HORIZON 2020

Information and Communication Technologies

Integrating experiments and facilities in FIRE+

Deliverable D3.5
Performance Test Tools Final 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:	**31/08/2018**
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.


Co-funded by the European Union


Co-funded by the Swiss Confederation

Document properties

Responsible partner	EANTC
Author(s)/editor(s)	Chika Ngwu
Version	0.6
Keywords	Remote testing, Online testing, Network testing, Performance testing, Scalability testing, Energy efficiency testing, Internet of Things (IoT).

Abstract

The deliverable D3.5 describes the 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 CoAP Server.

The tool also provides means to monitor and analyze traffic behavior.

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
CS	CoAP Server Module
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
DUT	Device under Test
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
GUI	Graphical User Interface
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
IUT	Implementation under Test
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
SC	Subcomponent(s)
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
UI	User Interface
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

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)
- The energy consumption impact of the tested device under given traffic load.

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 *D3.5 – Performance Testing Tools Final Iteration* is the description of the second 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 CoAP Server and the required key enablers needed.

1.2.3.2 Deliverable Methodology

This document extends the information presented in Deliverable *D3.2 – Performance Tools First Iteration*, where the design principles of the Performance Test Tool CoAP Client Emulation and a detailed description of each component of the tool have been presented. Deliverable D3.5 focuses on the newly implemented CoAP Server and achievements integrated in the Performance Test Tool since the first iteration.

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 an emulated CoAP server and generate traffic 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.

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 Server Emulation
- Passive Monitoring

These components are described in detail in Section 3.

The CoAP Server Emulation module is the core component, that focuses on testing the CoAP Client Implementation of the User and managing communication between the testing tool and the VizTools/Result Store. It provides the necessary Resources and routines to test the stability of an IoT Device, whereas the Passive Monitoring module provides the means to monitor and analyse Layer 2-4 network traffic.

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 testing tools provided by the F-Interop platform e.g. interoperability testing tools.

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:

- performance_testing_tool_coap_server (as version-less name)
- performance_testing_tool_coap_server-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_testing_tool_coap_server/ in the Session Orchestrator deployment. Note that the in the future releases of the F-Interop platform.

See the Annex 6.1.1 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).

In detail, the orchestration process involves following steps:

