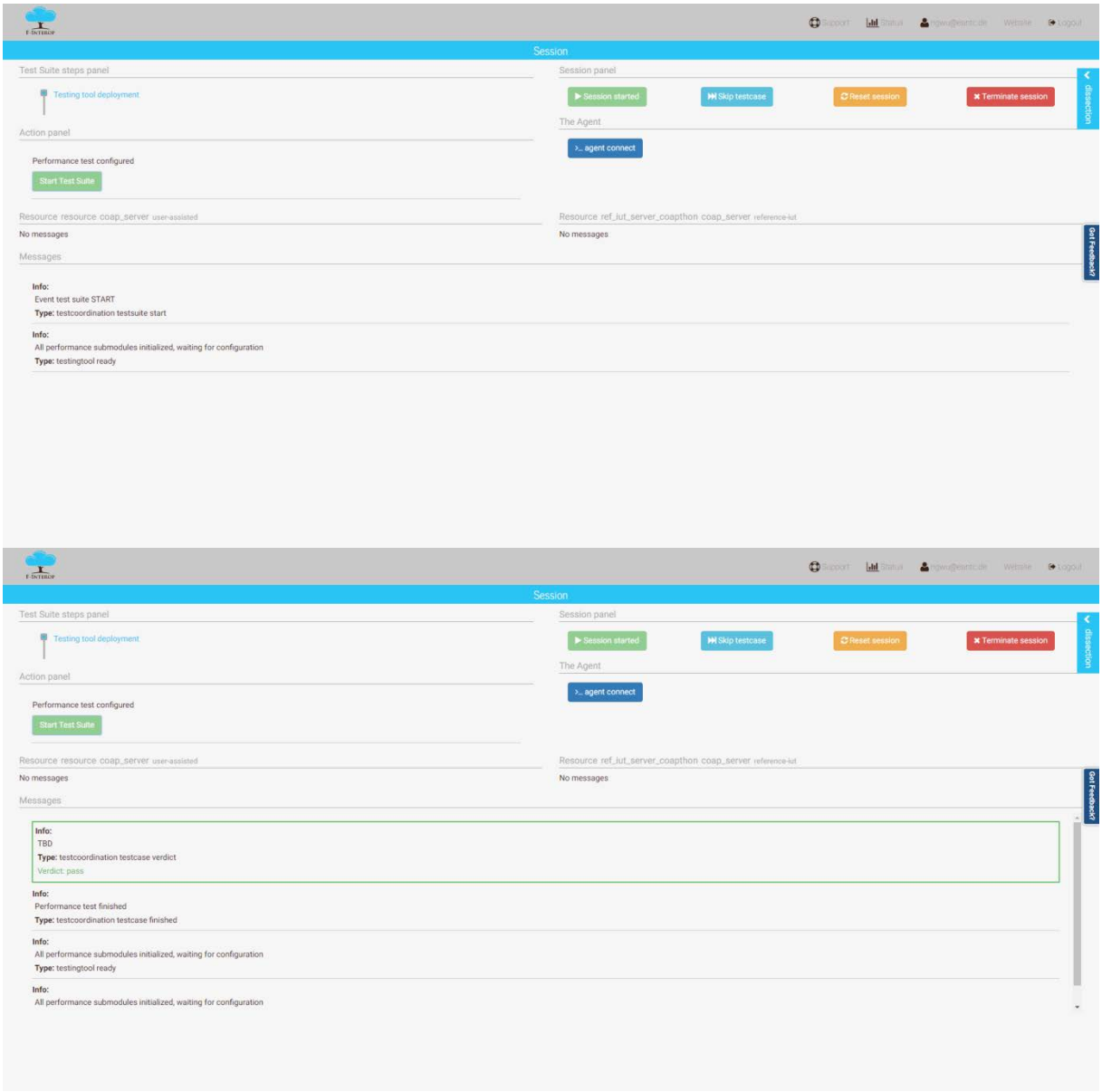

Figure 13 – Test and Verdict

The test has started and will stop on either error, user input (“Terminate session”) or upon completion. After the test has been successfully, the UI will present you the verdict of your current test session.