Project: H2020-ICT-688712

Project Name:

5G Applications and Devices Benchmarking (TRIANGLE)

Deliverable D3.5

Report on the implementation of testing framework Release 4

Date of delivery:	04/06/2019	Version:	1.2
Start date of Project:	01/01/2016	Duration:	36 months

Deliverable D3.5
Report on the implementation of testing framework Release 4

Project Number:	ICT-688712
Project Name:	5G Applications and Devices Benchmarking
Project Acronym	TRIANGLE

Document Number:	ICT-688712-TRIANGLE/D3.1
Document Title:	Report on the implementation of testing framework Release 4
Lead beneficiary:	Universidad de Málaga
Editor(s):	Universidad de Málaga
Authors:	Keysight Technologies Belgium (Michael Dieudonne), Keysight Technologies Denmark (Andrea Cattoni, German Corrales Madueño, Marek Rohr), Universidad de Malaga (Álvaro Martín, Almudena Díaz, Pedro Merino, Laura Panizo Jaime, Bruno García, Guillermo Chica, Maria del Mar Gallardo), Redzinc Services Limited (Jeanne Caffrey, Donal Morris, Ricardo Figueiredo, Terry O'Callaghan, Pilar Rodríguez), DEKRA Testing and Certification S.A.U (Carlos Cárdenas, Janie Baños, Oscar Castañeda, J.C. Mora), Quamotion (Frederik Carlier, Bart Saint Germain), TNO (Lucía D'Acunto, Piotr Zuraniewski, Niels van Adrichem
Dissemination Level:	PU
Contractual Date of Delivery:	30/10/2018
Work Package Leader:	Universidad de Málaga
Status:	Final
Version:	1.2
File Name:	TRIANGLE_Deliverable_D3.5_v1.2 FINAL

Abstract

This deliverable provides the description of the TRIANGLE testbed compiling the work done during the four releases in which the development process was divided. It presents the features provided by the TRIANGLE Portal and all the components integrated into the testbed.

Keywords

Architecture, workflow, deployment, orchestration, test case, Portal, measurements tools, RAN, EPC, SDN, UEs

Document history

V1.0	Initial release of the document
V1.1	Numbering problem in the table of contents has been fixed Section 10.5 describing DASH client has been removed as it is not a component of the testbed. It was used to demonstrate the functionaly of the features provided by the TNO extension. Table 5 has been updated with the extensions for Video quality evaluation and the MEC extension. Extensions coming from the open calls have been clearly identified in sections 10 and section 11. The testbed release integrating these extensions has been also indicated. MEC User Manual added in Annex 11 Figure 9 has been updated with the rest of the main components of the testbed architecture. Android and iOS footnote has been added.
V1.2	Moved Section 11.4 to Section 10.5 to clarify the technology was implemented in Rel 4.

Executive summary

This document is the fifth deliverable of WP3. WP3 is responsible for the development of the testing framework in TRIANGLE. The testing framework covers all the software that configures and manages the testbed infrastructure.

The document is a compilation of the latest version of the testbed plus information about previous releases (reported in D3.1, D3.2, D3.4). In particular, the document describes the final implementation of the TRIANGLE testbed including the final version of the Portal, the final set of features supported by the testbed, the reports auto generated after post-processing the measurements collected during the testing campaigns and the control interface developed to manage each of the components of the testbed.

It is a self-contained document serving as a general guide to all the features supported by the TRIANGLE testbed. The focus of the document is to provide a clear understanding of how the testbed has been implemented to deliver the testing and certification services offered by TRIANGLE.

Contents

List of Figures

List of Tables

List of Abbreviations

AUT	App Under Test
AP	Access Point
APNet	Antennas, Propagation and Radio Networking
BER	Bit Error Rate
BLER	Block Error Rate
BS	Base Station
CAPEX	CApital EXpenditure
CDMA	Code Division Multiple Access
CFO	Carrier Frequency Offset
CO	Confidential
CP	Cyclic Prefix
CR	Cognitive Radio
CRS	Cognitive Radio Systems
CSI	Channel State Information
CSMA	Carrier Sense Multiple Access
C2X	Car-to-Anything
D	Deliverables
DL	Downlink
D2D	Device-to-Device
DMRS	Demodulation reference signal
DRX	Discontinuous Reception
DTX	Discontinuous Transmission
DUT	Device Under Test
EIRP	Effective Isotropic Radiated Power
EIT	European Institute for Innovation and Technology
E2E	End-to-End
EVM	Error Vector Magnitude
FDD	Frequency Division Duplex
FD-MIMO	Full-Dimension MIMO
FEC	Forward Error Correction
FR	Frequency Response
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HARQ	Hybrid Automatic Repeat Request
ICI	Inter-Carrier Interference
ICT	Information and Communications Technology
IEEE	Institute of Electrical and Electronics Engineers
IMT	International Mobile Communications
IP	Intellectual Property
IPR	Intellectual Property Rights
IR	Internal report
ITU	International Telecommunication Union
ITU-R	International Telecommunication Union-Radio
KPI	Key Performance Indicator
LAN	Local Area Network
LOS	Line of Sight
LTE	Long Term Evolution
LTE-A	Long Term Evolution-Advanced
L2S	Link to System
M	Milestones
Mbps	megabits per second
Mo	Month
MA	Multiple Access
MAC	Medium-access Control
MGT	Management
MIMO	Multiple-Input Multiple-Output
MMC	Massive Machine Communication
M2M	Machine to Machine
MSE	Mean Squared Error
NLOS	Non line of Sight
OFDM	Orthogonal Frequency Division Multiplexing
OPEX	Operational Expenditure
PA	Power Amplifier
PAPR	Peak-to-Average-Power-Ratio
PC	Project Coordinator
PHY	Physical Layer

PU	Public
QAM	Quadrature Amplitude Modulation
QAP	Quality Assurance Plan
QMR	Quarterly Management reports
QoE	quality of experience
QoS	Quality of Service
RACH	Random Access Channel
RAN	Radio Access Network
RAT	Radio Access Technology
RF	Radio Frequency
R&D	Research and Development
RRM	Radio Resource Management
RTD	Research and Technological Development
RTT	Round Trip Time
SDR	Software Defined Radio
SINR	Signal to Interference and Noise Ratio
SRS	Sounding Reference Signal
T	Task
TDD	Time Division Duplex
TDMA	Time Division Multiple Access
TRX	Transmitter
TTI	Transmission Time Interval
UE	User Equipment
UL	Uplink
UMTS	Universal Mobile Telecommunications System
USRP	Universal Software Radio Peripheral
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-anything
WCDMA	Wide Code Division Multiple Access
WLAN	Wireless Local Area Network
WP	Work Package
WPAN	Wireless Personal Area Networks

1 Introduction

1.1 Why TRIANGLE

According to Cisco [1], global mobile data will increase from 12 Petabytes per month in 2017 to 77 Petabytes per month in 2022. It is interesting to highlight that fourth-generation (4G) traffic exceeded third-generation (3G) traffic for the first time in 2015The fast pace of the standardization process adds more complexity to this picture. In current network deployments, while evolving towards 5G, 4G LTE coexists with 2G and 3G technologies. In conjunction with the set of new features added in each 3GPP release, the potential combinations of network configurations grow exponentially for operators.

Despite that fact, mobile subscribers are increasingly focused on applications and keep demanding high levels of quality everywhere. Unfortunately, it is usual to find performance and connectivity problems that hugely affect the user experience.

In this context, the primary objective of the TRIANGLE project is to promote the testing and benchmarking of mobile applications and devices in Europe as the industry moves towards 5G. It also provides a pathway towards certification of qualified mobile developments in Europe.

1.2 The TRIANGLE Testbed and its Users

Figure 1 provides an overview of the TRIANGLE testbed with emphasis on the testing workflow.

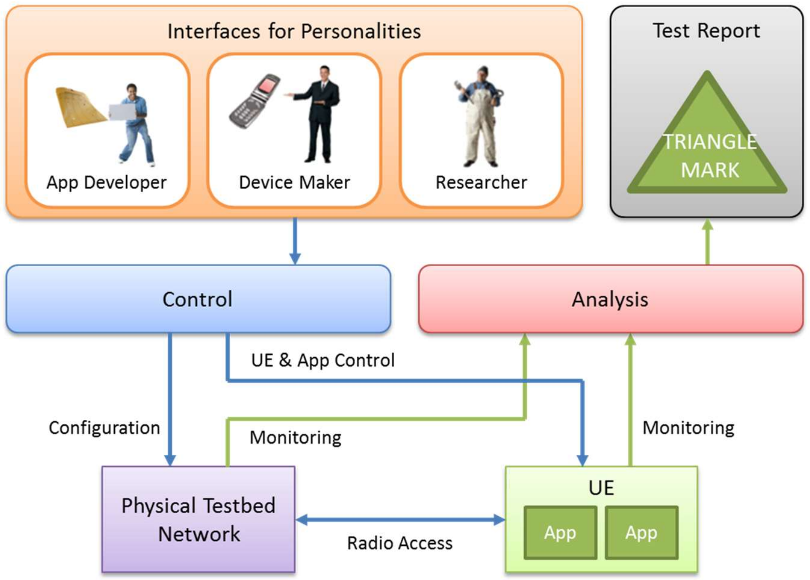


Figure 1 Overview of the TRIANGLE testing workflow

The point of entry to the testbed are the interfaces offered to the end users. For device makers and researchers the interface is based on TAP (see section 4) and a set of reference templates. For app developers the interfaces is the Portal (described in Section 4). These interfaces have been designed to adapt to the user's needs. At the same time, the complexity of the testing is wrapped into high-level scenarios, which prevent users from having to deal with the full set of configurable parameters. These high-level scenarios are based on 5G testing scenarios identified in Section 3 of Deliverable 2.1.

Based on this selection, the testbed configures the physical components and schedules the execution of the tests and the collection of measurements required to check the performance of the features of the application or device under test. The interaction with the apps under test is also automated, i.e., it automatically carries out the user interactions to be analysed.

As output, the testbed provides detailed reports to the user. These reports are generated based on the information collected by the reporting and automation tools running on both the testbed and the device.

The testbed support four types of test campaigns:

- **Certification campaigns**. A certification campaign executes all the test cases applicable to a device under test or an app under test. After filling a questionnaire, the test cases are automatically selected and no additional configuration is needed.
- **Standard campaigns**. When users opt for standard campaigns, they can select among the test cases defined for the uses cases and features supported by the app. The user can also configure the scenario and the device (for app testing) used during the test campaign. Remote screen and traffic capture are available during the execution of these campaigns.
- **Custom**. Users can define their own test campaigns providing a power shell script which replays the feature under test and can select the scenario and the device. Remote screen and traffic capture are available during the execution of these test campaigns.
- **Model-Based Testing**. The app user flows are generated automatically based on a formal description of the application under test. The method applied for the generation is described in Section 11.3.

1.3 Testbed High-Level Architecture

Figure 2 shows a more detailed view of the main functional blocks that make up the TRIANGLE testbed architecture. The architecture can be divided into several subsystems, whose role will be introduced briefly in the rest of this section. The figure also includes a note with the section in which each component is described in more detail.

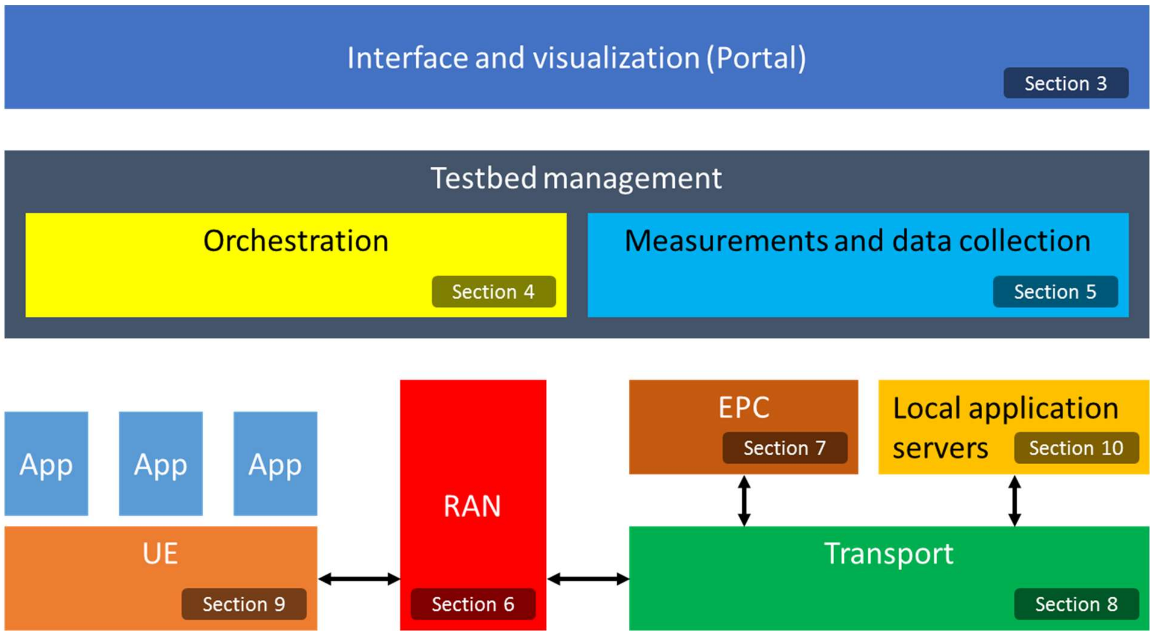


Figure 2 High-Level architecture of the testbed

1.3.1 Interface and Visualization (Portal)

The TRIANGLE portal is a user-friendly interface for remote interaction with the testbed. It provides a view of the testbed that is adequate for each user profile, hiding unnecessary complexity. The main purpose of the testbed portal is to prepare and run tests, and later review the results.

Tests are configured based on the selections performed by the users, e.g. which app to test, on which device, and on which high-level scenario the user wants to run the tests. These user inputs are processed and transformed into inputs for the different components of the underlying architecture.

A high-level scenario is an understandable term of the network conditions which are configured during the test case to reproduce conditions experimented in the high-level scenario selected. In other words, a high-level scenario is an abstraction of similar network configurations, e.g. “Vehicular” is a high-level scenario which compromises different configurations of the speed: 30 km/h, 60 km/h, 90 km/h and 120 km/h.

The portal stores all the campaigns and other user provided data, as well as the results obtained from the tests, so that test case results are traceable to their configuration, and can be repeated if needed.

The Portal is described in Section 3.

1.3.2 Orchestration

To run a test case, all components must be controlled by an orchestrator, which must coordinate their configuration and execution. In the TRIANGLE testbed, the Test Automation Platform (TAP), from Keysight, serves as coordinator responsible for configuring and running the tests. Each testbed component is controlled through a TAP driver, which serves as bridge between the TAP engine and the actual component interface. These drivers have been developed for the testbed.

The configuration of the different elements of the testbed network depends on the high-level scenarios selected by the users. The testbed translates these high-level scenarios into the specific configurations that TAP applies to each network component. In short, TAP controls the overall execution of each test case.

An integral part of testing apps is automating their execution, i.e. simulating the interactions of a user with the app. Quamotion automation tools provide the means to create sequences of user actions, and then replaying them on a testbed device.

To synchronize radio access and power consumption measurements, the orchestration components include a PTP based synchronization system.

These orchestration components are described in Section 4.

1.3.3 Measurement and data collection

A measurement is a value discovered by measuring, that corresponds to a property of something. To obtain measurements to compute the TRIANGLE Mark and other reports, the testbed provides several probes (both software and hardware) which extract the required measurements. Software probes running on the UE include DEKRA Agents and the TestelDroid tool from UMA. TRIANGLE also provides an instrumentation library for app developers, in order to provide additional measurements that cannot be extracted by other means. Hardware probes include a power analyzer connected to the UE to measure power consumption.

Measurements are collected and analyzed in order to calculate the key performance indicators (KPIs) associated to the features provided by the apps or devices, e.g. video streaming or VoIP calls. To facilitate the aggregation of measurements and KPIs, all measurement values are stored in a central OML server, which uses a PostgreSQL database server as backend.

All these measurement components are described in detail in Section 5.

1.3.4 RAN (Radio Access Network)

Radio access emulation plays a key role in the TRIANGLE testbed. RAN is provided by a UXM Wireless Test Set from Keysight, a mobile network emulator that provides state of the art test features. Some of its key features for testing include flexible Inter Cell Interference Coordination (eICIC) schemes, WLAN offloading, IMS/End to End VoLTE communications between multiple devices, and battery drain performance with flexible network and sleep mode settings.

Only two UEs can be connected at the same time to the UXM. To connect more devices, e.g. all the reference devices used in the testbed, RF switches are used.

The RAN emulator is described in section 6.

1.3.5 EPC (Evolved Packet Core)

To provide an end-to-end system, a commercial EPC from Polaris Networks, which includes the main elements of a standard core network: MME, SGW, PGW, HSS, and PCRF has been integrated. In addition, this EPC includes the EPDG and ANDSF components for dual connectivity scenarios. As an extension for this project, the S1 interface to interconnect the UXM with the EPC has been developed.

The EPC system is described in Section 7.

1.3.6 Transport

To emulate the transport network between the eNodeB and the EPC, TRIANGLE uses an SDN deployment that provides features such as traffic prioritization, separation of data and control plane traffic, and transparent mirroring of selected traffic flows. Moreover, the testbed offers the possibility of integrating artificial impairments in the interfaces of the core network and the application servers.

Finally, the testbed includes an over-the-top-content enabler, the VPS Engine, which can be used by third-party applications to configure certain aspects of the SDN deployment and the EPC policies to request a specific Quality of Service (QoS).

The transport network components are described in Section 8.

1.3.7 App(s)

The TRIANGLE testbed can run apps from app developers in a set of reference devices. Support for running and automating apps is explained in Section 4, as part of the orchestration of the testbed.

1.3.8 UE (User Equipment and Accessories)

As with apps, the testbed allows testing the apps provided by users in (i) reference devices and (ii) new devices that will be tested under the conditions set by the testbed. In both cases, devices must be physically connected to the testbed. To fully control the radio conditions configured at the UXM Wireless Test Set, the RF connection must be conducted through cables. Also, to properly analyze power consumption, the device must be powered directly by the N6705B power analyzer.

In addition, the device should provide some control and automation interface that can be used from the testbed orchestration tools. For instance, in the case of Android, this means a USB connection to a testbed computer to support connection through the adb tool.

The integration of a set of reference Android devices is described in Section 9.

1.3.9 Local application servers

The testbed includes a virtualized infrastructure (see Section 10) to suppor the local deployment of services. Allocating these servers in the testbed ensures that measurements are not altered by the Internet connection between the testbed and the remote server, which cannot be controlled by the testbed.

1.4 Testbed workflow

Figure 3 provides the overall logic flow of the testbed from top to down.

The workflow of the testbed starts with the user defining, through the Web portal, the app/device under test, the features and the user behaviours, and the desired test procedure: certification or custom test. For custom tests, the user will also have to specify the high-level scenarios in which he or she is interested. In this context, user behaviours are sets of actions that represent a particular usage scenario of the application. For instance, login into a particular application, which will require pressing a button, entering a text in the username field, entering a text in the password field, and clicking the “Ok” button.

The testbed will transform these inputs into:

- Specific configurations of the network elements. For each configurable component, the testbed will select among a predefined pool of configuration files the ones that match the input given by the user. The resulting test plan to be executed will use the contents of these files to configure each component.
- User behaviours will be translated into a specific user flow control, which will be included in the test plan to automate the behaviour of the application during the test.
- KPIs associated with the features declared by the user. The test plan will take into account these KPIs to define the measurements that have to be collected during the test.

The orchestrator will conduct the execution of the test and the collection of the measurements, which will be stored in a common database for each test. The results will be analyzed and post-processed to calculate the KPIs, generate the test reports and (when applicable) deliver the TRIANGLE mark.

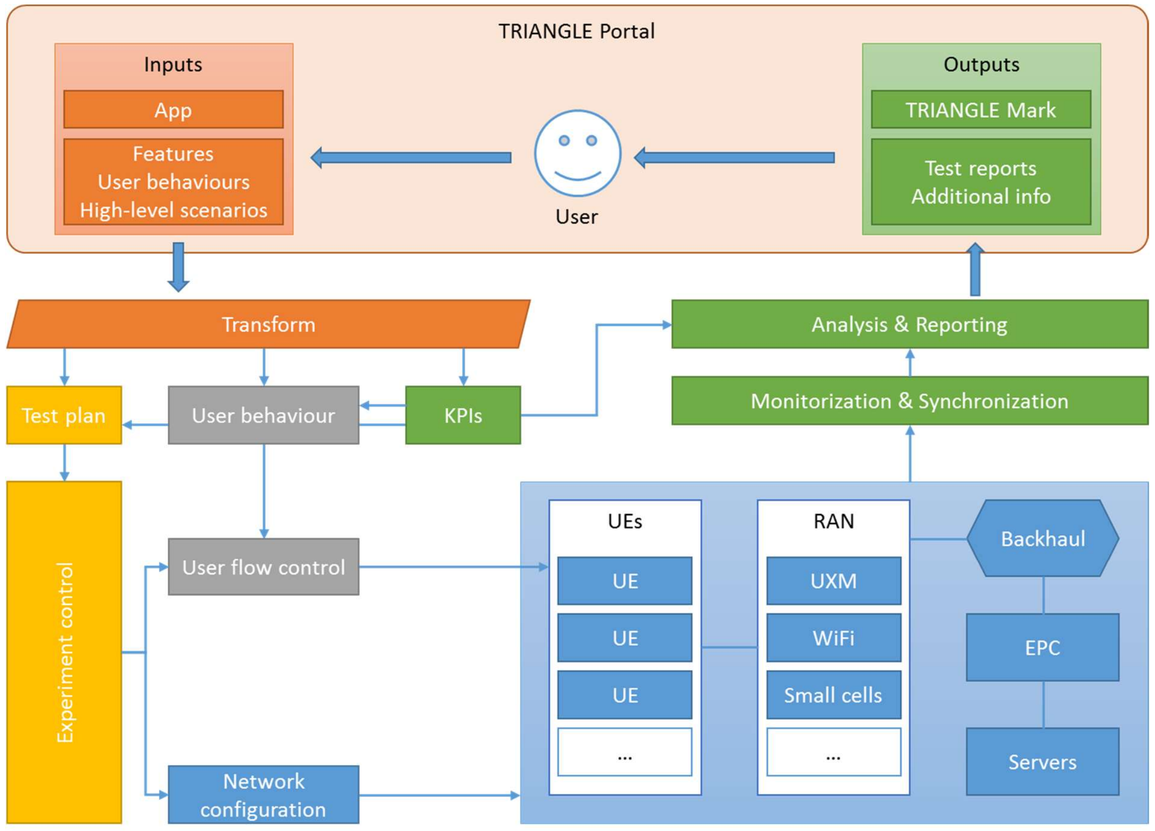


Figure 3 TRIANGLE testbed high-level workflow

1.5 TRIANGLE testbed releases

The TRIANGLE testbed has been continuously developed along the project. To control the testbed versions four releases (R1 to R4) have been produced. The following table summarizes the main features of the TRIANGLE testbed and the releases in which they were included, for a better understanding of its evolution.