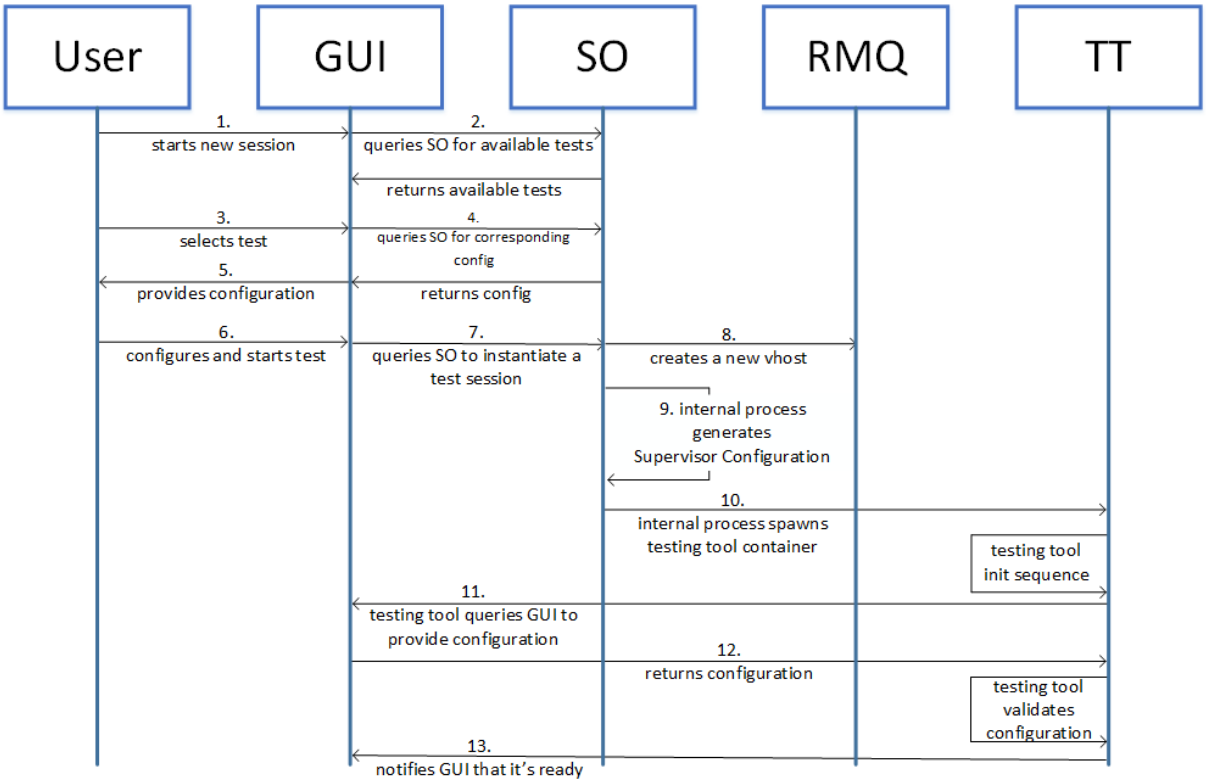

Figure 1 Message flow before test execution

1. The User starts a new test session.
2. The GUI queries the SO for the list of available tool types
3. The User selects the desired tool type (in our case - performance) and implementation (CoAP Server Emulation).
4. The GUI queries the SO for the configuration file (index.json).
5. The GUI generates the configuration page for the selected tool type using the UI template provided in the configuration file.
6. The User fills in the configuration values and starts the test.
7. The UI requests the SO to create a new test session.
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 "performance_testing_tool_coap_server". 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 GUI sends the configuration file from the initial request to create session (Step 6) 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. During session execution, the UI can request the teardown of the test session by sending a PATCH REST request to the Orchestrator with a “{action: “terminate”}” payload.

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

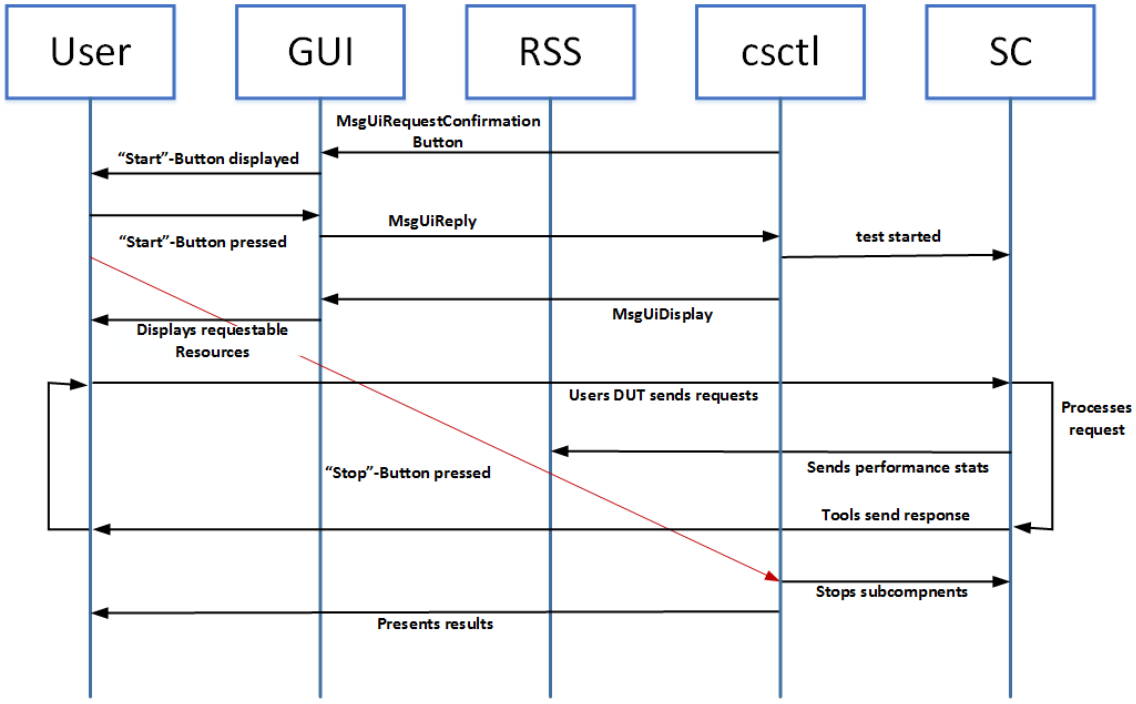

Figure 2 Message flow during test execution

After the performance testing tool coap server instance for a session has been spawned and configured, it will initiate the test execution. The CoAP Server “sends” a Confirmation Button to the GUI to be displayed for the User, to start the test execution. This button is not available if a subcomponent fails to start or to configure itself. All subcomponents, as they are listening to the same AMQP vhost, will then receive the confirmation that has been sent by the user via GUI and start their corresponding routines.

The CoAP Server will send a list of resources that can be reached by the IUT/DUT to be tested. Sending corresponding requests to these resources will trigger increasing block wise responses, timeout-based testing routines, request denying resources or resources that will fuzz message ids. It will also send another Confirmation button to the GUI, that will act as a test execution stopping mechanism.

The passive monitoring module on the other hand will immediately start to monitor and analyse both inbound and outbound network traffic. The analysis contains checks for following parameters:

- Ethernet Frame Analysis
- IPv4/IPv6 Analysis
- TCP/UDP Analysis
- CoAP Analysis

During these routines the tools will gather and send statistics to both VizTools and RSS, for monitoring and result storage.

If the previously mentioned “Stop-Button” is pressed, the GUI will send another confirmation and all subcomponents will cease all activity and will begin to start their shutdown routines.

The CoAP server will send the final results to the GUI and RSS for the User to view immediately after test and in his/her test overview at later times.

2.4 Communication between modules

The communication between all modules within a test session, including the internal communication between the CoAP Server 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.

Figure 3 provides an example of the message format.

```
- - - - - - - - - - - - - - - - - - - - -
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 3 Example of the message format

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

Table 1 Messages used by the Performance Test Tool

Message type	Direction	Description
performance.heartbeat	SC > CS	Periodic status reports from subcomponents to the CoAP Server.
ui.user.all.display	CS > GUI	Sends data (verdict, buttons, etc..) to the GUI
ui.core.session.get.request	CS > GUI	Provides test configuration to the performance testing tool
viztool-grafana.init.request	CS > VT	Instruct the VizTools to start their init routine
viztool-grafana.set_dashboard.request	CS > VT	Requests a Dashboard URL from the VizTools
viztool-grafana.write_data	CS > VT	Transmit the performance test statistics to the visualization component
testsuite.testcase.finished	CS > SC	CoAP Server notifies Submodules about the end of test
results_store.session.report.save.request	CS > RS	Requests RSS to handle sent results

2.5 Test Tool Configuration

The test configuration for the performance testing tool coap server is provided by the GUI and transmitted via the configuration key in the MsgUiSessionConfigurationReply message. By sending the MsgUiRequestSessionConfiguration message to the GUI, it will respond with the configuration.

Compared to the performance testing tool coap client emulation, the amount of configuration options has drastically decreased.

Here is an overview on what the configuration contains:

```
{
"static":
    {
    "coapserver.port":"5683",
    "coapserver.destination":"californium.eclipse.org"
    },
"initial":{},
"segments":[]
}
```

Table 2 Configuration Parameters

Name	Type	Description
coapserver.destination	string	Address (IPv4, IPv6 or hostname) of the target CoAP Client. Used by both CoAP server to direct the certain requests to, but also by the passive monitoring module to identify the test traffic streams
coapserver.port	int	Port number of the CoAP server. Can either be set by the user or left at default value.

The testing tool informs the User via the GUI on how to reach the session's instance and which resources to be reached.

3 Performance Test Tool Modules Functionalities

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

3.1 CoAP Server

The CoAP Server Emulation module 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.

It is also the module that allows emulating CoAP-conformant traffic. The tool component is based on the aiocoap Python library. Periodically sending CoAP pings to the IUT while receiving and processing incoming requests, collecting the statistics on the responses respective errors, it provides the test suite with the ability to test and analyse the performance of the DUT/IUT.

Table 3 CoAP Server Statistics

Name	Units	Description
coapserver.rtt	ms	Transaction response time, i.e. time between sending a request and receiving a response or error.
coapserver.req_attempt	trans./s	Rate of attempted transactions
coapserver.req_success	trans./s	Rate of successful transactions
coapserver.req_failed	trans./s	Rate of failed transactions, i.e. an error response was received from the client
coapserver.req_timeout	trans./s	Rate of requests timed out, i.e. a response was not received within defined timeout interval

3.2 Passive Monitoring

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

Table 4 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 Conclusion

Although the project is coming to an end, we strive to improve our software.

Thus, we will continue fixing bugs, adding new features to increase the amount of test methodologies and support both F-Interop Contributors and Users.

5 References

F-Interop Message API –

https://gitlab.distantaccess.com/f-interop-contributors/utils/blob/master/messages.py

6 Annex

6.1 Configuration Files

6.1.1 Session Orchestrator Configuration Files

This configuration template file is responsible for the instantiation of the testing tool container and required service containers.

supervisor.conf.j2:

```
; A user can write comments here to document how the processes are launched and interact with
; each others.
; amqp_url pattern : amqp://user:password@host/virtual_host
; {{ my_variable|default('my_variable is not defined') }}

; DOCKER PARAMS:
; --rm : Automatically remove the container when it exits
; --name : Assign a name to the container
; --privileged=true : Allows processes to create tun and modify network params
; --sysctl net.ipv6.conf.all.disable_ipv6=0 : ipv6 must be enabled testing tool's agent

; IMPORTANT
; put explicit command at the end (e.g. supervisord --nodaemon --configuration supervisor.conf)

[program:{{ session }}|testing_tool]
stopsignal=TERM
killasgroup=true
autostart=false
stdout_logfile = %(here)s/logs/{{ session }}-testing_tool-stdout.log
stderr_logfile = %(here)s/logs/{{ session }}-testing_tool-stderr.log
command = docker run
    --cap-add=NET_ADMIN
    --env FINTEROP_PERF_AMQPURI={{ amqp_url }}
    --env FINTEROP_PERF_EXCHANGE={{ amqp_exchange }}
    --rm
```

```
    --privileged=true
    --name="session_{{ session }}-performance_testing_tool_coap_server"
    performance_testing_tool_coap_server
    supervisord --nodaemon --configuration supervisor.coapserver.conf

[program:{{ session }}|service_viztool_grafana]
stopsignal=TERM
killasgroup=true
autostart=false
stdout_logfile = %(here)s/logs/{{ session }}-service-viztool-grafana-stdout.log
stderr_logfile = %(here)s/logs/{{ session }}-service-viztool-grafana-stderr.log
command = docker run
      --env AMQP_URL={{ amqp_url }}
    --env AMQP_EXCHANGE={{ amqp_exchange }}
    --env VIZTOOL_URL={{ service_viztool_grafana_url }}
    --env GF_SERVER_ROOT_URL={{ service_viztool_grafana_url }}
    --rm
      --name="session_{{ session }}-service-viztool-grafana"
      -p {{ service_viztool_grafana_port }}:3000
      service-viztool-grafana