Table 1 List of TRIANGLE features

Testbed features	**Components**	**Rel**	**Comments**
Testbed accessibility	Portal	R2	Main entry point to the testbed
	Web Reporting Tool	R2	Raw measurements visualizer
	Test Automation Platform (TAP)	R2	Testbed access for advanced users
	Booking System	R3	The booking system consists in a web calendar service used by the experimenters to book the TRIANGLE Portal Testbed.
	Scheduler	R4	Testing campaings are scheduled until the testbed is available. it is possible to queue several campaign executions at a time.
	PDF reporting	R4	This report provides a summary of the results obtained in all the test cases executed in each one of the domains.
Testbed usage	Standard campaigns	R2	When users opt for standard campaigns, they can select among the test cases defined for the uses cases and features supported by app. The user can also configure the scenario and the device used during the campaign
	Custom campaigns	R2	Users can define their own test campaign providing a power shell script which replays the feature under test and can select the scenario and the device.
	Researcher campaigns	R2	allows modification of the test templates and configuration of low level parameters
	Certification campaigns	R3	A certification campaign executes all the test cases applicable to the app based on the uses cases and the features supported by the app. The uses cases and the features are configured when the application is uploaded to the Portal. No additional configuration is needed.
	Model-based campaign	R4	Automatic generation of app user flows based on a formal model of the app under test
Results	Raw results	R2	Battery, cpu, memory, traffic, power consumption, logs, radio measurements, …
	KPIs	R3	Raw results are post-processed to compute the KPIs defined in D2.5
	MOS	R3	Each KPI is transformed into a syntethic MOS (See section 2.7.1)
	TRIANGLE mark (3 domains)	R3	The synthetic MOS is the aggregation of 3 domains: App Energy Consumption, Device Usage resources, App User Experience. After executing the tests a synthetic MOS per

			domain is obtained. The final TRIANGLE mark is obtained aggregating synthetic MOS scores for all the domains.
	TRIANGLE mark (6 domains)	R4	New domains: Reliability, Network Resources Usage, Network adaptation
	Additional results	R4	Videos, enhanced logs
Device automation	Quamotion Webdriver	R1	The Quamotion Automation Tools integrated as interface to interact with the apps and the devices under test.
	Powershell support	R3	App users flows are defined via powershell scripts
Monitoring tools	TestelDroid	R1	Android applications which enables traffic capture at the UE
	DEKRA Performance Tool	R1	Multiplatform performance monitoring tool: device resources (RAM, CPU, GPU), traffic generation, traffic statistics, reference applications
	Instrumentation library	R2	Defines a set of measurements points to correlate the measurements collected with the actions performed by the applications under test
	WLAN Access Point Automation	R4	A module to facilitate the integration and control of Wi-Fi access points in the TRIANGLE testbed.
Measurements post-processing and storage	ETL Framework	R3	MOS scoring per domain, aggregated MOS
	KPIs computation	R2	KPIs specified in D2.2 are computed after the execution of each test case and stored in the general database
	General database	R2	Main data base of the project
Backhaul	Emulated Impairments	R3	Core interfaces and servers connections
	SDN	R3	Servers connectivity
	VNF	R3	Servers deployment
	Openstack	R3	Servers deployment
	EPC (Polaris)	R2	Commercial EPC
Instruments	UXM	R1	LTE-A Pro base station emulator
	N6705B Power Analyzer	R1	Power Analyzer
Devices	Android devices	R1	Full automation: apps user flows, cpu, battery, ram, traffic, adb control, logcat access, power consumption, gps signals
	NB-IoT devices	R2	Ad-hoc automation depending of the capabilities exposed by the device

	iOS devices	R3	Partial automation: App user flow, internal log, traffic capture
Additional features	S1 Interface	R1	With this functionality the eNB Emulator is capable of connecting to commercial core networks
	S1 handover	R2	Vehicular scenarios also require handovers.
	Robotic arm	R2	The robotic arm platform meets the technical requirements derived from the Virtual Reality, Gaming and Augmented Reality test specification, mostly on the ability to move the device.
	GPS emulation	R2	Software defined radio solution for the generation of GPS signals
	Model based testing	R2	Model-based testing is a testing technique that uses a model of the app under test to automatically generate the app user flows.
	Remote pcap	R2	Enables the capture of traffic in any of the interfaces.
	Heterogeneous access	R3	LWIP support
	Remote screen	R3	VNC access to the app and device under test
	Measurements compesation	R4	Baseline calculation per domain for obtaining more accurate KPIs
	Error-handling	R4	Additional checks for ensuring the correct execution of the App Userflows during a test, Campaign Execution Retries, Loose Processes Handling
	Management console	R4	The Administration Console communicates with the Portal and the queue manager using their REST APIs, and is able to modify the contents of the execution queue, while displaying useful information about current or past executions and campaigns

The tables below list the featutes developed in TRIANGLE testbed in its different releases.

Table 2 TRIANGLE Release 1 features

Release	**Testbed features**	**Components**
R1	Device automation	Quamotion Webdriver
	Monitoring tools	TestelDroid
		DEKRA Performance Tool
	Instruments	UXM
		N6705B Power Analyzer
	Devices	Android devices
	Additional features	S1 Interface

Table 3 TRIANGLE Release 2 features

Release	**Testbed features**	**Components**
R2	Testbed accessibility	Portal
		Web Reporting Tool
		Test Automation Platform (TAP)
	Testbed usage	Standard campaigns
		Custom campaigns
		Researcher campaigns
	Results	Raw results
	Monitoring tools	Instrumentation library
	Measurements post-processing and storage	KPIs computation
		General database
	Backhaul	EPC (Polaris)
	Devices	NB-IoT devices
	Additional features	S1 handover
		Robotic arm
		GPS emulation
		Model based testing
		Remote pcap

Table 4 TRIANGLE Release 3 features

Release	**Testbed features**	**Components**
R3	Testbed accessibility	Booking System
	Testbed usage	Certification campaigns
	Results	KPIs
		MOS
		TRIANGLE mark (3 domains)
	Device automation	Powershell support
	Measurements post-processing and storage	ETL Framework
	Backhaul	Emulated Impairments
		SDN
		VNF
		Openstack
	Devices	iOS devices

	Additional features	Heterogeneous access
		Remote screen

Table 5 TRIANGLE Release 4 features

Release	**Testbed features**	**Components**
R4	Testbed accessibility	Scheduler
	Testbed usage	PDF reporting
	Results	TRIANGLE mark (6 domains)
		Additional results
	Monitoring tools	WLAN Access Point Automation
	Streamowl	Video quality-of-experience assessment extension
	MEC extension	MEC server and S1 breakout
	Additional features	Measurements compesation
		Error-handling
		Management console

2 TRIANGLE Testbed Architecture

The TRIANGLE testbed is composed of several interconnected hardware units, computers and virtual machines. All these components work together to provide the means to execute tests over apps or devices, and to provide test reports. On top of the infrastructure layer the project has developed a management frawework which fully automates the configuration, control and execution of the test cases specified in D2.5.

2.1 Testbed Infrastructure

Figure 4 shows an overview of the physical interconnections between the testbed components.

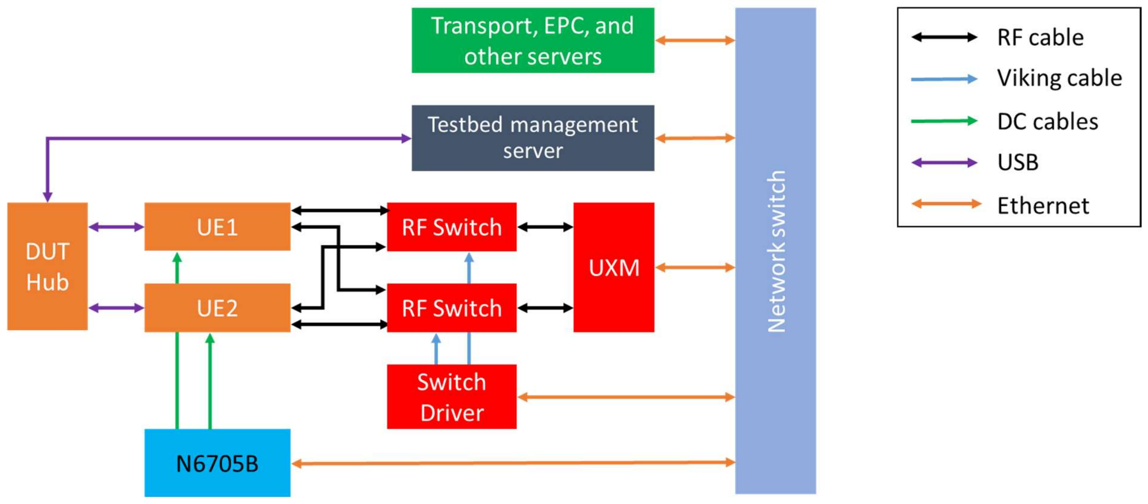


Figure 4 Testbed physical infrastructure

UEs (either reference devices for app testing, or a UE under test, when testing devices) are connected to the UXM mobile network emulator. The radio signal is not radiated (over-the-air); instead it is conducted through calibrated RF cables to the UE antenna connector. For testing purposes most UEs typically contain small antenna connectors which are hidden from the user. The UXM only support the interconnection of two UEs. To connect more devices to the same UXM, the testbed uses RF switches, controlled by a switch driver. The switches are placed in the RF connection between the UEs and the UXM, and the switch driver will select which RF connections (RF paths) to be routed to the UXM.

The UXM emulates all the network signalling and physical signals, as well the MIMO radio channel. All the protocol layers in the emulated network operate realistically as defined in the 3GPP test specifications and can be configured. Moreover, for testing purposes, the UXM instrument provides additional useful capabilities, such as a downlink channel emulator, detailed logging and friendly control.

The battery pins of a UE are connected to the DC power analyzer N6705B. This allows both the control of the input voltage into the phone and to measure the instantaneous current drawn by the device. The N6705B power analyzer supports up to four devices connected at the same time.

The UE is connected via USB cable to the server running the testbed automation software. More specifically, the UE USB data cables are connected to a DUT HUB, which in turn is connected to the server by a single USB cable.

Finally, the following elements are connected via Ethernet in a local network: UXM, N6705B, switch driver, orchestration server, EPC, transport, and additional services (see Figure 4 for more details).

2.2 Software Architecture

Figure 5 shows the software interfaces between the testbed management layer, the infrastructure components, and with the testbed Portal.

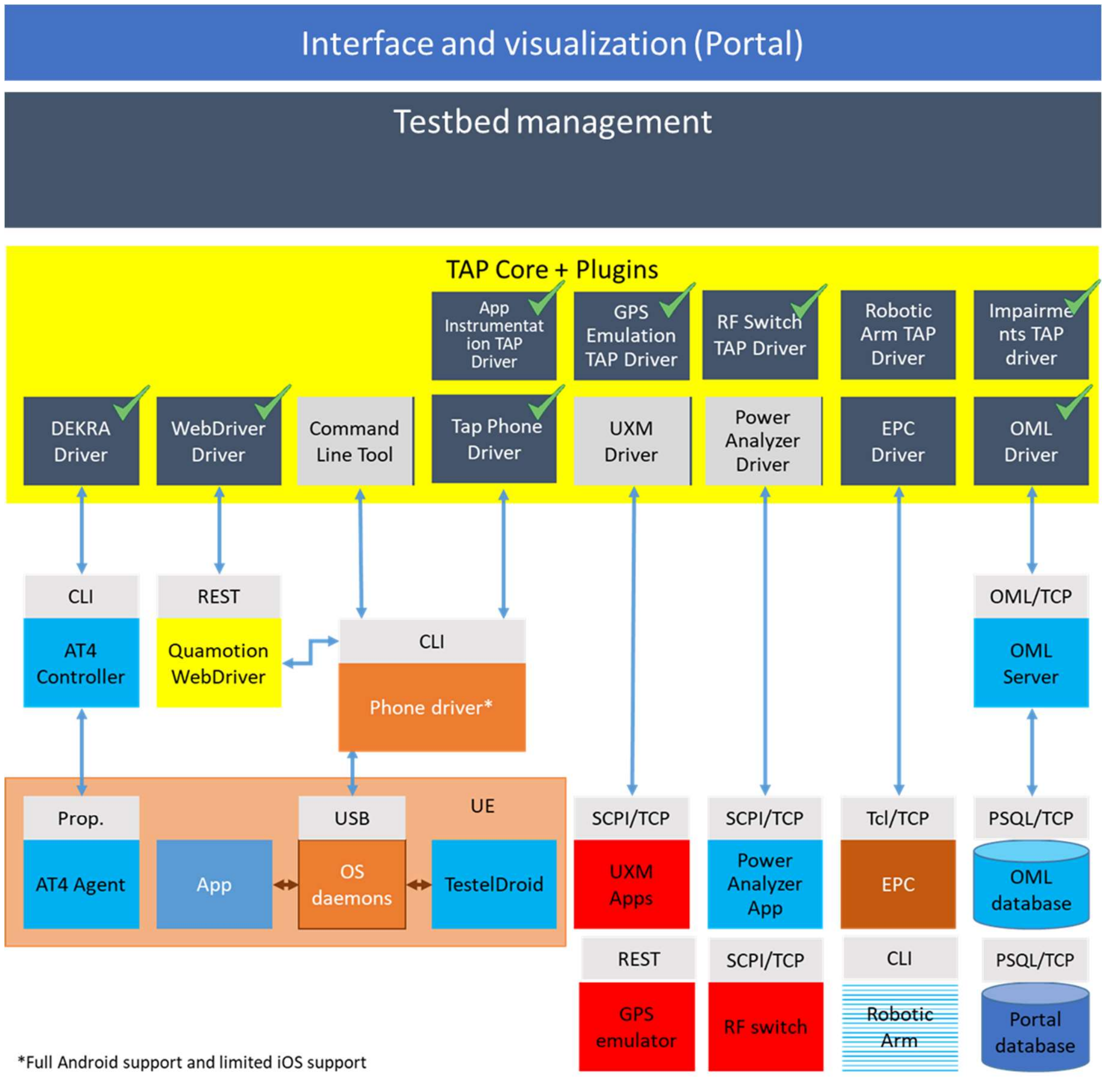


Figure 5 Testbed software architecture

In the testbed management layer lies the Test Automation Platform (TAP), the controller of the testbed components. TAP controls all the components of the testbed through appropriate TAP drivers provided by TAP plugins. The TAP core also comes with a set of standard plugins (see Section 4.1).

Many components offer a SCPI interface to receive commands. TAP provides support for writing drivers for SCPI-based components, which facilitates the work of adding more components. This is the case of the drivers for the UXM and Power Analyzer apps running on their corresponding hardware units. In both cases, the SCPI commands are delivered through a TCP connection.

For Android apps, the ADB command-line tool is a fundamental component. ADB can be used to send commands to apps running on the UEs, or automate certain actions such as switching airplane mode ON and OFF to force the attachment of the UE to the base station. Some of the UE automation tools that are part of the testbed, such as Quamotion WebDriver, use ADB to perform their function.

The Quamotion WebDriver provides a REST API to manage the apps on the Android device, and perform user actions, such as tapping or swiping. These commands are delivered to the Android UE using ADB. TestelDroid is also managed through ADB.

The EPC can be controlled through a fixed set of TCL scripts that use a TCL scripting API provided by the EPC components emulators. The TAP driver will execute these scripts, which then will send the appropriate commands to the ECP through the TCL API.

The measurements from all the tools are collected and sent to a central OML server, which uses a custom OML protocol. While some tools may send measurements directly to the OML server, the TAP orchestrator will use a driver that will allow sending measurements to the OML server from TAP, in two ways. First, for tools that generate results in CSV files, the driver will collect these files and send them as measurements to the OML server. Second, the driver will implement the standard TAP mechanism for handling results from drivers, so that drivers which are already well integrated with TAP can publish them to the OML server without additional work.

2.3 Software Deployment

Regarding the deployment, software is split between Windows and Linux machines (as required by each tool), and virtualized when possible. The current deployment uses a single Windows machine and several Linux virtual machines (VMs). Table 6 provides an overview of the installed software components in the testbed.

The Windows computer runs the TAP software, the ADB server that can talk directly to the (Android OS) UE and the DEKRA controller which manages the DEKRA agents installed in the UE. Other tools that use ADB to relay commands to the UE, such as the Quamotion WebDriver, need to be installed in the same computer, or be configured to use ADB remotely.

Table 6 Software deployment in the testbed

Software component	**OS**	**Notes**
Test Automation Platform	Windows	Main computer
DEKRA performance tool controller	Windows	Main computer
DEKRA agent	Android / iOS	UE
TestelDroid	Android	UE
Quamotion WebDriver	Windows/Linux	Helper instance in Linux as part of Portal
Testbed Portal	Linux	VM, automated recreation
OML / PostgreSQL	Linux	VM, automated recreation
Local application servers	Linux	VM, automated recreation
ADB server	Windows	Main computer

EPC	Linux	Multiple VMs

Linux VMs have been defined using Vagrant [2], which allows recreating them from scratch and updating their contents easily. Vagrant uses a mixed declarative and imperative approach to describe the contents of a VM, including the actions that must be performed to set up any required software that will run on the VM. The OML and the local servers are installed in these VMs. The EPC is distributed across a set of Linux VMs.

The Portal is deployed in a separate Linux VM. This machine hosts the web server that runs the Portal, as well as the database that is used as backend.

2.4 Testbed Management Workflow

End users, of the type Application developers, use the TRIANGLE portal to upload their apps and then define the test campaigns. The Portal provides a set of forms or templates where the users declare the features of their Apps (implementation statements), and any required additional information (such as the app user flow which contains a sequence of actions to stimulate the app during the test) to carry out the tests. The Portal is described in more detail in Section 3. The overall workflow is illustrated in Figure 6 and is described below.

First, to configure and execute a test, the orchestrator (implemented in Python as shown in Figure 7) uses the information provided by the end user in the Portal. The information is retrieved by the Orchestrator using the REST API provided by the Portal as described in Annex 2.

With this information the Orchestrator instantiates the Compositor to compose a TAP test plan that includes the network scenario setup, the app installer, the app user flow and the measurements required to compute a given set of KPIs. The KPIs to compute depend on the App features declared by applicant or testbed user. The TAP test plan is built from existing TAP templates and TAP scenarios as shown in Figure 7.

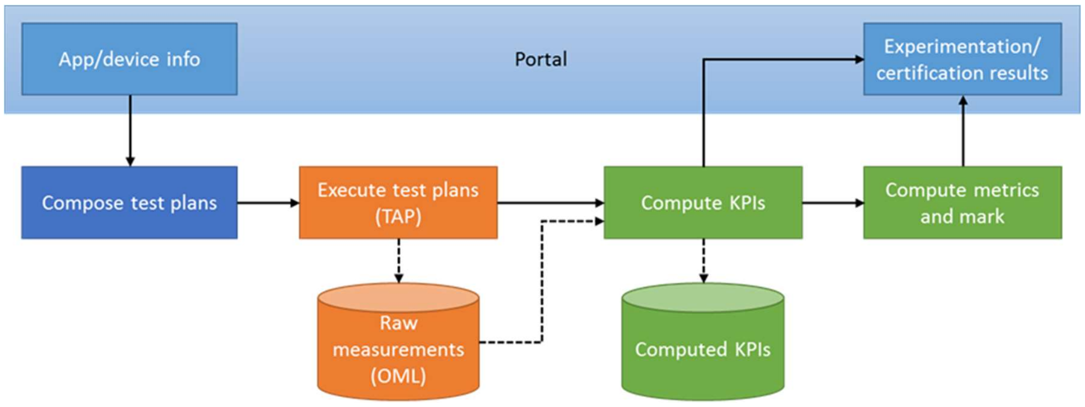


Figure 6 Testbed managment workflow

In the next step of the testbed workflow, the Executor is instantiated to execute the TAP test plan. The TAP test plan configures all equipment needed to run the tests (such as the UXM or the N6705B), executes the body of the test, and gathers results produced by any of the measurement probes and tools. The results are stored in a database.

Then, the Orchestrator instantiates the ETL module (Extract, Transform and Load) to compute the KPIs from these measurements. The computed results are stored in a separate database. For certification, a set of metrics are derived out of the computed KPIs. To rate the product using the TRIANGLE mark, the metrics are compared against a set of reference values. These metrics are stored in a separate database.

The final step is to present the results to the end user in the Portal.

The workflow described above requires the components show in Figure 7. For the sake of clarity,Figure 7 only shows control and management entities. Out of all these components, the key one is the so-called Orcomposutor, which is the combination of the Orchestrator, the Compositor and the Executor. The Orcomposutor also provides a REST API which is used by the Portal to order the execution of a test campaign defined by the user in the Portal. The Orcomposutor is described in more detail in Section 4.

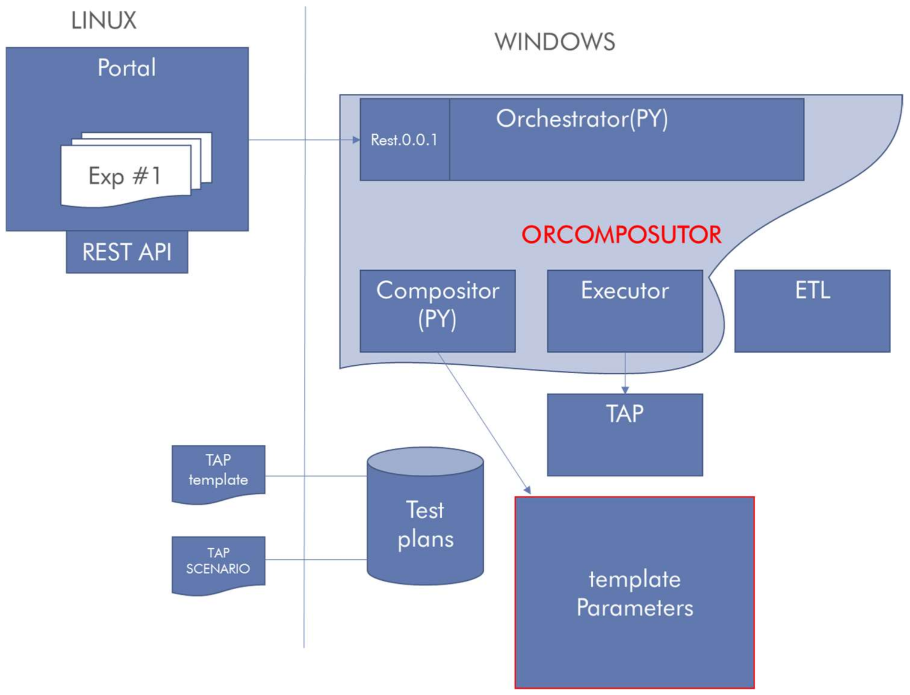


Figure 7 Control and management entities of the testbed

3 Interface and Visualization (Portal)

The TRIANGLE Portal is the main entry point to the TRIANGLE testbed for app developers. In this Portal, among other things, end users can upload new apps. In addition, end users will have to declare the uses cases and their apps (see Figure 8). Each use case has a set of features associated, as shown in Figure 11. These features will define what can be tested through experiments, or which tests specifications will be applicable when they opt for.