[program:{{ session }}|service_results_store]
stopsignal=TERM
killasgroup=true
autostart=false
stdout_logfile = %(here)s/logs/{{ session }}-service-results-store-stdout.log
stderr_logfile = %(here)s/logs/{{ session }}-service-results-store-stderr.log
command = docker run
    --env AMQP_URL={{ amqp_url }}
    --env AMQP_EXCHANGE={{ amqp_exchange }}
    --env RS_AMQP_URL={{ rs_amqp_url }}
    --env RS_AMQP_EXCHANGE={{ rs_amqp_exchange }}
    --rm
    --name="session_{{ session }}-service-results-store"
    service-results-store
```

6.1.2 Dockerfile

This Dockerfile is used to create an image that will later be used to create a container for a testing session.

Dockerfile.coapserver:

```
FROM ubuntu:18.04
MAINTAINER ngwu@eantc.de

RUN apt-get update -y -qq && apt-get -y -qq install apt-utils software-properties-common
RUN apt-get -y -qq install python3 python3-dev python3-setuptools python3-pip supervisor nano sudo libpcap-dev less iproute2
RUN python3 -m pip install --upgrade pip
RUN python3 -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 1
ENV FINTEROP_PERF_THREADS 4

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

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

6.1.3 Internal Supervisor Configuration

This internal supervisor configuration file is responsible for starting the test tool after instantiation.

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

6.1.4 F-Interop GUI template

This template is used by the GUI to provide an environment for the user to configure a performance test session.

Index.json:

```
{
  "_type": "testsuite.manifest",
  "testing_tool": "CoAP Server 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 server"
      ]
  },
  {
    "field_name": "coapserver.destination",
    "description": "Address (IPv4, IPv6 or hostname) of the target CoAP client",
    "type": "text",
    "value_default": "californium.eclipse.org",
    "mandatory": true,
    "timeline": false
  },
  {
    "field_name": "coapserver.port",
    "description": "Port number of the CoAP server",
    "type": "int",
    "value_default": "5683",
    "value_min": "1024",
    "value_max": "65535",
    "mandatory": true,
    "timeline": false
  },
  {
    "field_name": "coapserver.timeout",
    "description": "Request timeout in milliseconds",
    "type": "int",
    "value_default": "1000",
```

```
      "value_min": "0",
      "value_max": "1000000",
      "mandatory": true,
      "timeline": false
    }
  ],
  "ansible_playbook": "ansible/main.yml",
  "owner": "F-Interop",
  "mantainer": "Chika Ngwu",
  "mantainer_email": "ngwu@eantc.de"
}
```


6.2 Example of a Test Session

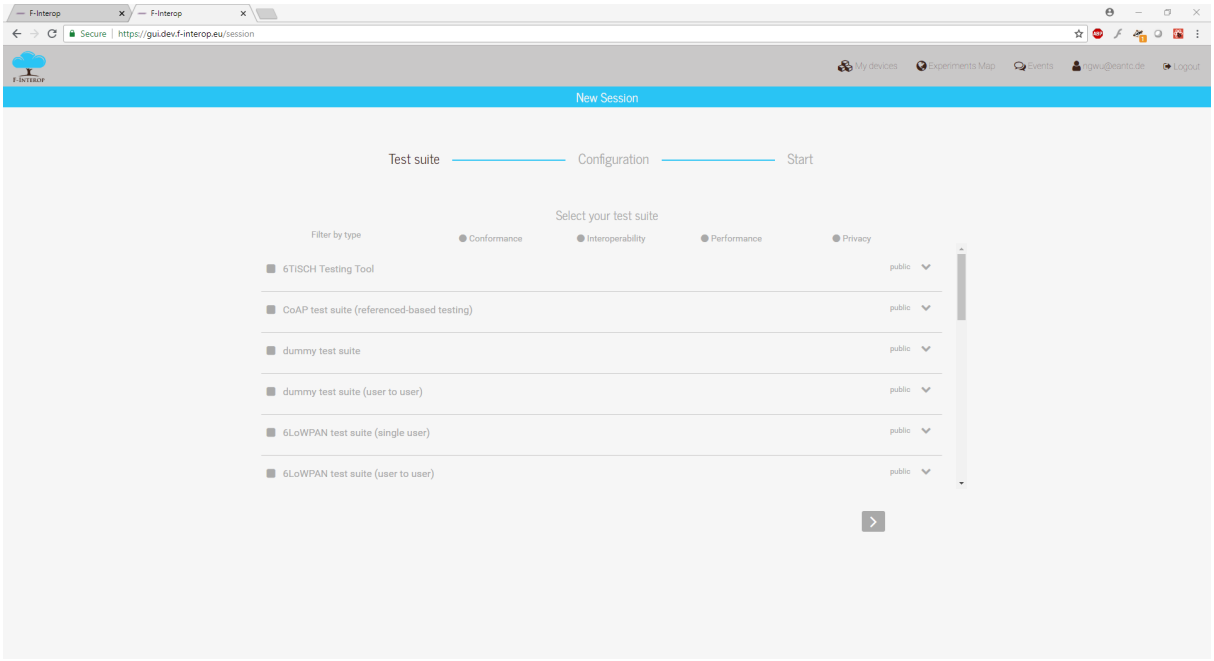

Figure 4 Test Selection 1

After starting a new Test-Session you are greeted with a list of all F-Interop tests.