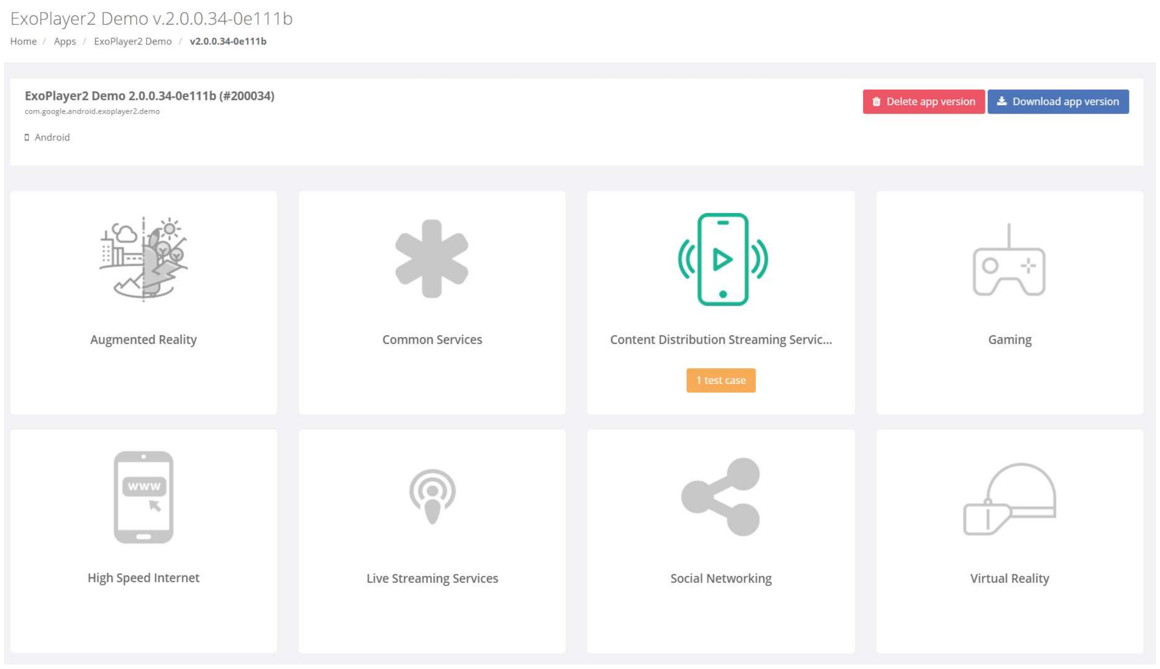

ExoPlayer2 Demo v.2.0.0.34-0e111b
Home / Apps / ExoPlayer2 Demo / v2.0.0.34-0e111b
ExoPlayer2 Demo 2.0.0.34-0e111b (#200034)
com.google.android.exoplayer2.demo
Android
Delete app version
Download app version
Augmented Reality
Common Services
Content Distribution Streaming Servic...
1 test case
Gaming
High Speed Internet
Live Streaming Services
Social Networking
Virtual Reality

Figure 8 Uses cases

Users can also define their own experimentation campaigns to test certain features of their app. For these campaigns, users have some high-level options they can configure: the scenario of the test, the device on which the test will be carried out, and a subset of the applicable KPIs. The Portal also provides a code snippet that should be used by the developer inside its app to measure the KPIs associated to the features declared.

3.1 Organization

The main sections of the Portal (for an app developer user) are:

- Login: a standard login page, integrated with LDAP.
- Dashboard: the main page that users see when they log in into the Portal. It contains a summary of the activity of the user, e.g. apps uploaded, or tests performed.
- Apps: app developers can manage their apps in this section. They can upload new app files, which will be recognized and categorized automatically. For each app, they can see the test campaigns (i.e., the group of test cases selected by the user) in which it has been used.

- Booking: Testbed booking is managed by the users directly.
- Test Campaigns: Users create test campaigns to test their apps in a particular device and scenario. A test campaign is a set of test cases selected by the user to be executed from one or several test specifications.
- One specific test campaign: It is the certification test campaign. The certification test campaign is the set of test cases required (i.e., applicable based on the features supported) to obtain the TRIANGLE mark. The set of test cases in the certification test campaign is automatically generated based on the app characteristics/features.
- Help: information about the usage of the Portal.

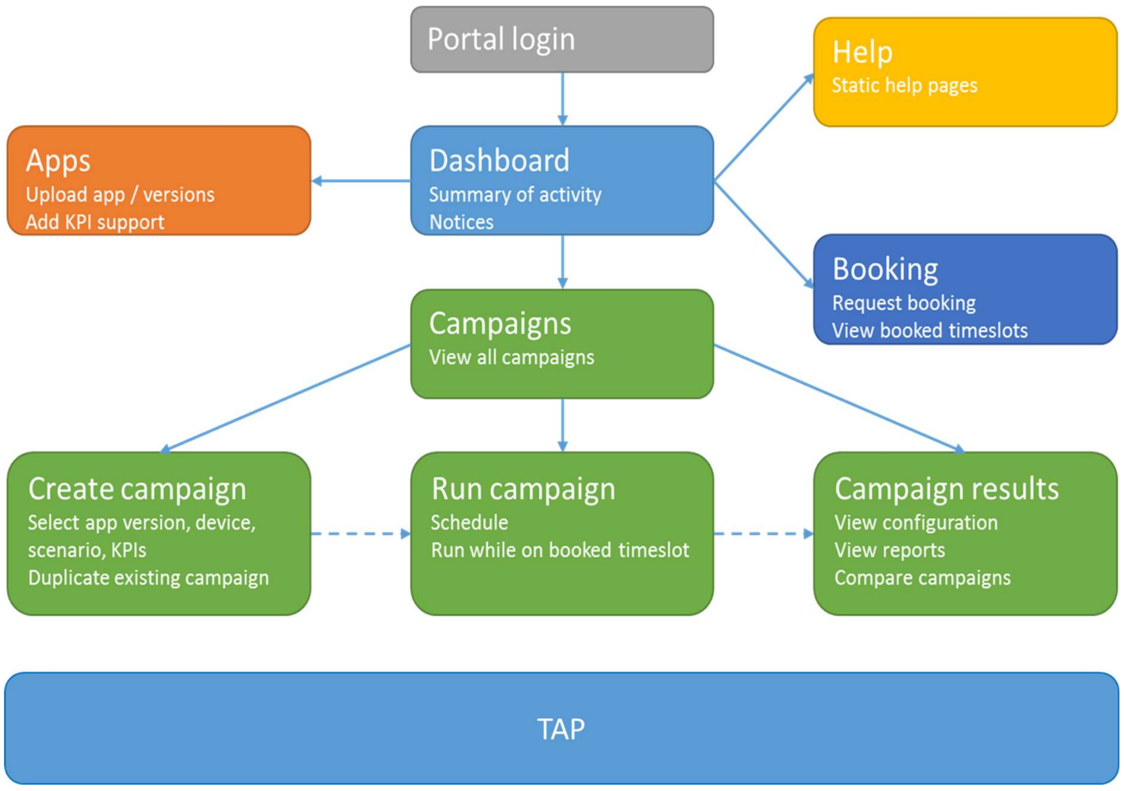


Figure 9 Overview of Portal organization for app developers

3.2 Data

The Portal must store data to support the features outlined above. The main purpose of the Portal is to prepare and run tests. The main entities managed by the Portal and their relationship are shown in Figure 10. They are described in the remainder of this section. The names of the entities managed by the portal are written in uppercase, to highlight them.

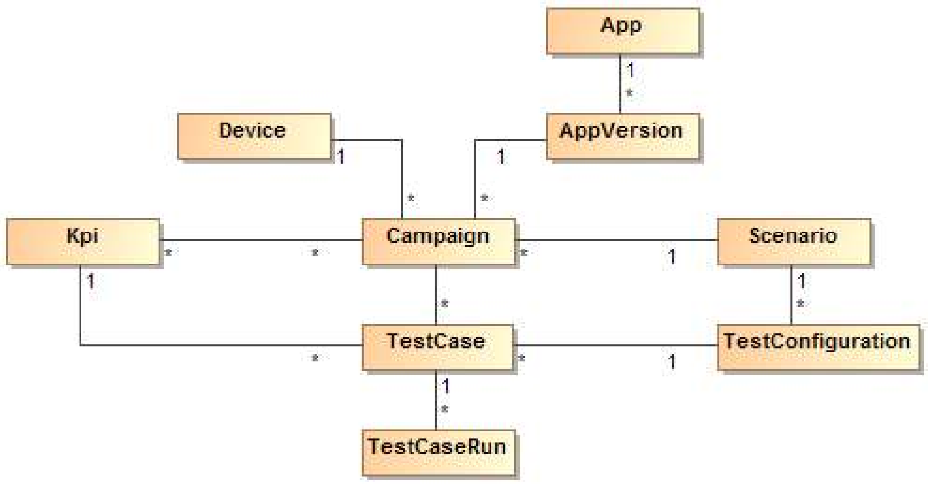


Figure 10 Main entities for app developers in the TRIANGLE Portal

App developers upload apps to the Portal, to run tests on the reference devices provided by the testbed. The app developer must select the features implemented by the app, from a predefined list of features (implementation statement). Once the app has been uploaded, the developer can run test cases. The developer can later upload several app versions (e.g. APK files for Android) of the same app, and compare the results obtained in different test cases. Different apps from the same app developer will be isolated, i.e., the user will not be able to compare results from test across different Apps.

Apps will be tested according to the test cases selected in the test campaign. The test cases that build the test campaign are automatically selected based on the features of the app, previously selected by the app developer. The developer can modify the list of test cases in the test campaign or can change the test cases which are provided in the TRIANGLE testbed to run its own test cases. The test case automation controls instruments, devices, etc., captures measurements and compute KPIs. The measurements and the KPIs are made available to the users of the TRIANGLE testbed. Furthermore, measurements and KPIs are used also to compute certain metrics. A metric is a categorization of the KPIs from a user satisfaction perspective. The metrics are used to provide the assessment before the TRIANGLE mark can be granted.

To run test cases with different app versions, app developers will create different test campaigns. For each test campaign, the app developer can select the device on which to run the test cases, the high-level scenario from a fixed set and, optionally, a subset of the applicable KPIs. A single high-level scenario abstracts several concrete test configurations (e.g., “Urban” includes “Pedestrian” and “Driving”), with the actual network configuration that will be applied.

Each test campaign includes a list of test cases. A test case is a sequence of actions required to achieve a specific test purpose. Each time the same test campaign is executed, all its test cases are executed. Each one will produce a new set of results (test case run, i.e. the results and metadata of the execution).

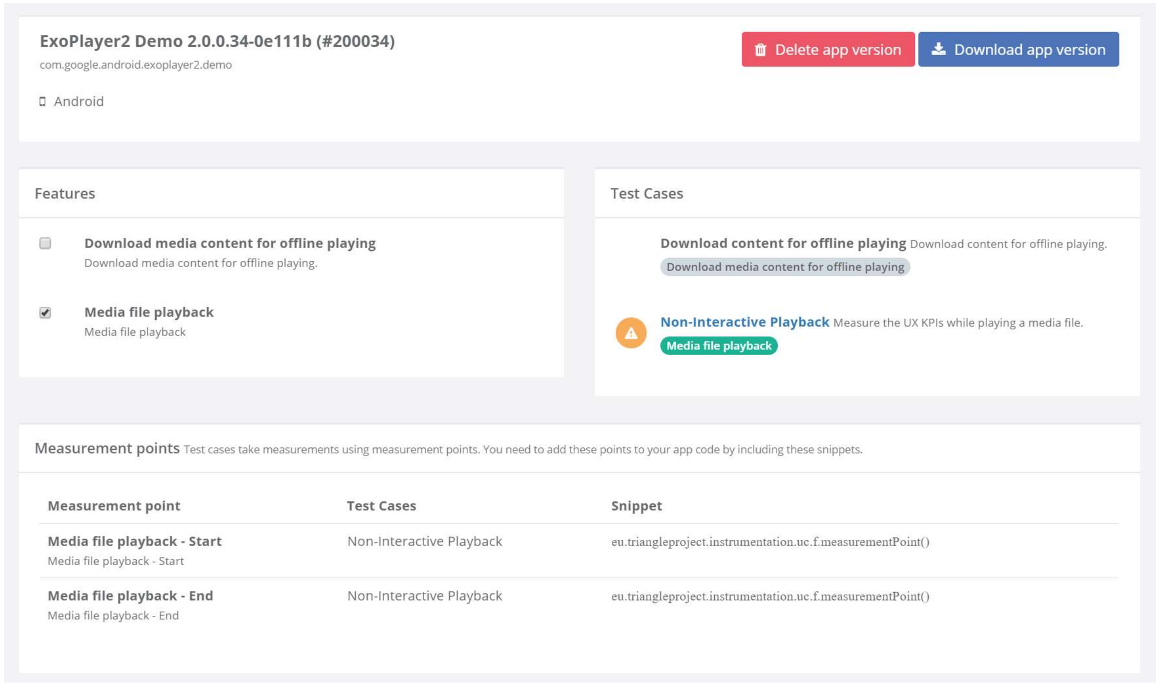
3.3 Implementation

The Portal has been implemented using the Ruby on Rails framework c. For the backend, the Portal uses a PostgreSQL database (different from the ones used by the OML server or TAP). For the frontend, the Portal uses the Bootstrap framework [3]. The Portal follows the organization outlined above. The data stored in the backend is structured as described in the previous subsection. The Portal follows the organization outlined above. The data stored in the backend is structured as described in the previous subsection. The database tables are listed in Annex 1.

3.4 Information Provided by Users

App developers provide information about their apps through the Portal. In addition, when creating an experimentation campaign, or going for certification, they may be asked for additional information.

Since the information provided for apps is of considerable size, users will be able to enter it at their own pace, using the forms available in the Portal.


ExoPlayer2 Demo 2.0.0.34-0e111b (#200034)
com.google.android.exoplayer2.demo
Android

Features

- [ ] **Download media content for offline playing** Download media content for offline playing.
- [✓] **Media file playback** Media file playback

Test Cases

Download content for offline playing Download content for offline playing.
Download media content for offline playing

Non-Interactive Playback Measure the UX KPIs while playing a media file.
Media file playback

Measurement points Test cases take measurements using measurement points. You need to add these points to your app code by including these snippets.

Measurement point	Test Cases	Snippet
Media file playback - Start Media file playback - Start	Non-Interactive Playback	eu.triangleproject.instrumentation.uc.f.measurementPoint()
Media file playback - End Media file playback - End	Non-Interactive Playback	eu.triangleproject.instrumentation.uc.f.measurementPoint()

Figure 11 Features selection and measurements points

3.4.1 App info

The following shall be provided for each app:

- **App file**, e.g. APK file for Android apps. Some metadata can be extracted from this file, such as the **app name**, **version**, and **codename**.
- **Features applicable to the app**. The app developer will select from a list of uses cases and its associated features, which ones apply to his or her app. The list of features were extracted from D2.2, so that a clear mapping between them and the test specifications/ICS table can be done.

- **How to measure**. Each feature will be tested according to a set of KPIs. In order to test many of the features, and to get the appropriate measurements to compute the KPIs, app developers will have to provide additional information.
    - **App user flows**. A sequence of user actions that can be executed automatically to test that feature. When possible, the app user flows will be asked once per feature, so that users do not have to enter separately for each KPI, or in groups. For instance, if there is a "post photo" feature on which several KPIs can be measured, they will only be asked for a single app user flow.
    - **Measurement points**. In order to compute some KPIs, the app developer must define how some of the required measurements can be obtained in his or her app. Depending on the KPI, the user will be able to provide these measurement points in several ways. For instance, as a particular user action within the app user flow, or a measurement point set using an instrumentation library.

Since apps may evolve over time, and their features, or how to measure some of the KPIs, can change over time, the developer will be able to customize this information for each app version.

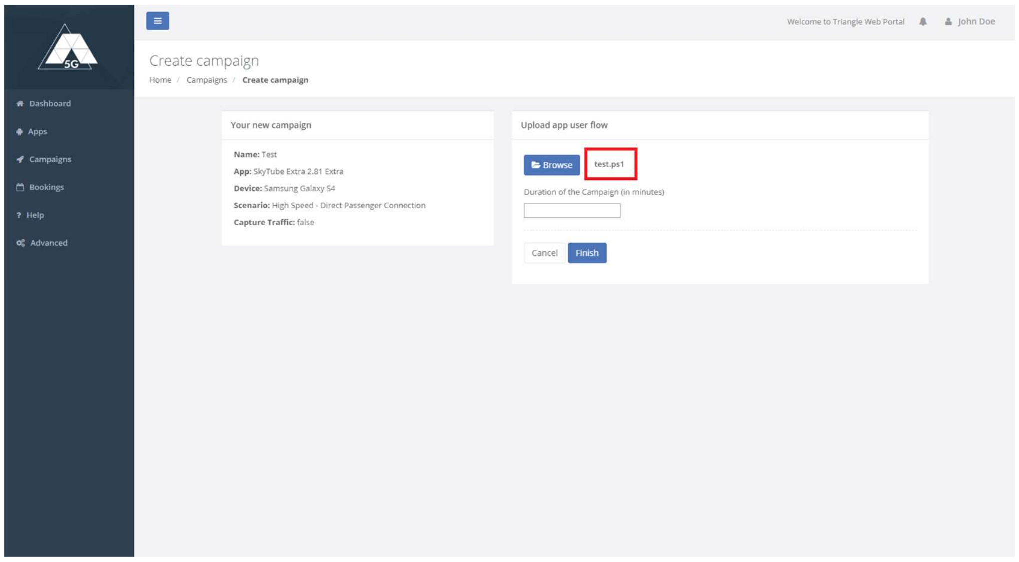
3.4.2 App experimentation

When the user wants to carry out an experimentation campaign, he or she must select the following information:

- **App and version**. The user may have more than one app, and more than one version of that app. Therefore, he or she must select the one that will be tested.
- **Scenario**. One of the high-level scenarios defined in the TRIANGLE project, which hides the complexity of the parameters that must be configured in the Testbed equipment.
- **Device**. One of the devices available in the Testbed, on which the app will be tested.
- **Features/KPIs** (optional). A subset of all the features/KPIs applicable to the app, if the developer is interested only in testing part of the app.

3.4.3 Powershell Scripts as App User Flow

This feature allows PowerShell files (.ps1) recorded with Quamotion tool to be uploaded as App User Flows.



Create campaign

Home / Campaigns / Create campaign

Your new campaign

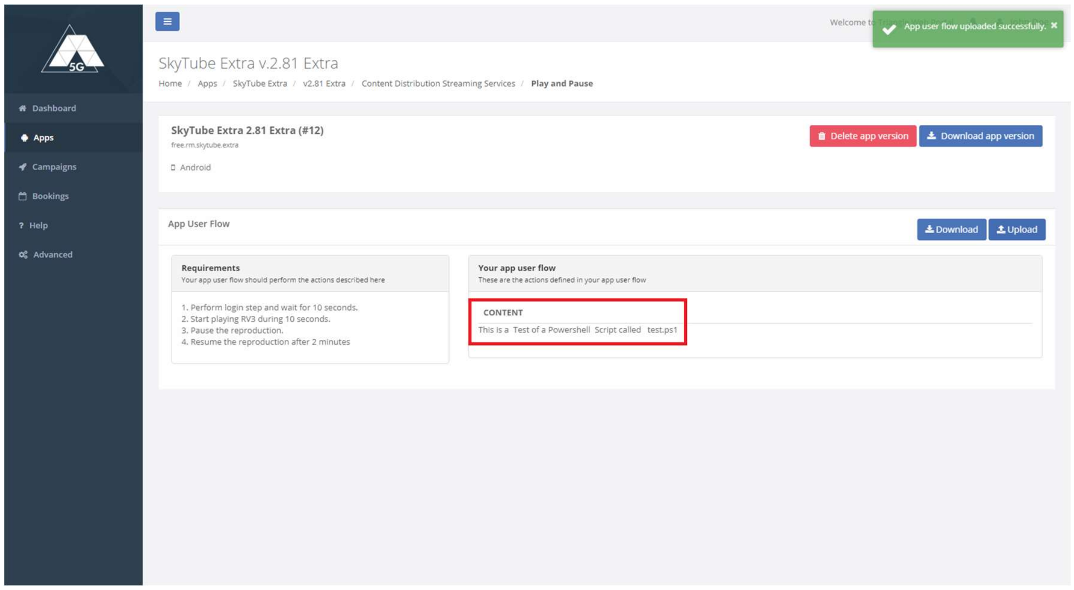
Name: Test
App: SkyTube Extra 2.81 Extra
Device: Samsung Galaxy S4
Scenario: High Speed - Direct Passenger Connection
Capture Traffic: false

Upload app user flow

test.ps1

Duration of the Campaign (in minutes)

Figure 12 Allow uploading PowerShell files (.ps1)



App user flow uploaded successfully.

SkyTube Extra v.2.81 Extra

Home / Apps / SkyTube Extra / v2.81 Extra / Content Distribution Streaming Services / Play and Pause

SkyTube Extra 2.81 Extra (#12)
free.rm.skytube.extra
Android

App User Flow

Requirements
Your app user flow should perform the actions described here

1. Perform login step and wait for 10 seconds.
2. Start playing RV3 during 10 seconds.
3. Pause the reproduction.
4. Resume the reproduction after 2 minutes

Your app user flow
These are the actions defined in your app user flow

CONTENT
This is a Test of a Powershell Script called test.ps1

Figure 13 Display powershell script content

Powershell scripts are used for specifying the actions in the App Userflows. Quamotion's Powershell library provides commands for executing the actions on the app under test (such as Click-Element or Enter-Text) while Powershell provides statements for controlling the execution flow. It is also possible to include standard cmdlets, for example, Write-Output to save information on a log file or Start-Sleep to introduce delays on the execution.

The following is an example of an userflow that checks if a custom keyboard is available and, if so, enters 'Hello' 10 times using a combination of standard Powershell and Quamotion commands:

```
$messageBox   =   Find-Element   -xpath   "//android.widget.EditText[@resource-id='com.test.app:id/send_text']"


if ([string]::IsNullOrEmpty($messageBox)) {
            throw "Could not find keyboard."
} else {7
            for ($i=1; $i -le 10; $i++) {
                        Click-Element -elementId $messageBox
Set-Value -elementId $messageBox -value "Hello"
       Click-Element -xpath "//*[@resource-id='com.test.app:id/btn_send']"
                        Start-Sleep -Seconds 1
            }
}
```

3.5 Backend REST API

The Portal includes a REST API that provides access to its backend. This API allows external services to request information stored in the backend database or update it. Outside of the Portal, the primary user of this REST API is Orcomposutor (described in Section 4.5), which uses the API to fetch the details of campaigns to be executed, and to upload the results once they are finished.

All API calls return a JSON object with data about the requested resource. The REST API largely adheres to the HATEOAS (Hypermedia as the Engine of Application State) principle. In practice, this means that JSON responses include URLs that point to other related resources. For instance, when querying a single application, the JSON response includes a list of its versions with some information such as their Id and version code, as well as a URL to the resource of that application version, in order to request more details.

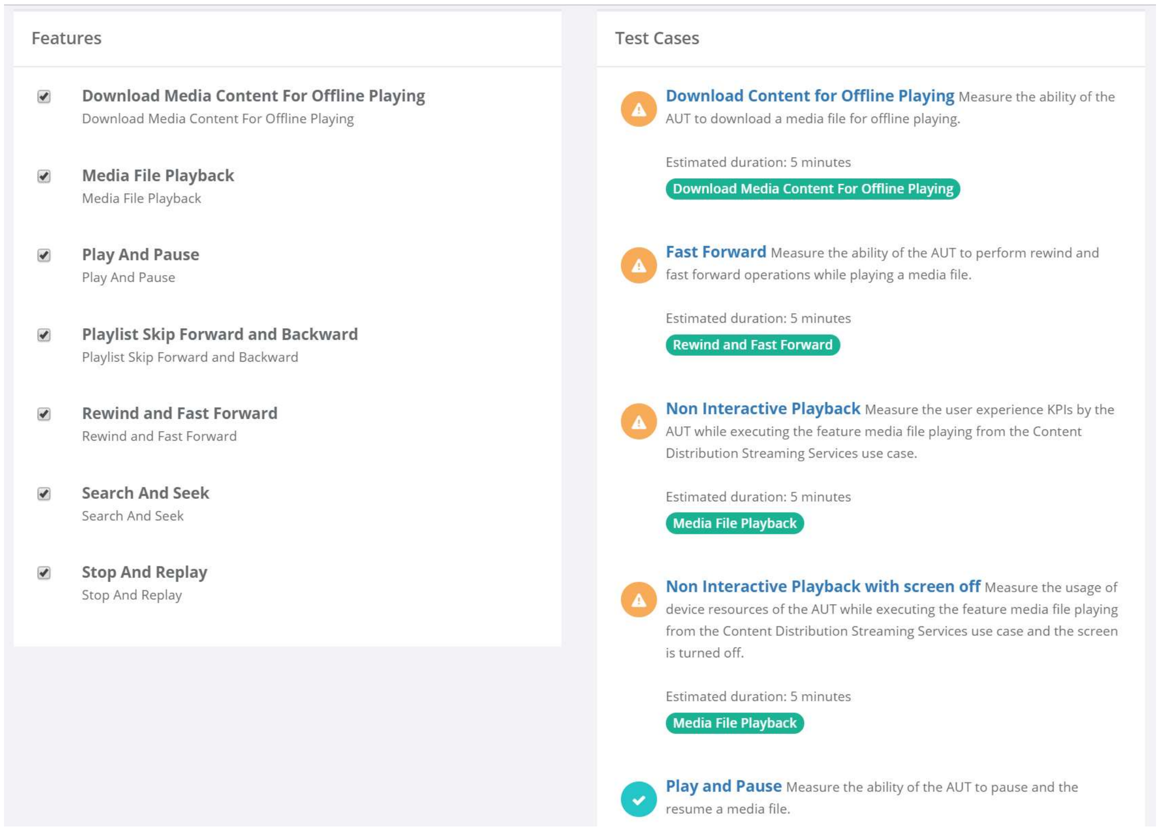
The REST API provides access to the following resources:

- Devices
- Users
- Apps and their versions
- Features
- Test cases
- Scenarios
- Campaigns

Methods available to access to these resources are detailed in Annex 2.

3.6 Test Cases Supported

The Portal offers full support for the test cases defined in D2.2 for the App User Experience (AUE) domain, the Device Resources Usage (RES) domain, the App Energy Consumption (AEC) domain, the Network Resources Usage (NWR) and the App Reliability (REL).



Features

☑ Download Media Content For Offline Playing
Download Media Content For Offline Playing

☑ Media File Playback
Media File Playback

☑ Play And Pause
Play And Pause

☑ Playlist Skip Forward and Backward
Playlist Skip Forward and Backward

☑ Rewind and Fast Forward
Rewind and Fast Forward

☑ Search And Seek
Search And Seek

☑ Stop And Replay
Stop And Replay

Test Cases

Download Content for Offline Playing Measure the ability of the AUT to download a media file for offline playing.
Estimated duration: 5 minutes
Download Media Content For Offline Playing

Fast Forward Measure the ability of the AUT to perform rewind and fast forward operations while playing a media file.
Estimated duration: 5 minutes
Rewind and Fast Forward

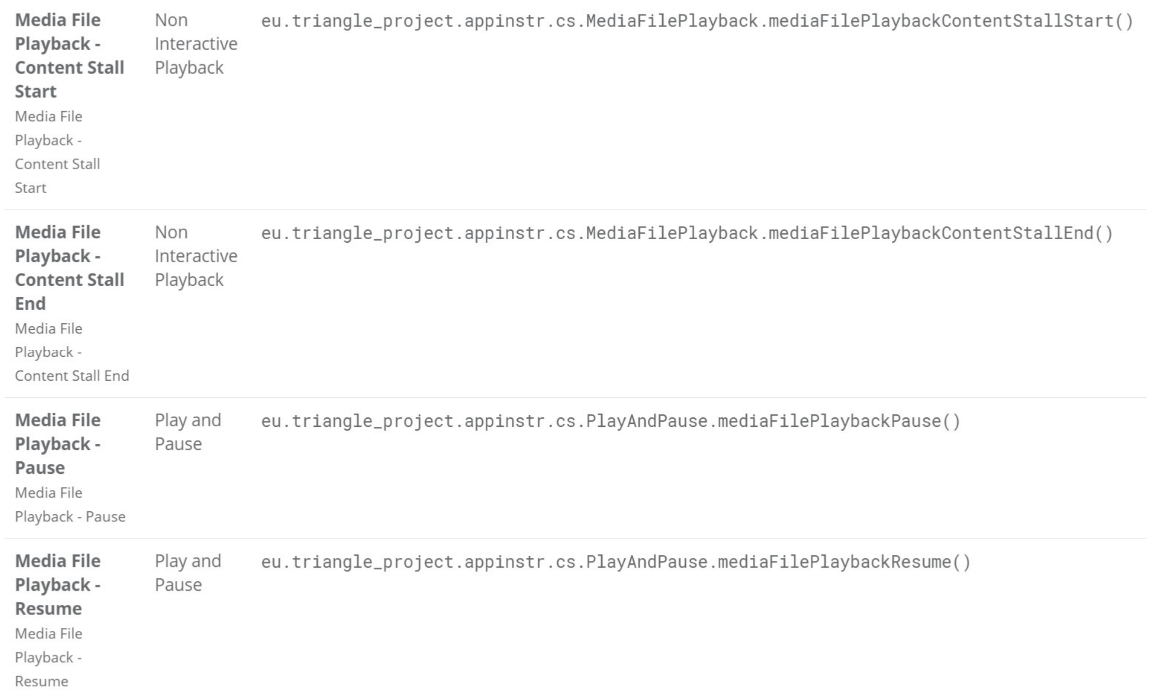
Non Interactive Playback Measure the user experience KPIs by the AUT while executing the feature media file playing from the Content Distribution Streaming Services use case.
Estimated duration: 5 minutes
Media File Playback

Non Interactive Playback with screen off Measure the usage of device resources of the AUT while executing the feature media file playing from the Content Distribution Streaming Services use case and the screen is turned off.
Estimated duration: 5 minutes
Media File Playback

Play and Pause Measure the ability of the AUT to pause and the resume a media file.

Figure 14 The Portal identifies the available test cases and presents a description for each one

As shown in Figure 15, the Portal also provides a list of the measurements points that must be included in the source code of the application under test to obtain the measurements specified in the test cases.



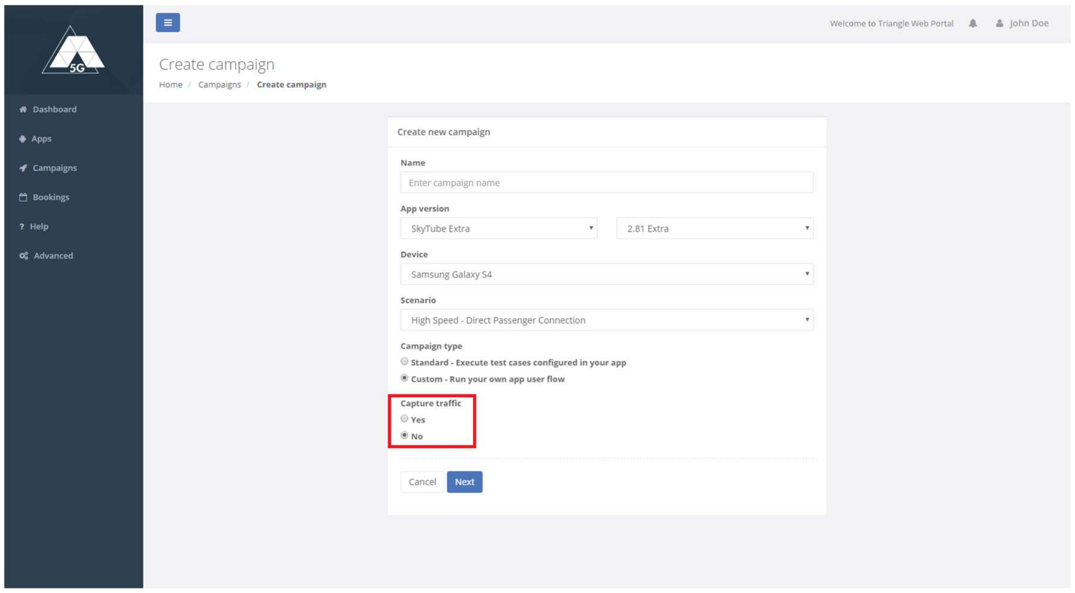
Media File Playback - Content Stall Start Media File Playback - Content Stall Start	Non Interactive Playback	`eu.triangle_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackContentStallStart()`
Media File Playback - Content Stall End Media File Playback - Content Stall End	Non Interactive Playback	`eu.triangle_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackContentStallEnd()`
Media File Playback - Pause Media File Playback - Pause	Play and Pause	`eu.triangle_project.appinstr.cs.PlayAndPause.mediaFilePlaybackPause()`
Media File Playback - Resume Media File Playback - Resume	Play and Pause	`eu.triangle_project.appinstr.cs.PlayAndPause.mediaFilePlaybackResume()`

Figure 15 Measurement points required to compute the measurements specified in the test cases

3.7 Traffic capture

Traffic capture can be computational intensive and can interfere with the measurements. This feature is configurable through the added field ‘Capture Traffic’ in the “Create Campaign” form. This is shown in

When the user creates a custom campaign or standard campaign, the user can enable or disable the traffic capture. For the certification campaigns this option is not available.


Welcome to Triangle Web Portal
John Doe
Dashboard
Apps
Campaigns
Bookings
Help
Advanced
Home / Campaigns / Create campaign
Cancel
Next

Create campaign

Create new campaign

Name: Enter campaign name

App version: SkyTube Extra | 2.81 Extra

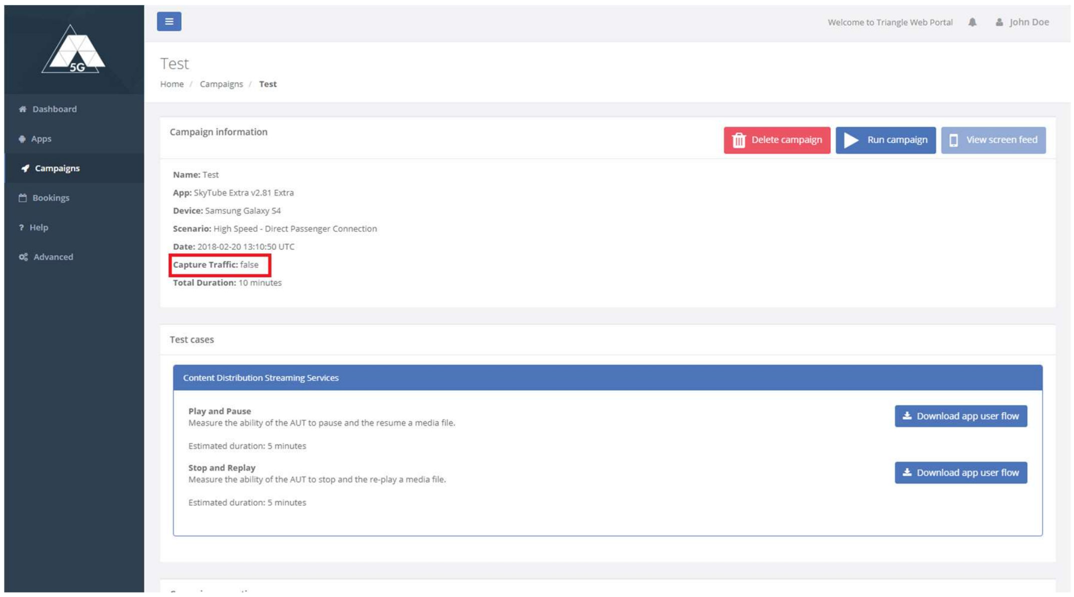
Device: Samsung Galaxy S4

Scenario: High Speed - Direct Passenger Connection

Campaign type: Standard - Execute test cases configured in your app [✓ Custom - Run your own app user flow]

Capture traffic: Yes [✓ No]

Figure 16 ‘Capture Traffic’ selector



Test

Campaign information

Name: Test
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Scenario: High Speed - Direct Passenger Connection
Date: 2018-02-20 13:10:50 UTC
Capture Traffic: false
Total Duration: 10 minutes

Test cases

Content Distribution Streaming Services

Play and Pause
Measure the ability of the AUT to pause and the resume a media file.

Estimated duration: 5 minutes

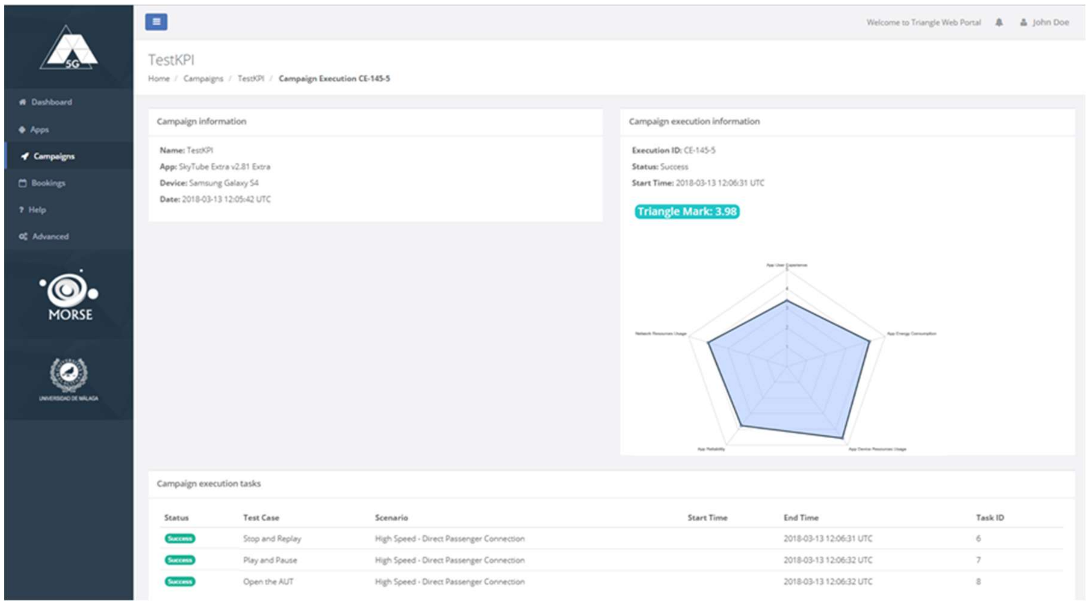
Stop and Replay
Measure the ability of the AUT to stop and the re-play a media file.

Estimated duration: 5 minutes

Figure 17 Display 'Capture Traffic' selection

3.8 TRIANGLE Mark and Spider Diagram

The Portal displays the TRIANGLE Mark and a spider diagram with the score obtained in each one of the domains.



TestKPI

Home / Campaigns / TestKPI / Campaign Execution CE-145-5

Campaign information

Name: TestKPI
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Date: 2018-03-13 12:05:42 UTC

Campaign execution information

Execution ID: CE-145-5
Status: Success
Start Time: 2018-03-13 12:06:31 UTC

Triangle Mark: 3.98

Campaign execution tasks

Status	Test Case	Scenario	Start Time	End Time	Task ID
Success	Stop and Replay	High Speed - Direct Passenger Connection		2018-03-13 12:06:31 UTC	6
Success	Play and Pause	High Speed - Direct Passenger Connection		2018-03-13 12:06:32 UTC	7
Success	Open the AUT	High Speed - Direct Passenger Connection		2018-03-13 12:06:32 UTC	8

Figure 18 TRIANGLE mark

3.9 Remote screen

Following a demand from the researchers that have been using the testbed, TRIANGLE now provides a remote screen option for the TRIANGLE Testbed Portal. This remote screen shows a live video of what is happening in the physical screen of the device under test using VNC technology.

To achieve this, we take advantage of the screenshotfeed feature present in the Quamotion Frontend[http://docs.quamotion.mobi/en/latest/frontend/frontend-devices.html#open-the-screenshotfeed-for-a-device], adding a basic authentication system to increase the security of Quamotion solution.

This way, the clients have access to the full Quamotion Frontend when they reserve the TRIANGLE Portal Testbed. Only the researcher that has made a reservation for a certain time slot will have access to the Frontend during that timeslot. In the Frontend, under the "Devices" tab, they will be able to open a new Remote screen session for the device. Additionally, they will be able to view the status of the sessions started during their experiments (in the "Sessions" tab), which will give them more information about the execution of their experiment, something that has also been demanded.

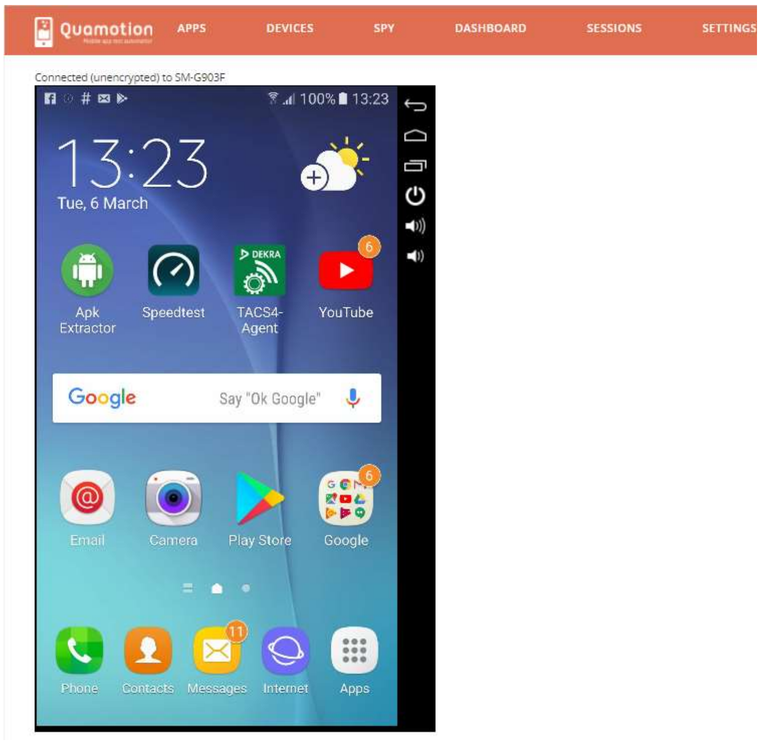


Figure 19 View of the screenshot feed feature provided by Quamotion.

3.9.1 Basic Authentication

As already mentioned, TRIANGLE has extended the Quamotion Frontend with a basic authentication system as a protection mechanism. This way, when a user tries to access the Frontend a prompt will ask for username and password, which has been previously provided by the testbed managers.

To increase the security, first we use an Nginx web server with the reverse proxy feature. This allows us to hide the existence of the Quamotion Frontend server, closing its port to the public, leaving open only the typical port 80. The Quamotion resources are returned to the client as if they originated from the Nginx web server.

Additionally, we use the Nginx Basic Authentication feature, forcing clients to use their email (the same one they use in the Testbed and Booking service) and a password to access the Quamotion Frontend. This authentication system uses a password file that stores a list of usernames and their unencrypted password. We, using user's email as seed, generate the password. This password is sent to their owners and is permanent.

The password file only contains the username and password of the client that has reserved the testbed at the present time. For example, if there are no reservations at a given time, the password file is empty, so nobody can try to access. If there is a reservation at a given time, only the username and password of the client who did the reservation is stored in the password file at that given time, so only him or her can try to access.

The editing of the password file is achieved with a Python script that consumes the Booked API to know if there is a reservation every hour. If there is not, it cleans the password file, deleting any info that there could be inside. If there is a reservation, it gets the email of the person who did it, generates the password, as described before, and updates the content of the password file with this information.