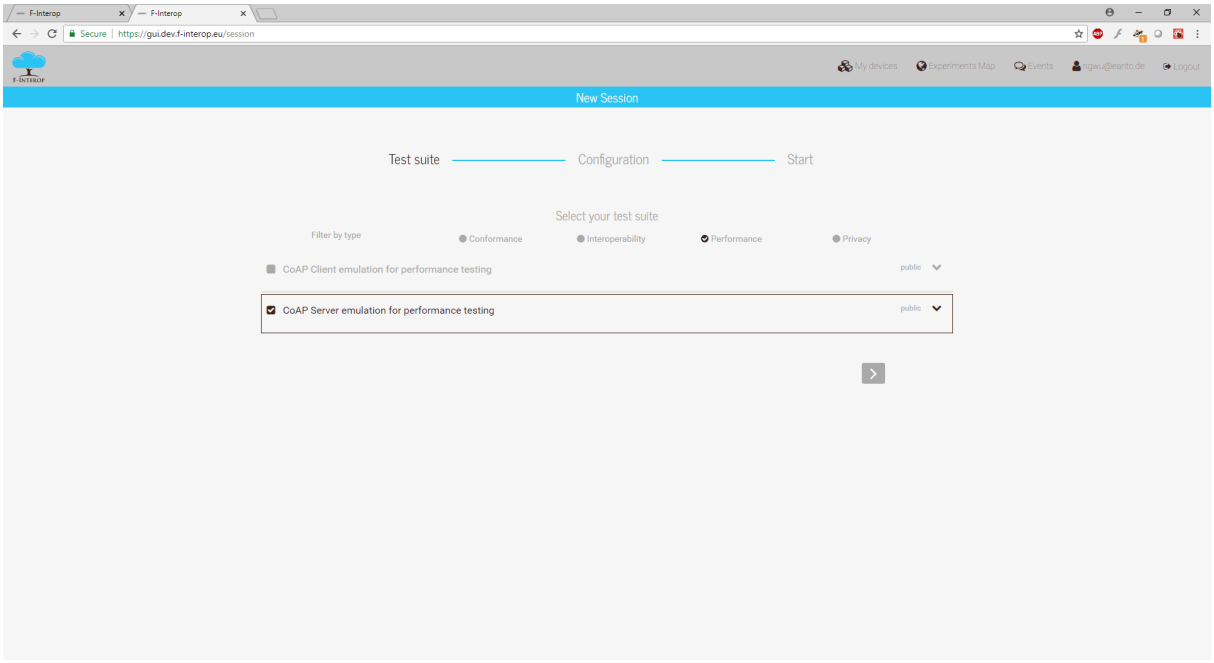

Figure 5 Test Selection 2

Sort the Tests by Tag “Performance” and select “CoAP Server emulation for performance testing”