3.10 Booking System

The booking system consists in a web calendar service used by the experimenters to book the TRIANGLE Portal Testbed. The system is completed with a script that queries the current owner of the Testbed at the beginning of every hour and updates the Portal Testbed owner accordingly.

3.10.1 Web Calendar Service

The web calendar service used is Booked Scheduler, an open source booking service based in LAMP that has an API to interact with it, providing the possibility to automate the process.

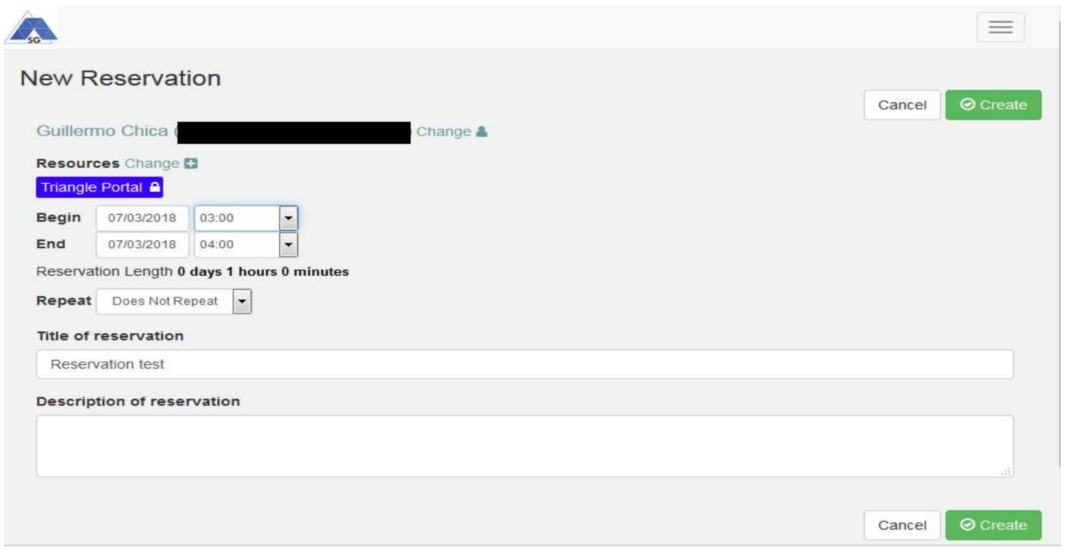
Booked Scheduler allows a user to define a resource element that will be shared by different people. This resource element will be associated with a resource schedule that will contain the resource reservations. In our case, the resource will be the TRIANGLE Portal Testbed. The capacity of this resource is one person, so only one researcher can use the Testbed at a given time. The resource schedule is divided in 1-hour slots.

The same email account used to access the Portal Testbed is used to access the web calendar service to make reservations.

There are several ways to make a reservation. It can be done using any submenu of the “Schedule” tab.

- From Bookings: Clicking on any slot from the timetable to start making a new reservation.
- From My Calendar: Clicking on any day of the calendar to start a new reservation.
- From Resource Calendar: The same as in “My calendar”.
- From Find a Time: Search a slot time and click one of the results.

Any of these methods will redirect to the “New Reservation” page. In this page, one can select the time start and end of the reservation and optionally give it a title and add comments. After clicking “Create”, the system will confirm the reservation if there are no conflicts.


New Reservation

Guillermo Chica ([redacted] Change

Resources Change
Triangle Portal

Begin 07/03/2018 03:00
End 07/03/2018 04:00

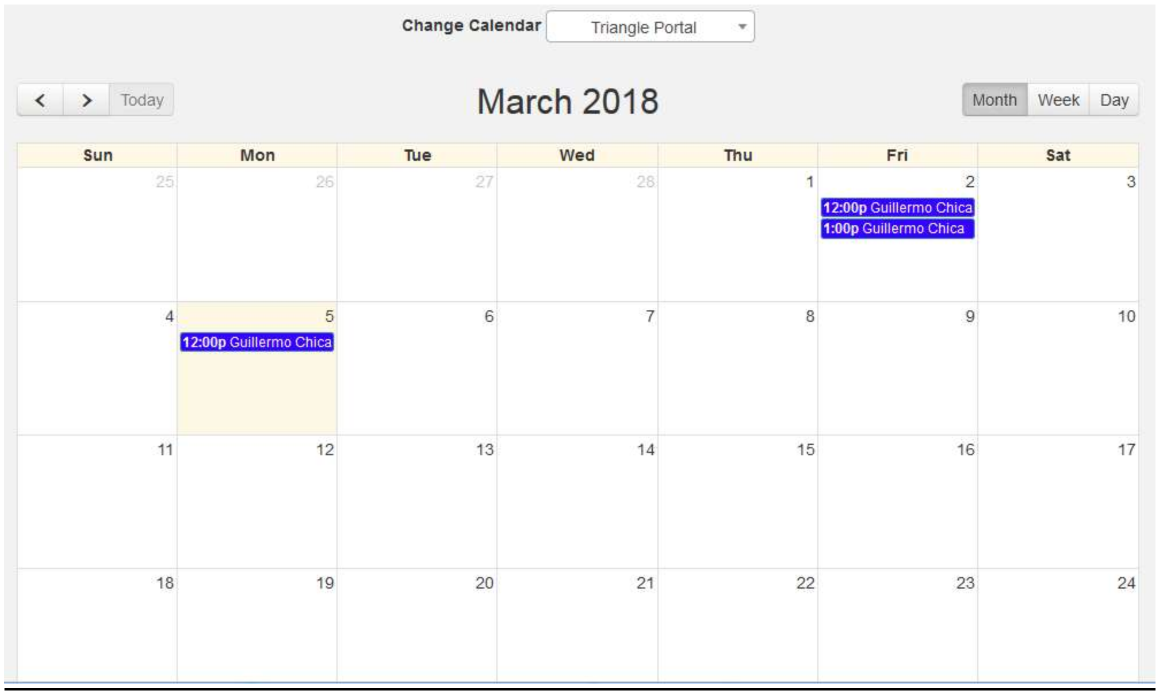
Reservation Length **0** days **1** hours **0** minutes

Repeat Does Not Repeat

Title of reservation
Reservation test

Description of reservation

Figure 20 The New Reservation Page allows a user to select a slot time to use the Portal Testbed



Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	1	2 12:00p Guillermo Chica 1:00p Guillermo Chica	3
4	5 12:00p Guillermo Chica	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24

Figure 21 Calendar interface. It shows the reservations made. Clicking on any day will take you to the New Reservation Page.

3.10.2 Testbed Owner Updating

The web calendar service itself only provides the calendar interface to create reservations, but these reservations alone do not have impact on the TRIANGLE Portal Testbed. In order to automatically update the testbed owner according to the reservations made in the web calendar a couple of scripts have been implemented.

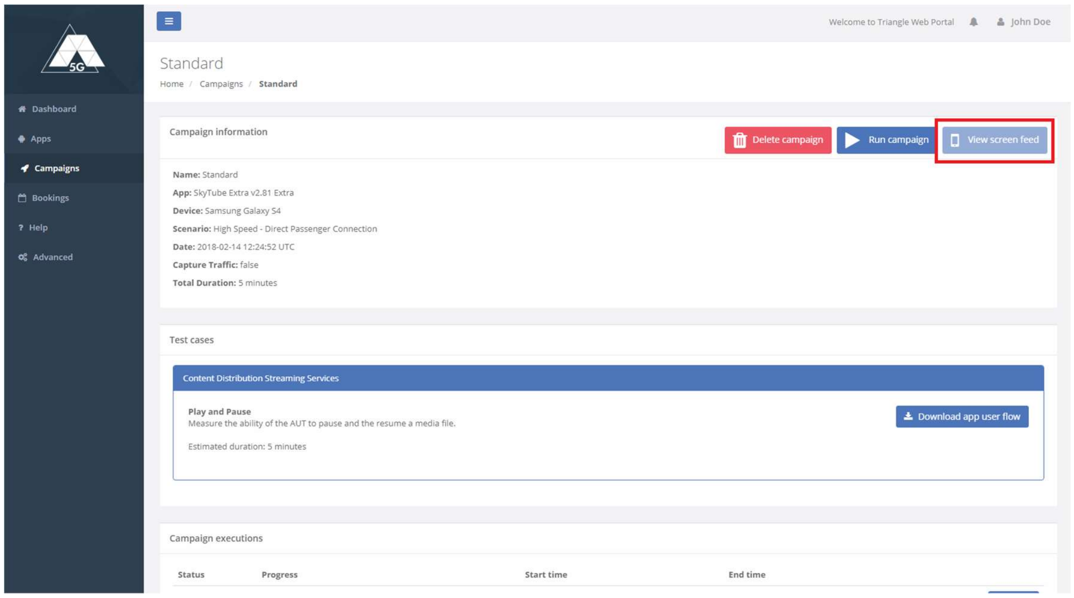
The first script, written in Python, uses the Booked Scheduler API to ask if there is a reservation for the current hour, and if there is one, to get the email of the person who reserved. With this email, a second script, written in Shell, will update the Portal testbed owner using the Portal Ruby on Rails console commands.

These scripts are executed at the beginning of every hour, since the web calendar is structured in 1-hour slots. This way, we provide automatic access to the TRIANGLE Portal Testbed for a researcher when he or she has booked it via the web calendar service.

3.11 Additional Features

3.11.1 Enabling/Disabling Remote Screen

The Portal enables the activation/deactivation of this feature for standard and customs campaigns.



Standard

Home / Campaigns / Standard

Campaign information

Name: Standard
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Scenario: High Speed - Direct Passenger Connection
Date: 2018-02-14 12:24:52 UTC
Capture Traffic: false
Total Duration: 5 minutes

Test cases

Content Distribution Streaming Services

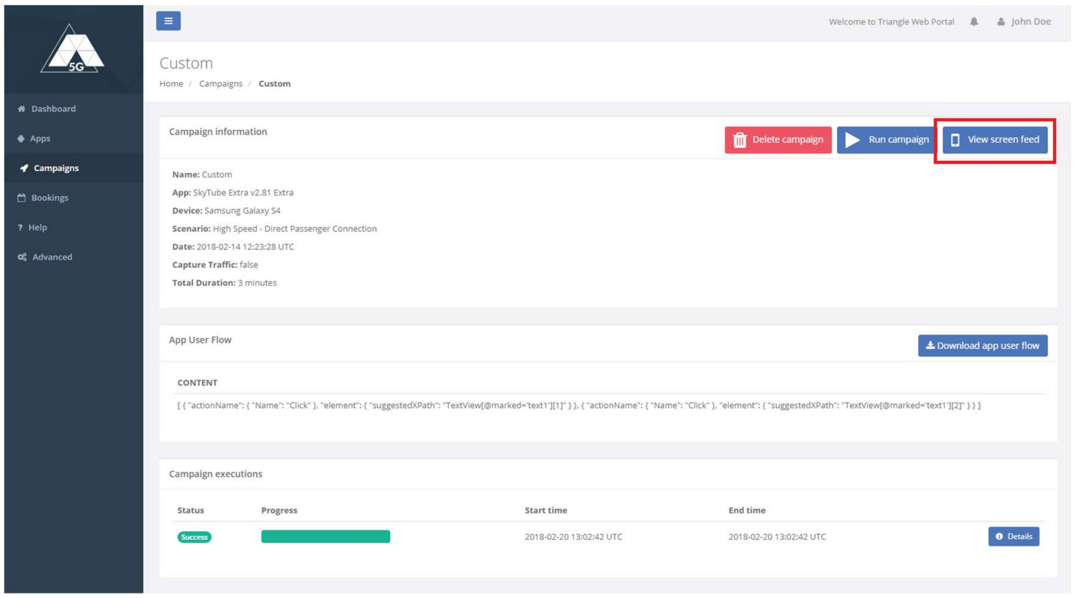
Play and Pause
Measure the ability of the AUT to pause and the resume a media file.

Estimated duration: 5 minutes

Campaign executions

Status	Progress	Start time	End time

Figure 22 Disabled 'View screen feed' in Standard Campaign



Custom

Home / Campaigns / Custom

Campaign information

Name: Custom
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Scenario: High Speed - Direct Passenger Connection
Date: 2018-02-14 12:23:28 UTC
Capture Traffic: false
Total Duration: 3 minutes

App User Flow

CONTENT

```
[ { "actionName": { "Name": "Click" }, "element": { "suggestedXPath": "TextView[@marked='text1'][1]" } }, { "actionName": { "Name": "Click" }, "element": { "suggestedXPath": "TextView[@marked='text1'][2]" } } ]
```

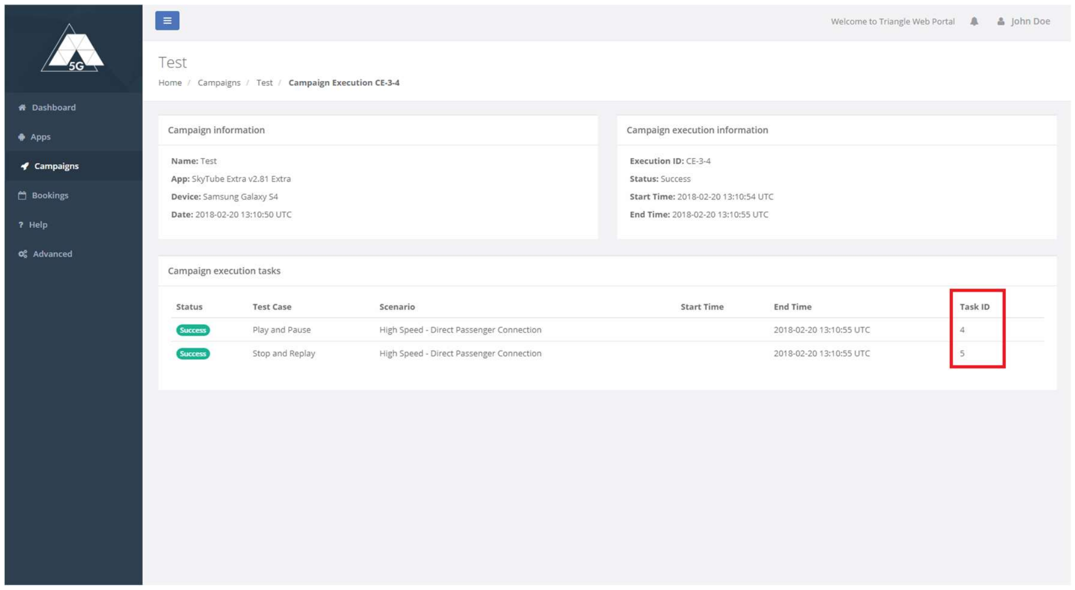
Campaign executions

Status	Progress	Start time	End time	
Success	🟩	2018-02-20 13:02:42 UTC	2018-02-20 13:02:42 UTC	Details

Figure 23 Enabled 'View screen feed' in Custom Campaign

3.11.2 Task ID

This feature shows *'Execution Task ID'* when displaying the Campaign Execution. This is the same ID used in the OML visualizer to identify the graphs corresponding to the execution.



Test

Home / Campaigns / Test / Campaign Execution CE-3-4

Campaign information

Name: Test
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Date: 2018-02-20 13:10:50 UTC

Campaign execution information

Execution ID: CE-3-4
Status: Success
Start Time: 2018-02-20 13:10:54 UTC
End Time: 2018-02-20 13:10:55 UTC

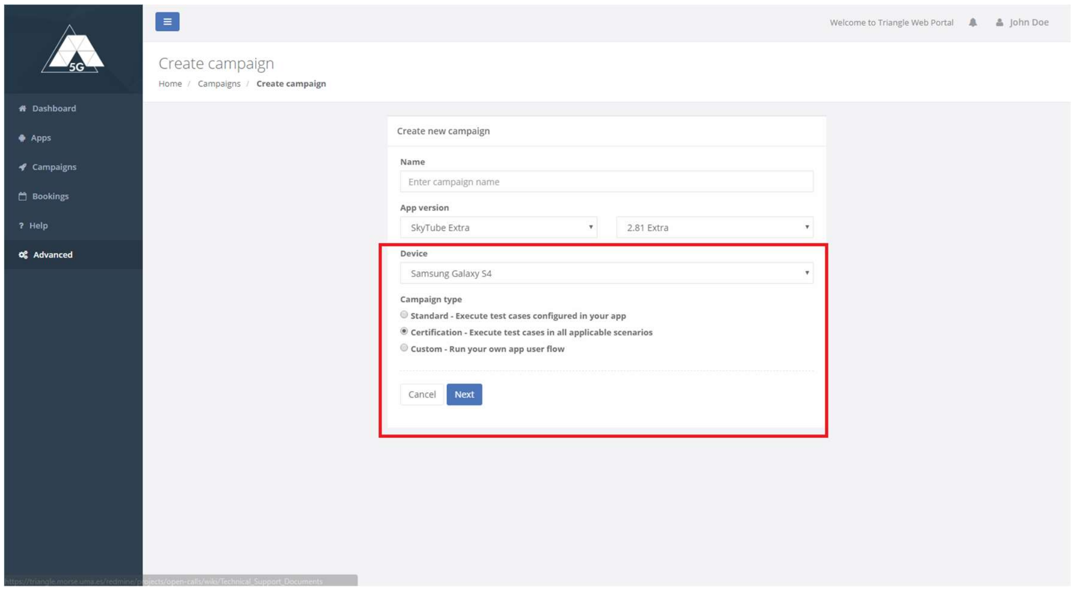
Campaign execution tasks

Status	Test Case	Scenario	Start Time	End Time	Task ID
Success	Play and Pause	High Speed - Direct Passenger Connection		2018-02-20 13:10:55 UTC	4
Success	Stop and Replay	High Speed - Direct Passenger Connection		2018-02-20 13:10:55 UTC	5

Figure 24 Show 'Execution Task ID' in Campaign Execution

3.11.3 Certification campaign

The type *‘Certification’* can be selected in the “Create campaign” form


Welcome to Triangle Web Portal
John Doe
Dashboard
Apps
Campaigns
Bookings
Help
Advanced
Home / Campaigns / Create campaign
Cancel
Next

Create campaign

Create new campaign

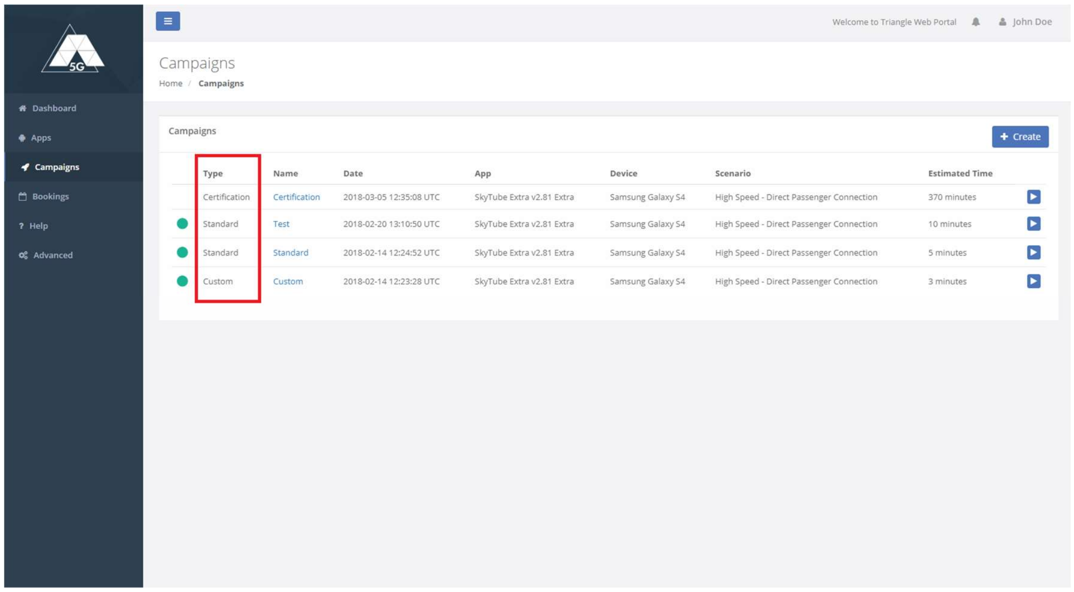
Name: Enter campaign name

App version: SkyTube Extra | 2.81 Extra

Device: Samsung Galaxy S4

Campaign type
- Standard - Execute test cases configured in your app
- [✓ Certification - Execute test cases in all applicable scenarios]
- Custom - Run your own app user flow

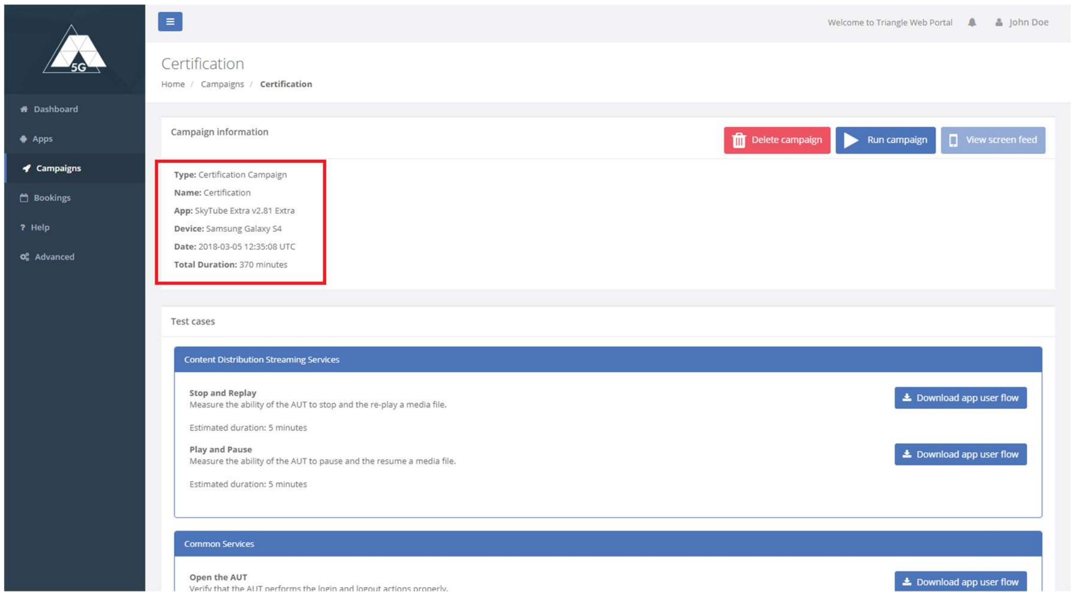
Figure 25 Show 'Certification' type in Campaign creation



Campaigns

	Type	Name	Date	App	Device	Scenario	Estimated Time
	Certification	Certification	2018-03-05 12:35:08 UTC	SkyTube Extra v2.81 Extra	Samsung Galaxy S4	High Speed - Direct Passenger Connection	370 minutes
●	Standard	Test	2018-02-20 13:10:50 UTC	SkyTube Extra v2.81 Extra	Samsung Galaxy S4	High Speed - Direct Passenger Connection	10 minutes
●	Standard	Standard	2018-02-14 12:24:52 UTC	SkyTube Extra v2.81 Extra	Samsung Galaxy S4	High Speed - Direct Passenger Connection	5 minutes
●	Custom	Custom	2018-02-14 12:23:28 UTC	SkyTube Extra v2.81 Extra	Samsung Galaxy S4	High Speed - Direct Passenger Connection	3 minutes

Figure 26 Show Campaign type in Campaigns view



Certification

Home / Campaigns / Certification

Campaign information

Type: Certification Campaign
Name: Certification
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Date: 2018-03-05 12:35:08 UTC
Total Duration: 370 minutes

Test cases

Content Distribution Streaming Services

Stop and Replay
Measure the ability of the AUT to stop and the re-play a media file.

Estimated duration: 5 minutes

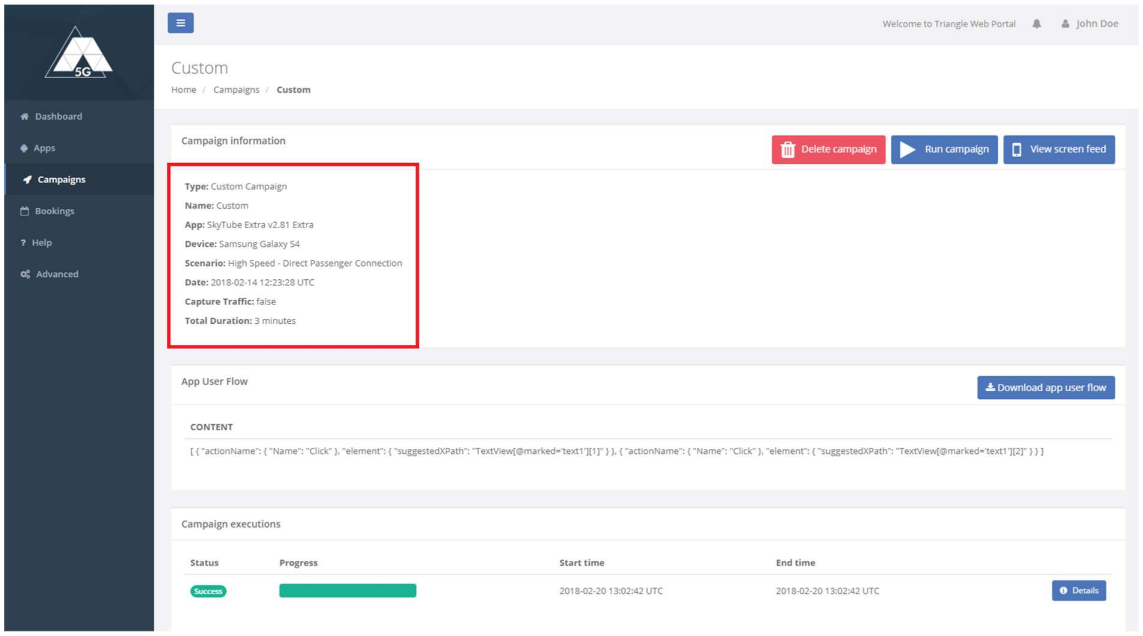
Play and Pause
Measure the ability of the AUT to pause and the resume a media file.

Estimated duration: 5 minutes

Common Services

Open the AUT
Verify that the AUT performs the login and logout actions properly.

Figure 27 Show Campaign type in each Campaign and remove Scenario and Control Traffic if type 'Certification'



Custom

Home / Campaigns / Custom

Campaign information

Type: Custom Campaign
Name: Custom
App: SkyTube Extra v2.81 Extra
Device: Samsung Galaxy S4
Scenario: High Speed - Direct Passenger Connection
Date: 2018-02-14 12:23:28 UTC
Capture Traffic: false
Total Duration: 3 minutes

App User Flow

CONTENT

```
[ { "actionName": { "Name": "Click" }, "element": { "suggestedXPath": "TextView[@marked='text1'][1]" } }, { "actionName": { "Name": "Click" }, "element": { "suggestedXPath": "TextView[@marked='text1'][2]" } } ]
```

Campaign executions

Status	Progress	Start time	End time
Success	🟩	2018-02-20 13:02:42 UTC	2018-02-20 13:02:42 UTC

Figure 28 Show Campaign type in each Campaign and show Control Traffic if type 'Custom'

3.11.4 TRIANGLE Report Generation

The portal includes the capability of generating a TRIANGLE test report in PDF format. This testnreport will provide a top-down view to track the results in each of the domains and test cases executed. The report structure, style and content are based on the template defined in the deliverable TRIANGLE D2.2 (TRIANGLE_D2_2_Apx_Test Report template.dot).

This generator has been implemented as a Python module and it uses the library python-dox [40]. This module has been integrated into the TRIANGLE portal architecture as depicted in Figure 29.There are two inputs to the module: The report template and the actual data, which comes down from the user interface portal in a YAML file.

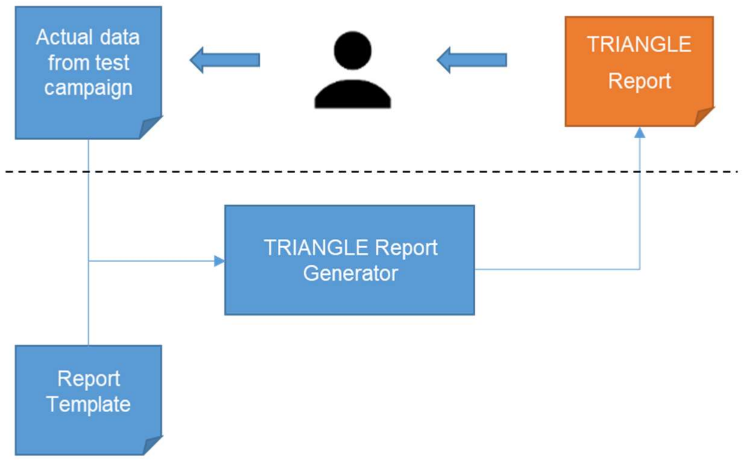


Figure 29 TRIANGLE Report Generation flow

There is an example of report in Annex 9.

4 Orchestration

4.1 Test Automation Platform (TAP)

This section provides the features of the control tool selected to carry out the orchestration of the testbed. This orchestration is needed to coordinate the different components of the testbed. The accuracy provided by OMF is not enough to synchronize the executions of the network behaviours (such as handovers), app actions and the collection of the measurements. In other words, OMF is not well-suited for time sensitive operations.

TAP provides flexible and extensible test sequence and test plan creation. TAP is a Microsoft .NET-based application that can be used stand-alone or in combination with higher-level test executive software environments developed by Keysight. Leveraging C# and Microsoft Visual Studio, TAP is a platform upon which it is possible to build tests solutions.

TAP plays a key role in TRIANGLE as it allows controlling and automating all the instruments present in the testbed. Included with Keysight TAP is the core sequencing engine, tools and plug-ins to minimize test system development time and test execution speed.

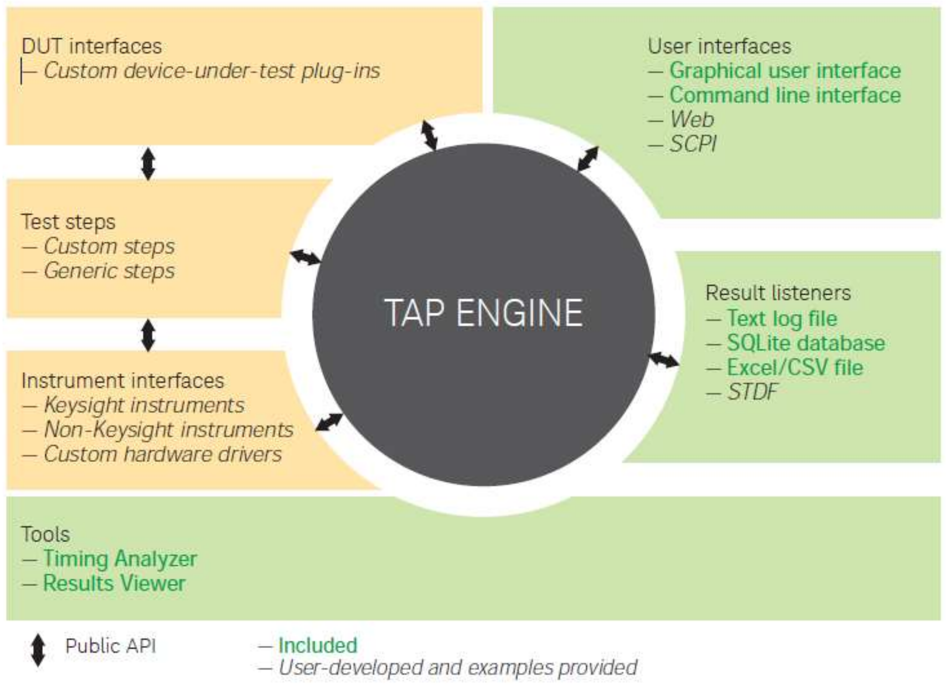
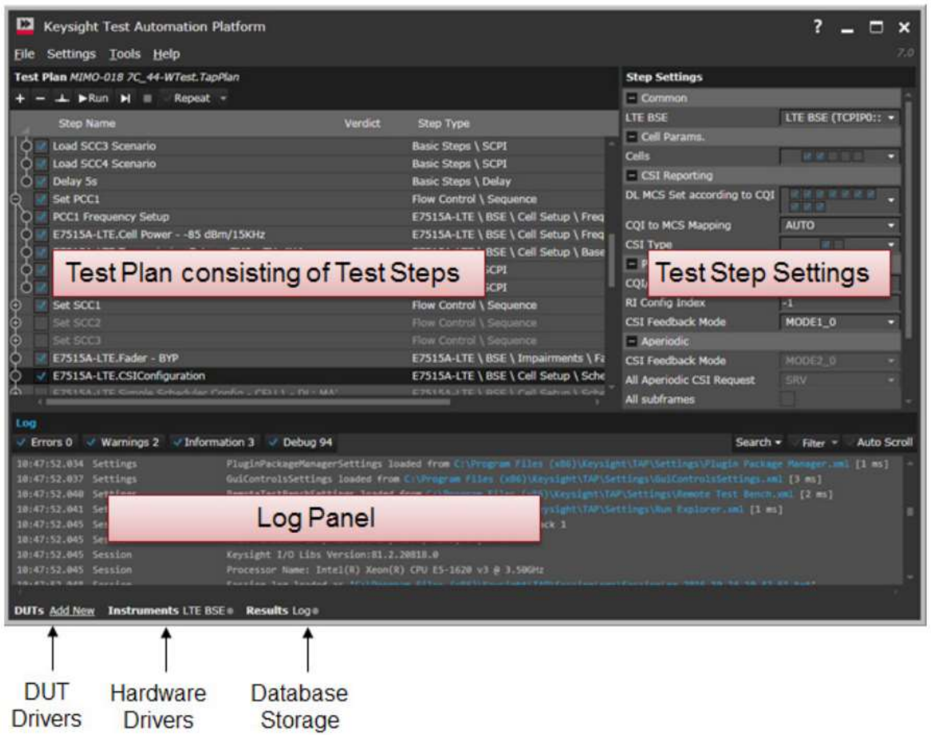


Figure 30 TAP architecture, showing core sequencing software engine.

4.1.1 Core Sequencing Engine

The Core Sequencing Engine is the “heart” of TAP, designed for speed-optimized test step execution. Tests (called test plans in TAP terminology) can include simple flow operations such as IF and LOOP. Complex hardware setups and parallel tests steps are also supported. The graphical interface of the core engine is shown in Figure 31.



Step Name	Verdict	Step Type
Load SCC3 Scenario		Basic Steps \ SCPI
Load SCC4 Scenario		Basic Steps \ SCPI
Delay 5s		Basic Steps \ Delay
Set PCC1		Flow Control \ Sequence
PCC1 Frequency Setup		E7515A-LTE \ BSE \ Cell Setup \ Freq
E7515A-LTE.Cell Power - -85 dBm/15KHz		E7515A-LTE \ BSE \ Cell Setup \ Freq
Set SCC1		Flow Control \ Sequence
Set SCC2		Flow Control \ Sequence
Set SCC3		Flow Control \ Sequence
E7515A-LTE.Fader - BYP		E7515A-LTE \ BSE \ Impairments \ Fa
E7515A-LTE.CSIConfiguration		E7515A-LTE \ BSE \ Cell Setup \ Sche

Step Settings

Common	
LTE BSE	LTE BSE (TCPIP0::
Cell Params.	
Cells	
CSI Reporting	
DL MCS Set according to CQI	
CQI to MCS Mapping	AUTO
CSI Type	
RI Config Index	-1
CSI Feedback Mode	MODE1_0
Aperiodic	
CSI Feedback Mode	MODE2_0
All Aperiodic CSI Request	SRV
All subframes	

Log: Errors 0, Warnings 2, Information 3, Debug 94

```
10:47:52.034 Settings    PluginPackageManagerSettings loaded from C:\Program Files (x86)\Keysight\TAP\Settings\Plugin Package Manager.xml [1 ms]
10:47:52.037 Settings    GuiControlsSettings loaded from C:\Program Files (x86)\Keysight\TAP\Settings\GuiControlsSettings.xml [3 ms]
10:47:52.040 Settings    RemoteTestBenchSettings loaded from C:\Program Files (x86)\Keysight\TAP\Settings\Remote Test Bench.xml [2 ms]
10:47:52.041 Se                                                          ysight\TAP\Settings\Run Explorer.xml [1 ms]
10:47:52.045 Se                                                          ck 1
10:47:52.045 Se
10:47:52.045 Session     Keysight I/O Libs Version:81.2.20818.0
10:47:52.045 Session     Processor Name: Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz
```

Figure 31 TAP's Core Sequencing Engine and GUI

4.1.2 Timing Analyzer

TAP's Timing Analyzer Tool provides insights into optimizing the overall test execution speed. It also allows visualizing the overall and in depth test execution time to see how much time each test step requires. The Timing Analyzer Tool is shown in Figure 32.

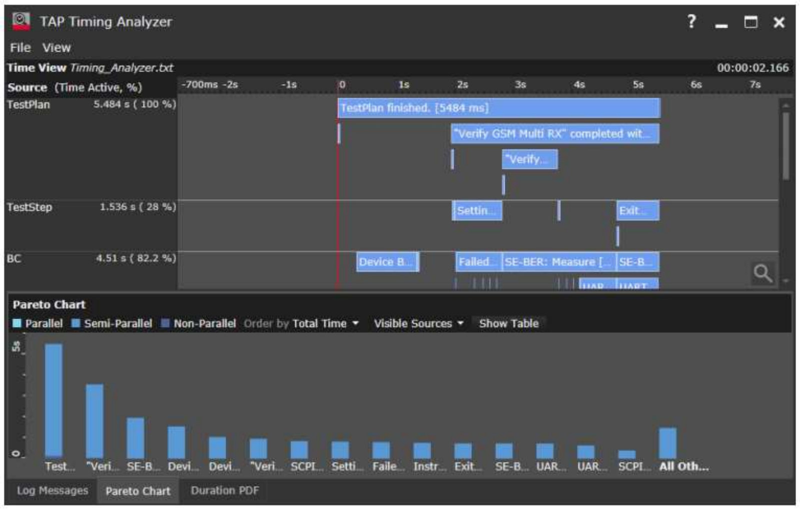


Figure 32 TAP's TAP's Timing Analyzer test step's execution information

4.1.3 Results Viewer

Each time a TAP test plan is executed the results are stored in a database in TAP which can be graphed and visualized using the TAP Results Viewer. Multiple data sets can be viewed to quickly compare results across different test runs (i.e., several executions of the same test). The Run Explorer helps manage test plan data, recall old test plans, merge and compare test log timings, compare test plan settings, search for specific test results, and plot them using the Results Viewer.

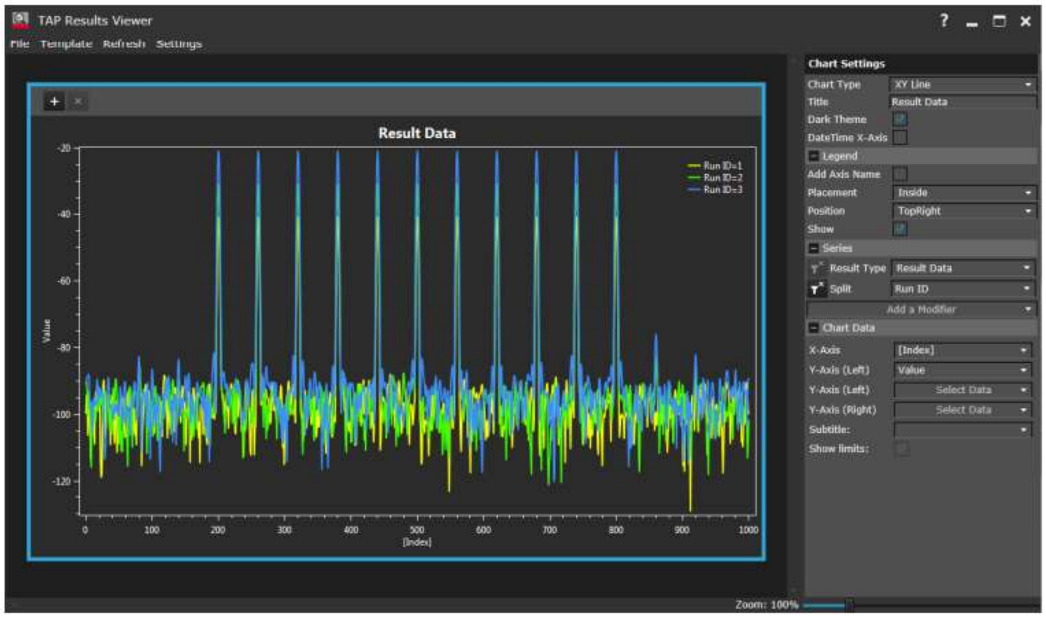


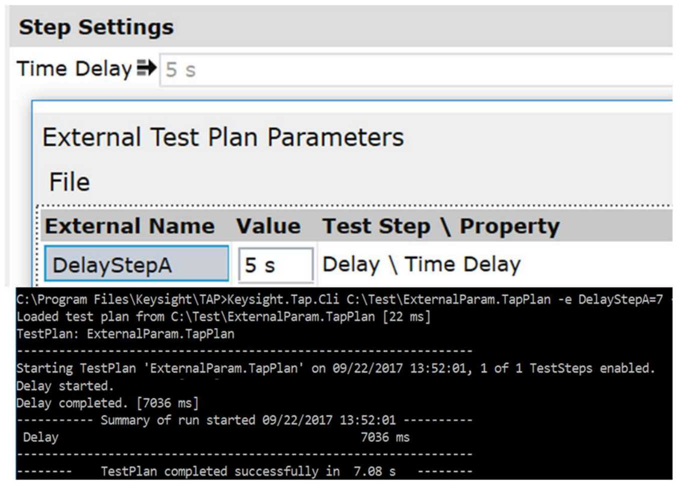
Chart Settings	
Chart Type	XY Line
Title	Result Data
Dark Theme	✓
DateTime X-Axis	
Legend	
Add Axis Name	
Placement	Inside
Position	TopRight
Show	✓
Series	
Result Type	Result Data
Split	Run ID
	Add a Modifier
Chart Data	
X-Axis	[Index]
Y-Axis (Left)	Value
Y-Axis (Left)	Select Data
Y-Axis (Right)	Select Data
Subtitle:	
Show limits:	

Figure 33 TAP Result Viewer provides a quick and flexible test run data visualization.

4.1.4 Test Plan Reference called as External Parameter

In TAP terminology, a Test Plan Reference test step is a special test step which allows TAP to call any 3rd party test plan to be executed in a sequence as a single test step. Despite the potential complexity of the called test plan, this is transparent to the original test plan. The parent TAP test plan cannot modify fields within the called test plan, guaranteeing no interference between master test plan and called test plan when using the test plan reference test step.

TAP additionally allows some fields within the test steps to be declared as External Parameters. These fields have a default value, but when launching TAP from a command line interface, can have their value replaced by the command line requested value, further increasing the flexibility of a single test plan. Indeed, the user can reuse a single TAP test plan with a large combination of parameter values, simply by declaring them as external and crafting command line calls with different values.



External Name	Value	Test Step \ Property
DelayStepA	5 s	Delay \ Time Delay

```
C:\Program Files\Keysight\TAP>Keysight.Tap.Cli C:\Test\ExternalParam.TapPlan -e DelayStepA=7
Loaded test plan from C:\Test\ExternalParam.TapPlan [22 ms]
TestPlan: ExternalParam.TapPlan
-----------------------------------------------------------------
Starting TestPlan 'ExternalParam.TapPlan' on 09/22/2017 13:52:01, 1 of 1 TestSteps enabled.
Delay started.
Delay completed. [7036 ms]
----------- Summary of run started 09/22/2017 13:52:01 ----------
 Delay                                                 7036 ms
-----------------------------------------------------------------
--------    TestPlan completed successfully in  7.08 s   --------
```

Figure 34 External Parameter setting & command line execution

Figure 34 shows an example of a TAP test plan which is created with a Delay step, using a Time Delay parameter called “DelayStepA” set to a default value of 5 seconds. At execution, the external parameter is forced to 7 seconds and the delay step executes for roughly 7 seconds.

TAP 8 combines the two above-mentioned features to bring increased flexibility in the creation of dynamic test plans, by enabling Test Plan Reference test steps to be called as external parameters. This means that whole subsections of TAP test plans can be replaced at each test execution, by pointing the external parameter’s value to another 3rd party test plan.

4.1.5 TAP server

TAP offers the new feature to run as a server, listening to remote connections received via TCP or through REST API, to execute test cases at will. This enables a lab PC running TAP to receive asynchronous test requests from web interfaces, without exhibiting the access to the lab to the experimenter.

4.1.6 .NET Core

TAP moved from .NET to .NET Core to increase its compatibility with more platforms and be more flexible in its deployment.

4.1.7 TAP plugins

This section introduces the TAP plugins implemented to configure and control the different components of the TRIANGLE testbed. Figure 35 shows the hardware and software components of the testbed and the drivers associated to each one of them.