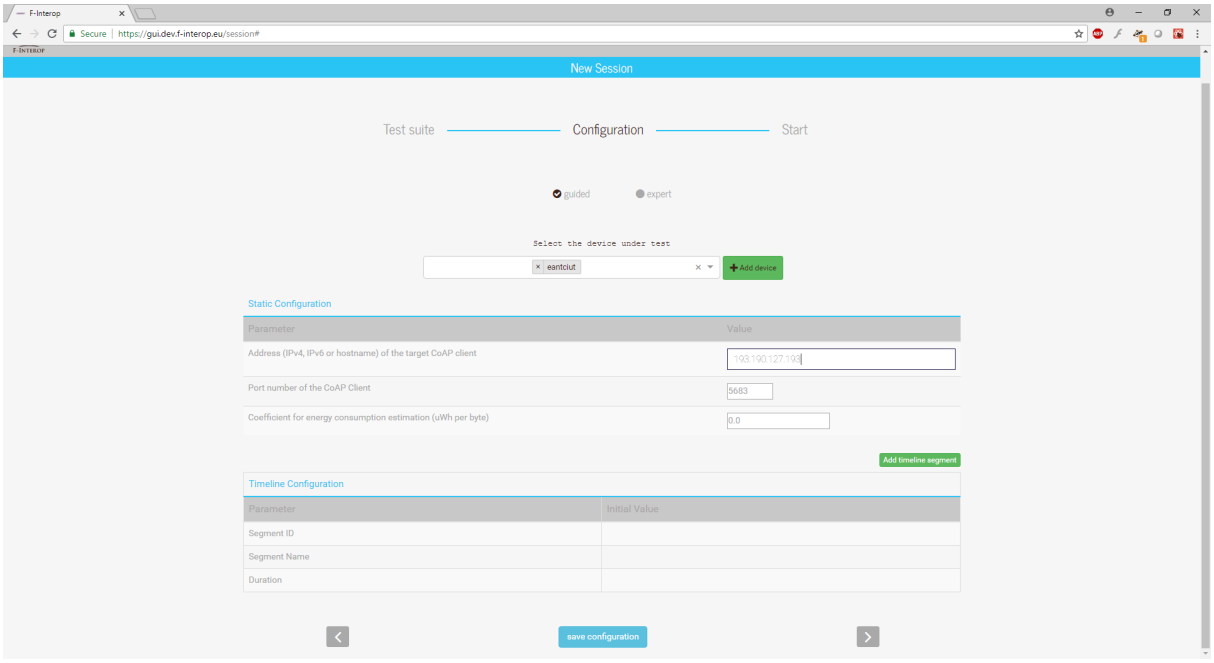

Figure 6 Basic configuration

Configure the test by adding Information about your DUT (DUT’s IP, Port and Power consumption) and save the configuration by clicking on the corresponding button.

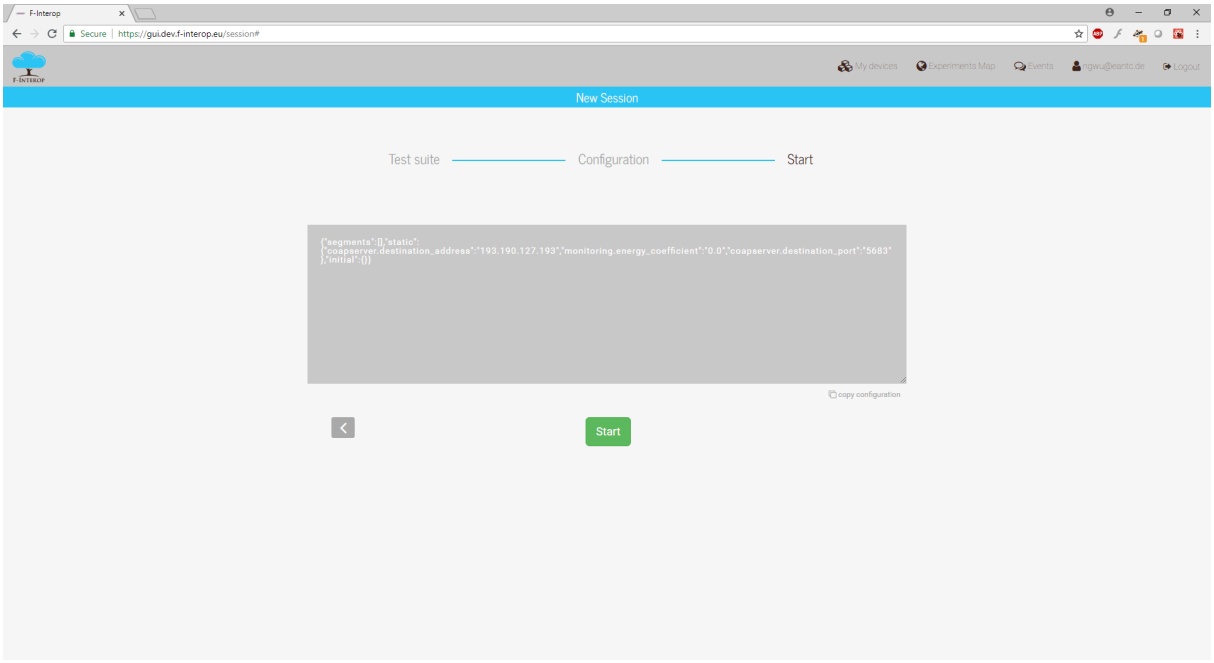

Figure 7 Configuration as JSON

As soon as the configuration has been set, you have the possibility to verify the configuration and save it in JSON for later reusing.

Press the “Start”-Button to start the Initialization-Phase!