In TAP, an Instrument is a logical entity that encapsulates the interaction with a physical instrument. At the very least, a TAP instrument must define all the necessary logic for connecting and disconnecting the TAP host machine with the real instrument, and it is best practice to define methods for performing every possible (or required) actions that the instrument can execute.

For example, a TAP instrument for controlling a real power supply via SCPI must include two methods (Open and Close) that create and release the connection with the instrument and can have two extra methods for setting the voltage and current.

Instruments are extensively used in TAP steps, which expose the instrument functionality to the end user. Continuing with the previous example, we could define two steps for this instrument:

One that turns off the power output, setting voltage and current to 0, and another that sets the voltage and the current using the values selected by the user in the test step configuration. TAP will automatically use the Open and Close methods from the instrument when required.

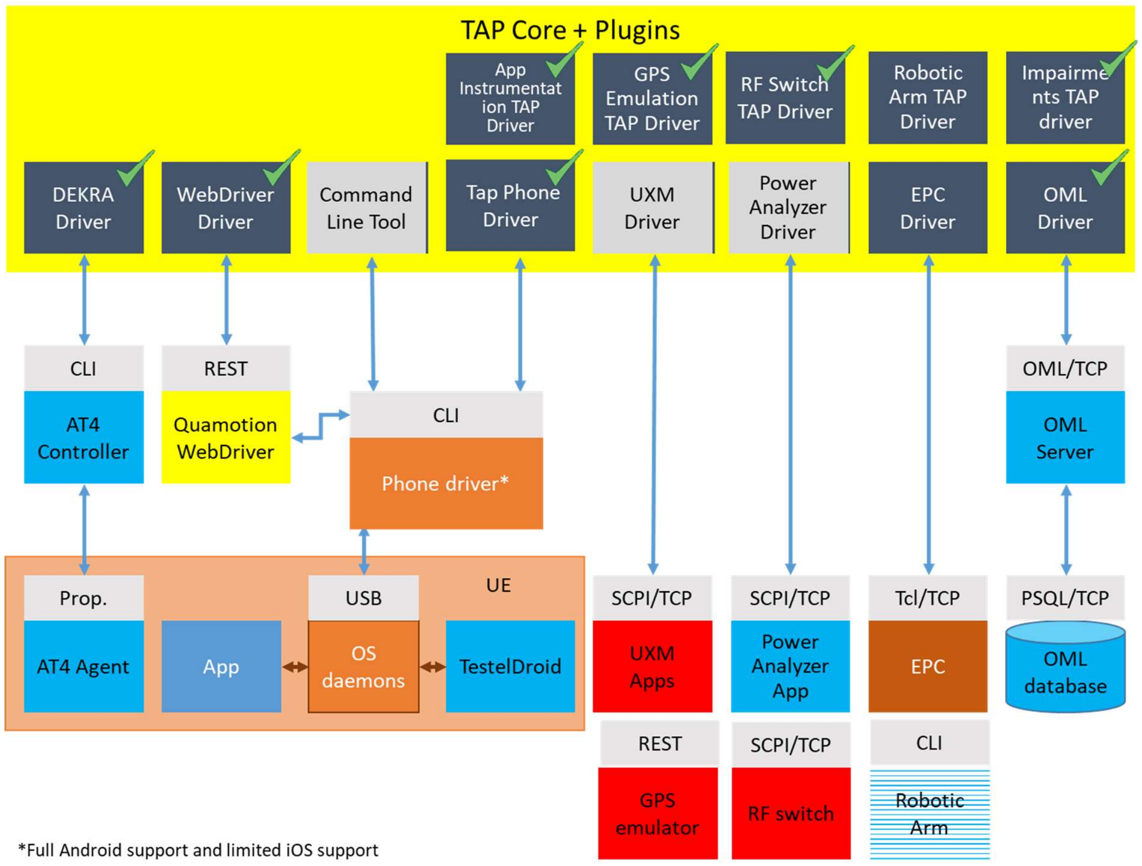


Figure 35 TAP pluging for the hardware and software components of the testbed

Table 7 provides a brief description of each one of the TAP plugins implemented.

Table 7 TAP plugins implemented

TAP plugin	**Description**
Quamotion WebDriver	This plugin allows TAP to send user actions (such as tapping a button or entering a text in a field) through the use of Quamotion WebDriver. The plugin will provide an instrument to connect to the Quamotion WebDriver, as well a series of test steps.
OML server	This plugin allows TAP to send the results reported during an experiment to an OML server (which will store them in a database).

Instrumentation library	This plugin allows TAP to parse logs from devices to extract measurements produced by the instrumentation library. These measurements will be published and processed by the corresponding result listeners. This plugin does not provide any additional instruments.
Android	This plugin provides a collection of steps that can be used for controlling an Android device connected to the TRIANGLE testbed through the Android Device Bridge.
iOS	This plugin allows TAP to control an iOS device in order to perform key actions, such as restart, save logs, capture network traffic and launch apps. An iOS device can be an iPhone, iPad or iPod Touch.
RF switch	This plugin provides steps and an additional instrument for controlling a LXI-compliant 11713C attenuator/switch driver available on the TRIANGLE testbed. This switch driver is used alongside Keysight L7104A electro-mechanical switches, in order to allow the users to select one of the available devices in the testbed at any given time.
GPS emulation	This plugin provides the instruments and the steps to configure and control the GPS emulation feature introduced in Section 11.
Dynamic Sequence Plugin	In order to efficiently enable the parallel test execution structure described in 4.2.5, a new plugin needs to be introduced, called DynamicSequence, has been introduced.
Iteration-aware result listener plugin	A test case requires to be run multiple times to reach statistically meaningful and converged test results. Results from each iteration need to be saved separately to calculate KPIs, and pinpoint possible sporadic performance outliers. To achieve iteration-aware tagging of test results, a new Result Listener has been added to TAP, injecting the application flow iteration into the test results.
KPIs calculation plugin	The plugin is aware of the domain, use case and test case of the results it receives as input, and calculates the relevant KPIs for this combination.
Plugins to reach TAP server	This plugin enables to reach remotely a TAP server which implies that the Orcomposutor can reach a TAP server installed in a different machine to run the composed TAP test plans.
DEKRA	This plugin enables to control the DEKRA Performance Tool from TAP to collect UE measurements during the executions of the TAP test cases.

VELOX	This plugin enables to use from TAP the RedZinc VPS Engine or Velox, an over-the-top-content enabler.
MANO	This plugin enables to configure and control the Management and Network Orchestration from TAP plugin.
StreamOwl	This plugin is using to control the video analysis tools developed by StreamOwl.

4.1.8 TAP Test Plan Master Template

The TAP test plan master template includes the sequence of actions that will be executed during the campaigns. The orcomposutor creates an instance of this template each time it executes a test case. The information provided by the user and the measurements defined in the test cases are used to deliver the final instance of this template that will be executed.

The final master template includes:

- Setting of the test plan reference as external parameters, to be called via command line
- All parameters of these test plan reference are set as external parameters within these test plans, so that they can be set again as external parameters in the master template
- Simultaneous capture of
    - Cellular statistics and device resource usage using the TACS4 agent and DEKRA TAP plugin, no matter the application flow.
    - Testeldroid logs (delivers PCAP files useful for traffic analysis).
    - Android Logcat (provides Android debug logs useful for all application domains).
    - Power consumption of the DUT via USB as well as fake battery connector.

The overall master template follows the high-level structure below:

1. Test case labeler
2. Instruments configuration
    1. UXM configuration
    2. Power analyzer configuration
    3. Other instruments configuration
3. Connect the DUT
4. Configure the DUT
5. Initialization of the TACS4 agent
    1. Configure the device automation profile to idle
6. Test body (repeat over X iterations)
    1. Test execution (parallel execution)
        A. Application flow
            1. Start of non-blocking measurements (Testeldroid, WebDriver)
            2. Execution of blocking measurements (parallel)
                A. TACS4 test run
                B. App flow run (Quamotion flow execution)
            3. Stop of non-blocking measurements
        B. Network scenarios
            1. Subscenario 1
            2. Subscenario 2
            3. Subscenario N

    2. Retrieve data per iteration
    3. Connectivity clean-up
7. Retrieve data per test case

Notes:

- The bullets marked 1. or 2. are executed successively, whereas bullets A. and B. are running simultaneously (in parallel).
- Most of the bullets above are actually references to other test plans, meaning that the template keeps its generic structure even if the nature of the DUT or test case changes
- The measurements are split into blocking (which will keep the test flow on hold until they are completed) and non-blocking (which are started in a single step and run in the background). For this reason, the blocking measurements are all combined in a parallel loop, to avoid blocking the rest of the test case logic.
- The center-core test step which dictates the actual duration of the test case still is the Quamotion application flow (replay of a powershell script through the Quamotion plugin). All other measurements or network scenario emulation are started before the app flow replay starts, and are stopped right after it stops, as only measurements when the app flow is running will be kept for post-processing.

4.1.9 DEKRA tool TAP plugin

From Release 2 onwards DEKRA has provided several updates of the DEKRA tool TAP plugin. The plugin change log can be summarized with the following highlights:

- Support for TAP 8
- Contest Stall measurements (see section 5.2.10)
- GPU Usage measurements (see section 5.2.10)
- Robotic arm integration
- Minor bugs and other maintenance updates

Out of these upgrades the major one has been the implementation of the TAP plugin for the integration of the robotic arm platform [D3.2]. The TAP plugin was implemented during the Release 2 timeframe, an integrated in Release 3 of the TRIANGLE testbed.

The robotic arm platform was implemented to meet the technical requirements derived from the Virtual Reality, Gaming and Augmented Reality test specification, mostly on the ability to physically move the device.

From Release 3 onwards DEKRA has provided several updates of the DEKRA tool TAP plugin. The plugin change log can be summarized with the following highlights:

- The radio and system monitor measurements have been changed to be reported in rows.
- The robotic arm with iOS support has been integrated.
- The data services can be loaded in a TAP test plan.
- Integration of the WLAN Access Point automation module (see section 5.2.10).

- Minor bugs and other maintenance updates

In Release 4, the module includes support for iOS devices and the TAP plugin has been updated accordingly.

In addition to supporting iOS some improvements have been implemented in the robotic arm module and properly integrated in the TRIANGLE testing framework:

- A “timeout” parameter has been added in the functions which require Image Recognition. In Release 3 the timeout was hardcoded to 30 s.
- A “speed” parameter has been added in the functions which require movements of the robotic arm. In Release 3 the speed of the arm was hardcoded to “very fast” leading to high aiming failure rate.
- A “matching score” parameter has been added in all the functions which require Image Recognition. In Release 3 the timeout was hardcoded to 90 %.
- New function to support “press” (in Release 3 only short tap and swipe was implemented).
- New function to implement “lagging” performance indicator. This function finds and taps on a target image and then it measures the time to find a second image. This function is intended to implement the performance indicator “lagging” (response time) for Gaming use case.

Annex 8 presents the modifications (compared to Release 3) on the specification of the Remote-Control Interface service.

4.2 Quamotion Automation Tools

The Quamotion Automation Tools have been used in the context of the project as interface to interact with the apps and the devices under test.

The Quamotion WebDriver is able to automate user actions on iOS and Android applications whether they are native, hybrid of fully web based. The Quamotion WebDriver handles the full lifecycle of app usages (installation, starting and automation) without any manual interaction. The only requirements are a valid application package (apk or ipa file) and in case of iOS a DeveloperProfile and the iOS DeveloperDisks.

The design of Quamotion WebDriver follows the specifications of the W3C WebDriver specification. In origin, the WebDriver provides a platform- and language-neutral protocol to remotely instruct the behaviour of web browsers.

The Quamotion WebDriver is an extension on the WebDriver specification and adds specific support to manage mobile devices and mobile applications. The Quamotion WebDriver is able to instruct the behaviour of mobile application similar to instructing the behaviour of web browsers.

The following instructions are added on top of the standard WebDriver instructions in order to support test automation for mobile applications.

Table 8 Command reference for mobile extensions to the WebDriver

HTTP method	Path	Summary
POST	/session	Creates a new session, starting the application on the device.

PUT	/quamotion/developercenter/profile	Creates a new Apple developer profile.
POST	/session/:sessionId/quamotion/elementByCoordinates	Search for a visible element in the page which matches the coordinates
POST	/session/$sessionId/touch/clickByCoordinate	Clicks on the visible element which matches the coordinates
POST	/quamotion/app	Adds an app to the Quamotion WebDriver app repository.
GET	/quamotion/app	Gets all applications in the Quamotion WebDriver app repository
GET	/quamotion/device	Gets all devices which are available to test
POST	/quamotion/device/$deviceId/app/$appId	Installs the application on the device
POST	/quamotion/device/$deviceId/app/$appId/$appVersion	Installs the application on the device
GET	/quamotion/device/$deviceId/app	Gets all installed applications on the device
DELETE	/quamotion/device/$deviceId/app/$appId	Uninstall the application from the device
DELETE	/quamotion/device/$deviceId/app/$appId/$appVersion	Uninstall the application from the device
DELETE	/quamotion/device/$deviceId/app/$appId/settings	Deletes the settings of the application installed on the mobile device
DELETE	/quamotion/device/$deviceId/reboot	Reboots the device

App flow scripts can be written in multiple languages e.g. PowerShell, Java and C#. Within the TRIANGLE project PowerShell has been selected because:

- PowerShell provides the necessary flexibility
- PowerShell is a script language and does not require compilation after each change
- PowerShell has minimal installation overhead. Windows 10 comes by default with a PowerShell editor.

A Powershell script looks as follows:

```
Click-Element -xpath "input[@name='email']"
Enter-Text "bart.saintgermain@quamotion.mobi"
Click-Element -xpath "input[@name='pass']"
Enter-Text "a strong password"
Click-Element -xpath "button[@name='login']"
```

The script above automates the login activity. It first clicks on the email text field and enters the email. Next it clicks on the password field and enters the password. Finally, a click is performed on the login button.

The Quamotion WebDriver provides through the Quamotion Frontend all necessary tools to generate an app user flow script.

4.2.1 Quamotion WebDriver support

The Quamotion WebDriver allows to create and execute user flows on both Android and iOS devices. Versions from 5% market share are actively supported including most recent Android and iOS versions.

The Quamotion WebDriver is available for Windows, Linux and Mac computers.

Two types of automation exist, application automation and device automation.

Application automation instruments the application to automate any gesture, find elements and asses element properties from within the application sandbox.

To do so the application (ipa/ apk) is unpacked, altered (without modifying the app logic), packed, resigned, pushed to the device, installed and started with Quamotion support. This is an automatic process and does not require any manual step.

Advantages:

- Speed
- Access to any element property
- Recording the app user flow is supported

Disadvantages:

- No possibility to automate OS specific elements (e.g. settings, push notifications, …)
- Resigning with non-original certificate is detected by some applications and triggers different behaviour or blocks the launch of the application.
- Limited support for WebViews

Device automation automates the device UI by making use of operating system hooks. These hooks are available for both Android and iOS. An agent is installed on the mobile device which is responsible to handle all automation requests.

Advantages:

- Possibility to automate any element in the screen.
- No application resign needed.
- Support for WebViews

Disadvantages:

- Only limited access to accessible UI properties.
- No application flow recording is supported.
  note: application flow editor (Spy and code generator) can still be used
- Slower execution and Spy.

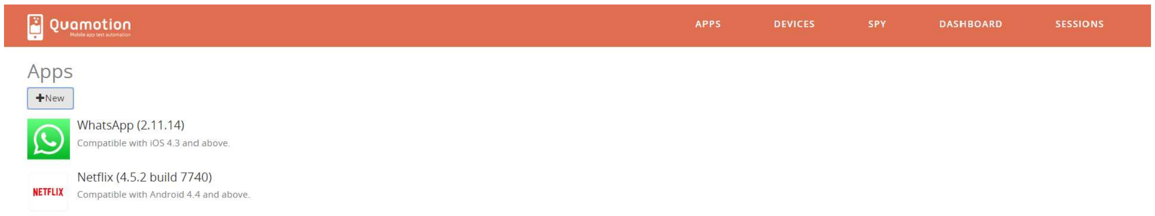
- Agent has bigger impact on device resources like CPU and battery usage.

4.2.2 Quamotion Frontend

The Quamotion frontend is web based and composed out of an “Apps” page, “Devices” page and a script editor of “Spy” page.

App management

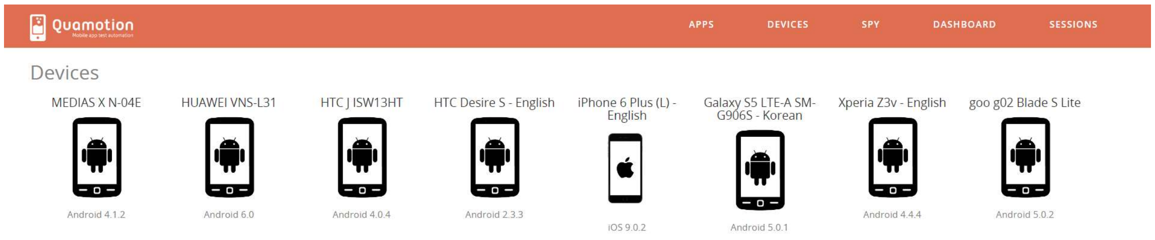
With the App management you can list, add and remove applications to the Quamotion WebDriver application repository. Information about the application e.g. unique application id, version number, can be retrieved.



WhatsApp (2.11.14)
Compatible with iOS 4.3 and above.
Netflix (4.5.2 build 7740)
Compatible with Android 4.4 and above.

Device management

The device management lists all devices connected to the computer running the Quamotion WebDriver. Basic information about the device can be retrieved, e.g., the serial number of the device.

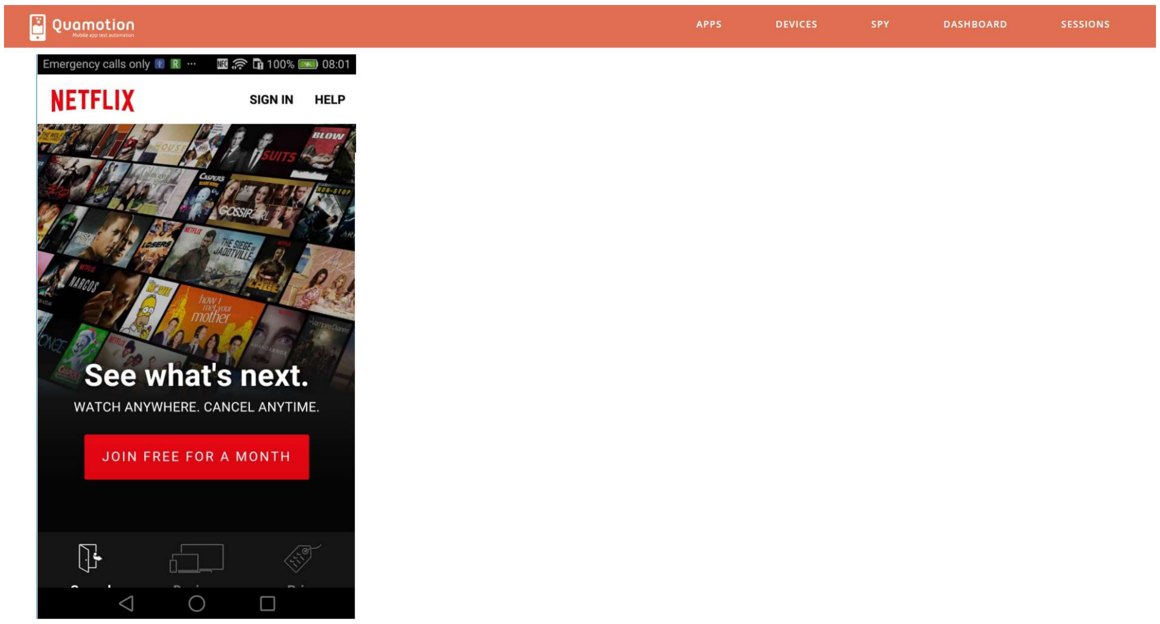


MEDIAS X N-04E	HUAWEI VNS-L31	HTC J ISW13HT	HTC Desire S - English	iPhone 6 Plus (L) - English	Galaxy S5 LTE-A SM-G906S - Korean	Xperia Z3v - English	goo g02 Blade S Lite
Android 4.1.2	Android 6.0	Android 4.0.4	Android 2.3.3	iOS 9.0.2	Android 5.0.1	Android 4.4.4	Android 5.0.2

Selecting the device gives you the ability to launch Remote Control.

A performant screenshot feed is made available mirroring the mobile device screen. On Android the screenshot feed goes up to 40 fps on iOS 10 fps. This videofeed can be captured and stored on disk by standard tools like ffmpeg.

The Quamotion WebDriver comes with the ability to remotely control a mobile device from the chrome browser or from a standard VNC client.



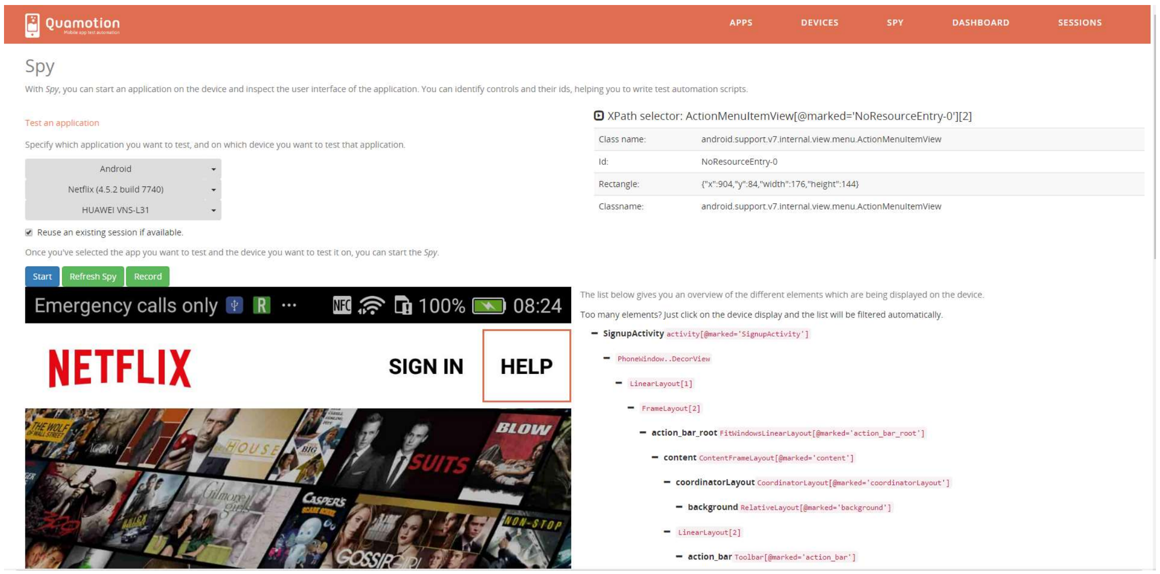
4.2.3 Quamotion Script Editor

The Quamotion script editor contains several tools to assist a user in creating script: a spy, a recorder tool and an editor tool.