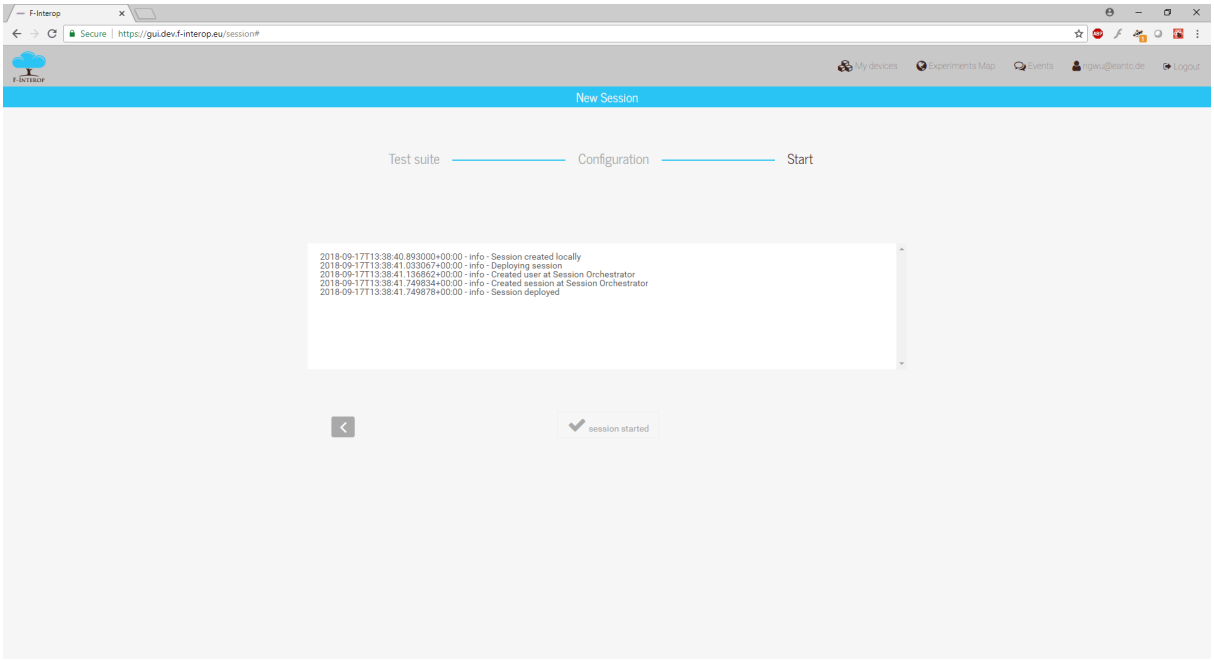

Figure 8 Session Instantiation

The Session Orchestrator is now spawning your Session.

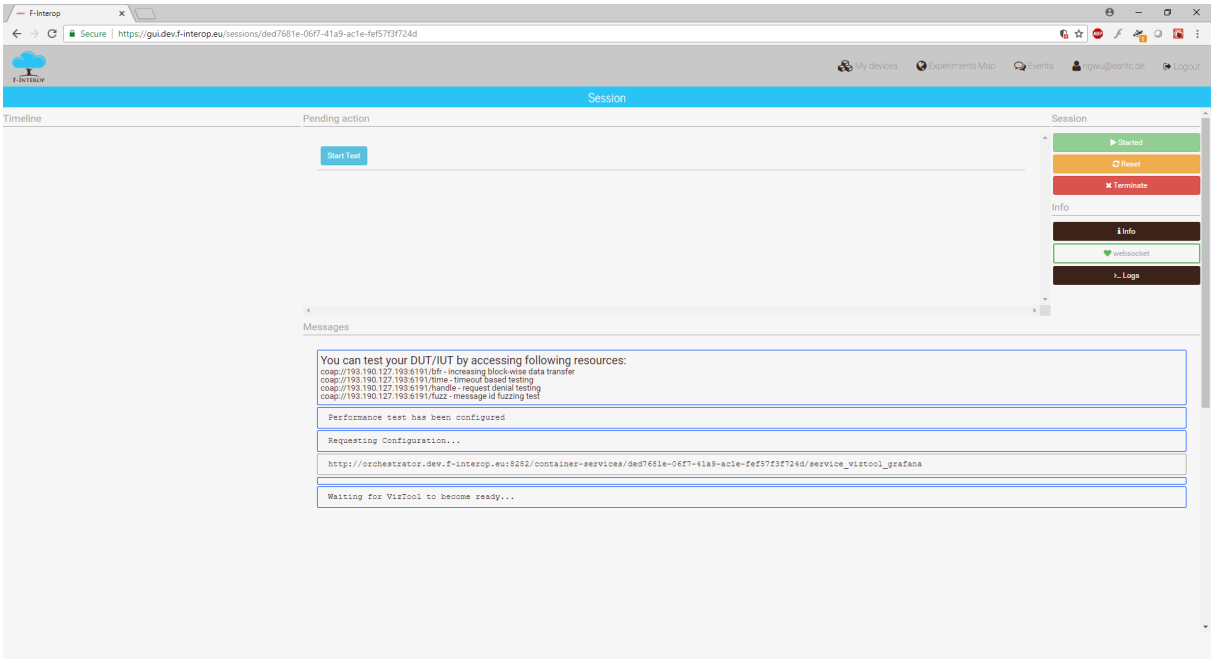

Figure 9 Testing Tool Initialization

The Testing Tools have passed their Initialization-Phase and are now ready to be started.

By pressing the “Start”-Button, the Subcomponents will begin with the Testing-Phase.

As a CoAP-Sever based test can’t be fully automated, you will need to configure your device to send requests to the URIs provided by the server.

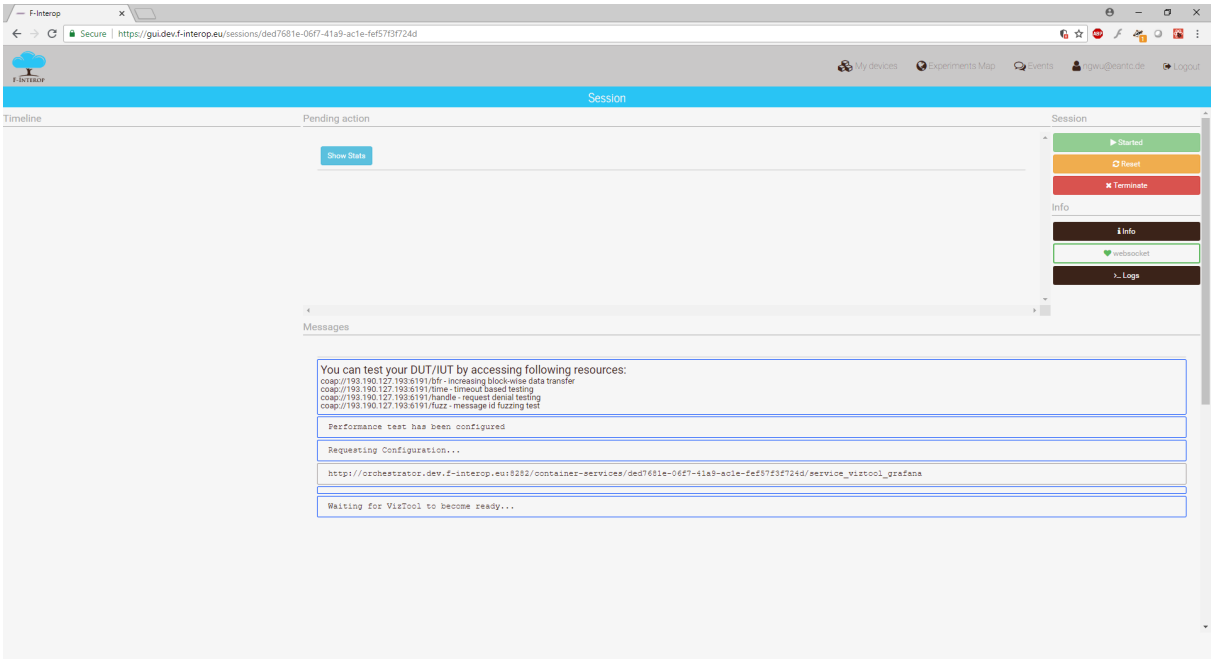

Figure 10 Started Test

After the test has started, you will have the Option to either view intermediate Statistics or to stop the Test.

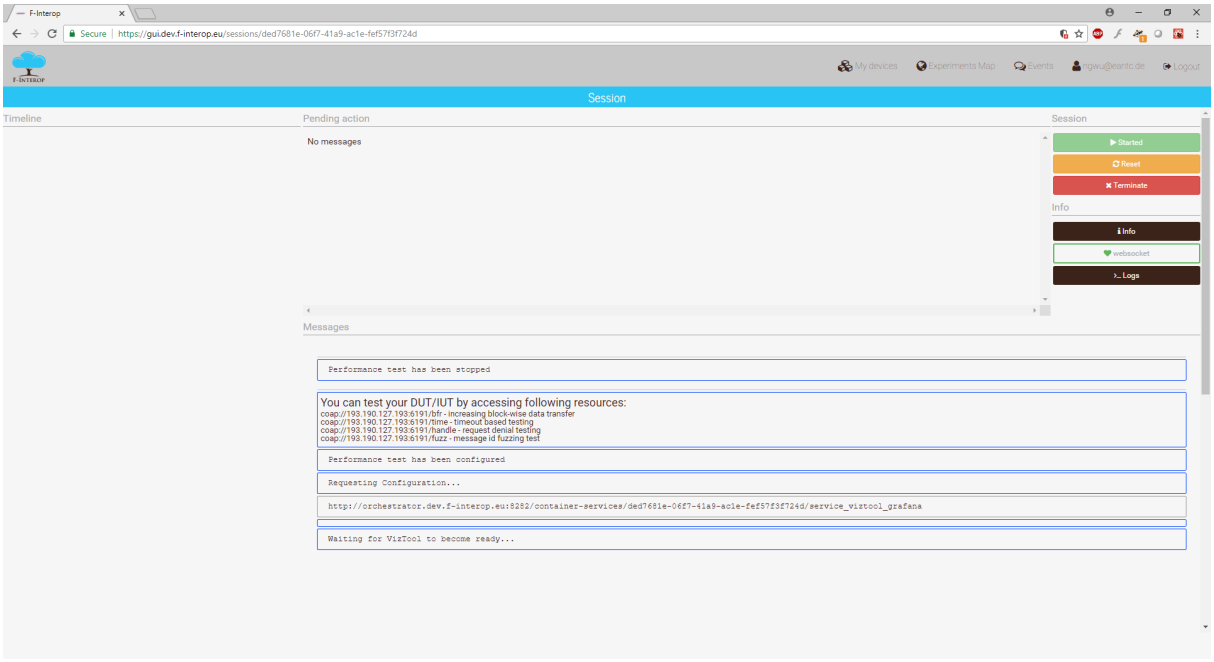

Figure 11 Stopped Test

After pressing the “Stop”-Button, the test will unconfigure and shut down.

Any sent requests are not getting answered anymore.

You should receive the final results.