Spy

Based on an active session and the remote device display, users are able to click on an element in the screen and retrieve all information about the user interface widget being selected. The spy shows the following information

- The XPath of the widget identifying uniquely the widget, which can then be used in the script to find elements
- A tree showing the ancestors of the widget.
- All available properties of the selected widget
- The widget is highlighted in the remote device screen



Spy

With *Spy*, you can start an application on the device and inspect the user interface of the application. You can identify controls and their ids, helping you to write test automation scripts.

Test an application

Specify which application you want to test, and on which device you want to test that application.

Android
Netflix (4.5.2 build 7740)
HUAWEI VNS-L31

☑ Reuse an existing session if available.

Once you've selected the app you want to test and the device you want to test it on, you can start the *Spy*.

XPath selector: ActionMenuItemView[@marked='NoResourceEntry-0'][2]

Class name:	android.support.v7.internal.view.menu.ActionMenuItemView
Id:	NoResourceEntry-0
Rectangle:	{"x":904,"y":84,"width":176,"height":144}
Classname:	android.support.v7.internal.view.menu.ActionMenuItemView

The list below gives you an overview of the different elements which are being displayed on the device.

Too many elements? Just click on the device display and the list will be filtered automatically.

```
− SignupActivity activity[@marked='SignupActivity']
  − PhoneWindow..DecorView
    − LinearLayout[1]
      − FrameLayout[2]
        − action_bar_root FitWindowsLinearLayout[@marked='action_bar_root']
          − content ContentFrameLayout[@marked='content']
            − coordinatorLayout CoordinatorLayout[@marked='coordinatorLayout']
              − background RelativeLayout[@marked='background']
            − LinearLayout[2]
              − action_bar Toolbar[@marked='action_bar']
```

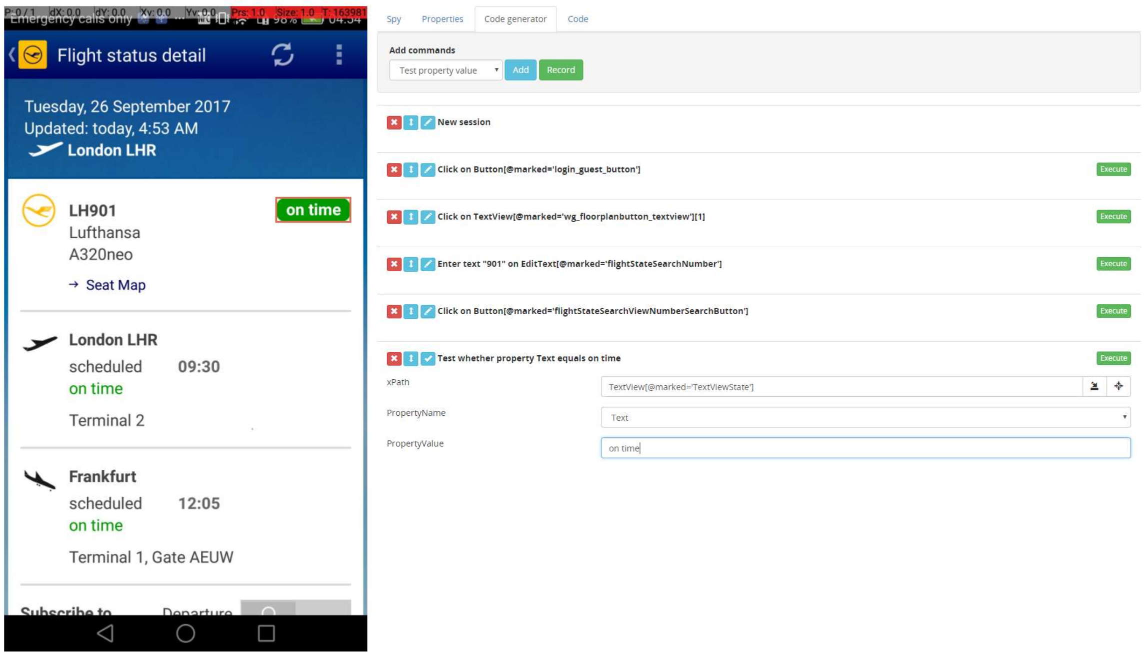
Recorder

The recorder generates a script based on the user interactions on the device. Gestures like Tapping and entering text are recorded. For each user action on the device a command is generated.

Note that the recorder will not generate automatic checkpoints. After recording, the script needs to be fine-tuned and enhanced manually in the editor.

Editor

The script editor allows to create and edit an application flow for both Android as well as iOS applications. There is no need to write code for basic scenarios.



New session

Click on Button[@marked='login_guest_button']

Click on TextView[@marked='wg_floorplanbutton_textview'][1]

Enter text "901" on EditText[@marked='flightStateSearchNumber']

Click on Button[@marked='flightStateSearchViewNumberSearchButton']

Test whether property Text equals on time

xPath: TextView[@marked='TextViewState']

PropertyName: Text

PropertyValue: on time

Figure 36 Quamotion WebDriver script editor

The script editor provides templates for the most common commands such as those listed in table 5

Table 9 Commands provided by the Quamotion Webdriver script editor

Command	**Description**	**Properties**
New session	Create a new session	
Remove session	Remove the current session	
Click element	Click on an element with the given xPath	xPath
Send keys	Send keys to the keystroke	Text
Dismiss keyboard	Dismiss the keyboard	
Clear text	Clear the text of the active text field	
Enter text	Enter text performs: 1. Click element 2. Clear text 3. Send keys 4. Dismiss keyboard	xPath Text
Go back	Press back button	

Implicit wait	Set the maximum allowed time to wait for an element	Time (milliseconds)
Explicit wait	Wait for the given time	Time (milliseconds)
Test element	Test whether an element with the given xPath exits	xPath
Test property value	Validate a property of an element	xPath Property name Expected property value
Get property	Get the value of a property	xPath Property name
Set property	Set the value of a property	xPath Property name Property value
Get element	Get the first element corresponding to the given xPath	xPath
Get elements	Get all elements corresponding to the given xPaht	xPath

The application flow can be exported to a PowerShell script. This application flow can be uploaded and used in the TRIANGLE portal.

4.2.4 Script Replay

The script can be replayed in any PowerShell environment.

End users can try the script using e.g. the preinstalled window 10 PowerShell editor. Note that PowerShell is also distributed for other operating systems like Linux, macOS or older windows versions.

In the testbed a docker container containing PowerShell is used to execute the script.

Docker execution

Executing scripts on the portal leads to vulnerabilities in the portal. Not only it is possible to harm the server on which the script is executed but also on the mobile device.

To avoid this, several docker containers are made available to isolate the different components:

- Filter proxy
  Instead of allowing access to all devices the filter proxy allows only connections to one device. This is available for both ADB and usbmuxd.
- Quamotion WebDriver
  Contains the Quamotion WebDriver and is connected to one or more filter proxies
- Quamotion Runner

Contains PowerShell and executes app user flow scripts. A new container is launched for each test run. Harmful scrips will only affect this container.

4.3 Secure Execution of Scripts Generated by Users

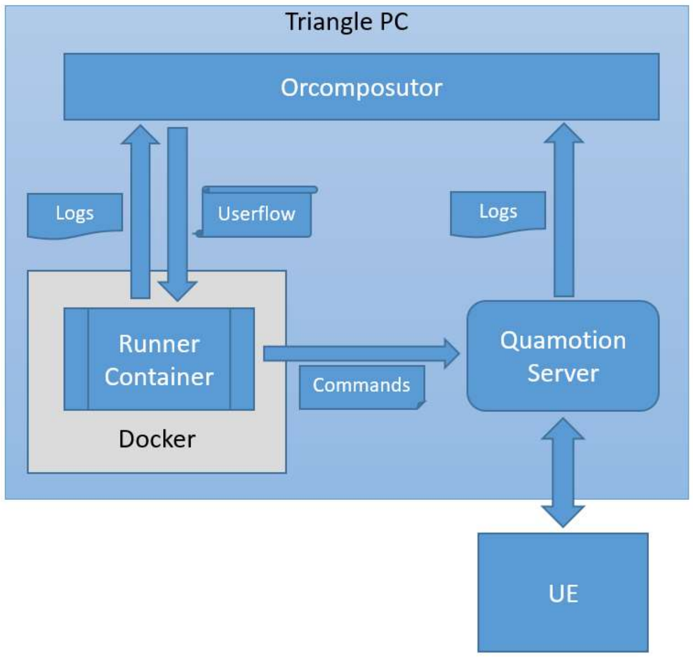


Figure 37 Userflow execution architecture

Due to the flexibility of Powershell scripting it would be possible to include malicious code inside the userflows provided by the users. In order to avoid security issues, we adopted the architecture shown on Figure 37.

For every task in a campaign execution a new Runner container is created using Docker. This container downloads the Userflow script from the Orcomposutor and executes it, sending commands to the Quamotion server running on the host machine. The server communicates with the UE using ADB.

Docker containers have limited access to the resources on the host, which creates a safe environment were the scripts can be executed. These containers are discarded at the end of the test, retrieving only the execution logs in plain text. Because of this, any possible issue caused by a malicious script will only affect the Runner container, which, in turn, is limited to causing the current test to fail.

4.4 Android Debug Bridge (ADB)

Android Debug Bridge (ADB) [5] is a command-line tool provided by the Android SDK. It lets the user communicate with an Android device to perform actions on it, or to send commands to an app. The tool follows a client-server approach, with clients that send commands, daemons that runs the commands on a device, and a server that handles the communication between clients and daemons. The daemons run on the Android devices, will the clients and server typically run on a development machine.

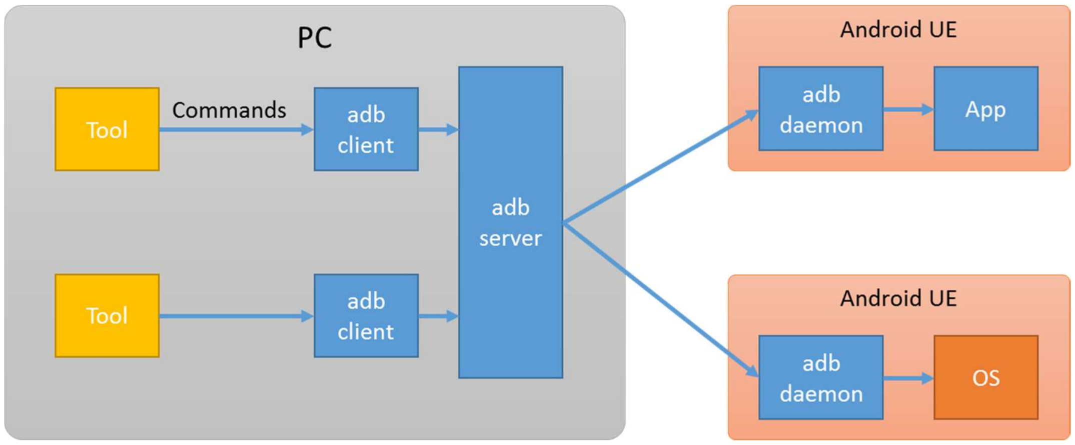


Figure 38 ADB usage scenario

Many tools use ADB to interact with an Android device or the apps running on it. For instance, Quamotion WebDriver uses ADB to send the user actions as commands that can be performed on the device. TestelDroid (introduced in section 5) is controlled through intents (inter-app messages used in Android) sent using ADB.

4.5 PTP Synchronization

Precise Time Protocol (PTP), also known as IEEE 1588, is a communication standard used for clock synchronization between different instruments/machines. The expected accuracy of PTP in a local area is in the order of microseconds. PTP is designed for systems with stricter requirements than what Network Time Protocol (NTP) protocol can provide.

PTP is used in the testbed to keep accurate time synchronization between the different instruments in the testbed.

4.6 Orcomposutor

The ORchestrator-COMPOSer-execUTOR (Orcomposutor) is a server with a REST API that runs on the same Windows machine as TAP. Its purpose is to bridge the Portal and TAP.

Once all the required information has been entered in the Portal, the TRIANGLE testbed end user can proceed with the experiment. The first step would be to take the information entered and turn it into executable TAP test plans. This is the task of the test plan Orcomposutor. According to the features introduced in the Portal for the product under test, the Orcompusutor will generate the applicable test plans.

To create the required TAP test plans, the Orcomposutor uses pre-defined TAP test plan templates. When possible, the Orcomposutor will take advantage of two TAP features: the ability to expose parameters of a test step to external callers, and a test step that allows the execution of another test plan.

For instance, many test plans will start by setting up the network scenario and configuring the required parameters in the Testbed equipment. This setup is the same, regardless of the body of the test plan. Thus, the Orcomposutor reuses existing TAP test plans that configure particular network scenarios.

For an app test, the body of the test plan typically includes replaying the user actions contained in an app user flow provided by the app developer. The Orcomposutor gets the app user flow from the Portal, and sets the corresponding external parameter of the WebDriver replay test step.

The Orcomposutor is also aware of which KPIs are going to be measured with each of the generated TAP test plans. If necessary, the test plan should provide explicit support for performing the measurements required for the KPIs. For instance, if a test plan will contribute to a KPI on power consumption, the power analyser must be configured and used in the test plan. In addition, the information on which KPIs are going to be measured by each test plan must be passed along in the workflow.

Each of the TAP test plans created by the Orcomposutor can then be executed in the Testbed using TAP. The TAP test plan contains all the information required to execute a test automatically.

During the execution of the TAP test plan, the measurement tools will gather measurements. The measurement tools that are fully integrated with TAP will publish them as usual. In this case, the results will be handled by a TAP result listener that sends them to a central OML server. This OML server uses a PostgreSQL database server to store the measurements. Some tools may include OML support, and thus send their measurements directly to the OML server.

The main functions of Orcomposutor can be summarized as follows:

- Accept test campaign execution requests from the Portal.
- Compose the TAP test plans required to run a test campaign and its test cases.
- Execute the TAP test plans.
- Upload the results of the execution to the Portal.

To carry out these functions, Orcomposutor needs to communicate with the REST API of the Portal backend (described in Section 3.5), and with the OML database.

4.6.1 REST API

Orcomposutor has a REST API that exposes the following methods:

Table 10 Orcomposutor REST API

URL	**Description**	**Caller**
[GET]*/run_campaign?id=<id>*	Request to execute a campaign.	Portal
[GET]*/retry_campaign\?id=<id>&execution_id=<e_id>*	Requests to retry a campaign execution.	Portal
[GET]*/generate_user_flows\?id=<id>*	Requests the generation of userflows for Model Based Testing campaigns	Portal
[GET]*/download_user_flow*	Serves a copy of the current userflow.	Runner container

[GET]*/upload_user_flow_log*	Serves the userflow execution logic	Runner container
[GET]*/available*	Check Orcomposutor availability	Queue manager Administration console

These method returns immediately. In the case of the Portal the backend will be updated periodically with the progress of the test campaign execution or the status of the userflow generation.

4.6.2 TAP Test Plan Composition and Execution

The request to execute a test campaign only includes the identifier of that campaign. Orcomposutor uses that id to request more information about the test campaign to the backend using its REST API. This information is used to determine which test case or test cases must be executed with TAP. A test campaign might include more than one test case, in that case the Orcomposutor will prepare the execution of more than one TAP test plan.

TAP test plans contain several external parameters and test plan references that must be filled in before execution can start. Orcomposutor must retrieve the appropriate information from the backend REST API in order to fill in these blanks, such as the id of the device used in the test case, or the network scenario. In the latter, Orcomposutor must select the appropriate TAP test plans that include the initialization and dynamic configuration for each scenario. We call the selection of this parameters and referenced TAP test plans the composition of the test plan.

Once the TAP test plan has been composed, it can be executed with the TAP CLI. Orcomposutor will store the TAP logs, as well as internal logs for diagnostic purposes.

4.6.3 Results

Once a TAP test plan has been executed correctly, Orcomposutor will upload the corresponding results to the Portal backend. These results include:

- Results collected in the TAP database from testbed instruments and tools (including the instrumentation library)
- Logs from the device (e.g. logcat for Android devices)
- Traffic capture in pcap format

4.6.4 Additional Features

Dynamic Scenarios and TAP Master Template

Executing dynamic scenarios, which make use of the TAP master template, requires a different set of external parameters. It is necessary to use several testplan references: one for configuring the initial network conditions and up to five testplans that are executed randomly simulating the changing conditions of the network.

The TAP master template delegates the configuration of the device under test test (for example, it is possible to use the same master template for Android and iOS devices) and other instruments to specific testplan references. The information about which testplans are required depending on the scenario and test conditions is stored in several files in yaml format, which allows easier customization without the need to edit the source code of the Orcomposutor.

Multiple Executions for All Available Domains

The Orcomposutor is able to execute each test case multiple times with various configurations for all the available domains. The results generated by each of these executions are uploaded to the TRIANGLE Portal organized in different folders inside the generated zip files.

KPI Extraction

The Orcomposutor is able to automatically execute the corresponding KPI extraction steps for any given domain. These KPIs are used during the TRIANGLE Mark calculation and are uploaded with the additional generated measurements to the Portal.

TRIANGLE Mark Calculation

Once the campaign has finished and all the KPIs have been generated, the Orcomposutor will initiate the ETL processing, obtaining the TRIANGLE mark and all the intermediate results of the test. These results will be uploaded to the Portal.

TAP Logging Upload

To facilitate the debugging of failed campaign executions the loggings generated by TAP are uploaded to the Portal alongside the results. Using these "logs", the experimenters can have a better understanding of the issues encountered during the execution.

Execution Error Management

The TAP master template also includes additional checks for ensuring the correct execution of the App Userflows during a test. Thus, the testbed is able to capture exceptions that may occur while the userflow is being played by checking at the contents of the generated logs.

If the testbed detects an error, the Quamotion server is automatically restarted, and the App Userflow is executed again. This process is repeated up to 3 times, in order to avoid an infinite loop in case of an unrecoverable error.

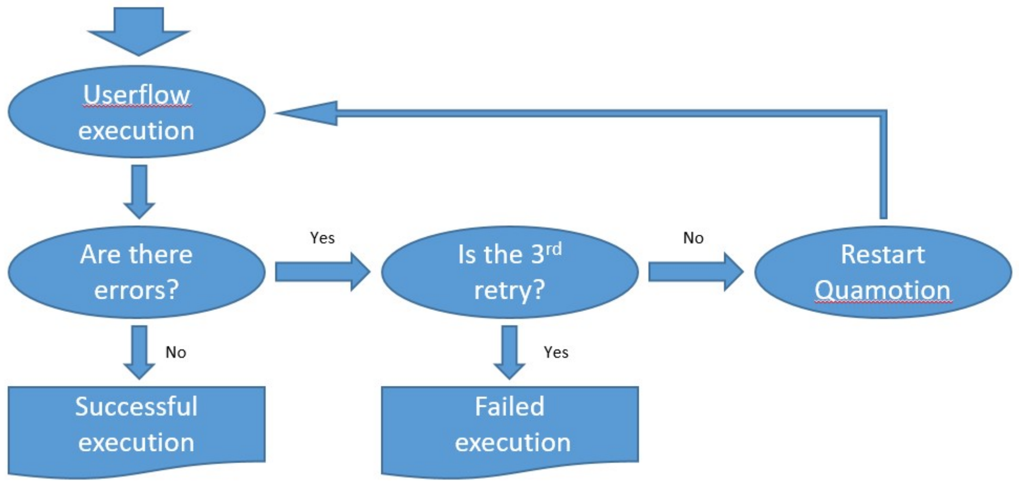


Figure 39 Testbed scheduling and management architecture

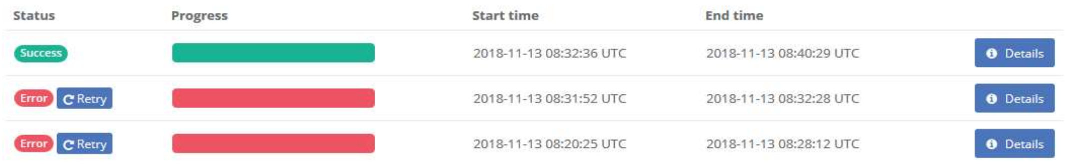
Loose Processes Handling

During a testplan execution and under some rare circumstances, some of the child processes created may reach a state in which they are not correctly closed. This would cause that the communication channel between TAP and Orcomposutor remains open indefinitely, requiring to manually close the remaining processes from the system.

In order to avoid this situation, the Orcomposutor includes the ability to monitor the messages generated by TAP, looking for patterns that signal the end of a testplan execution (such as the closure of the TAP instruments). The Orcomposutor then checks if the TAP process has closed successfully and, if not, checks for the presence of remaining processes. If this is the case, the Orcomposutor will automatically close them, which, in turn, releases the communication channel, allowing the campaign execution to continue.

Campaign Execution Retries

Due to the complexity of the systems which compose the testbed, it is possible for a campaign execution to fail in rare occasions. In order to avoid re-executing the tasks that finished successfully during these executions, the testbed allows to retry only failed tasks, sparing the rest of the tasks of the campaign. This feature is particularly useful in the case of certification campaigns, which can last for several days under normal circumstances, and would otherwise have to be repeated entirely in case of an error in the last task.



Status	Progress	Start time	End time	
Success	🟩	2018-11-13 08:32:36 UTC	2018-11-13 08:40:29 UTC	Details
Error Retry	🟥	2018-11-13 08:31:52 UTC	2018-11-13 08:32:28 UTC	Details
Error Retry	🟥	2018-11-13 08:20:25 UTC	2018-11-13 08:28:12 UTC	Details

Figure 40 Campaign retries in the TRIANGLE portal

The execution retry is available for the users of the TRIANGLE portal as a button that is available for errored campaigns. If the user choses to retry an existing execution, the Orcomposutor will check which of the required campaign tasks where completed successfully. These tasks will be removed from the internal list of tasks, scheduling only the execution of failed and pending tasks.

Due to the process followed in the ETL framework the fact that a campaign has been retried has no effect on the final TRIANGLE mark calculated for the campaign.

4.7 Execution Manager

In order to improve the utilization of the TRIANGLE testbed, it is possible to queue several campaign executions at a time. These campaigns will be executed successively by the testbed, avoiding idle periods of time while waiting for a user to start the execution of another campaign.

This has been achieved by means of a queue management layer between the TRIANGLE portal and the Orcomposutor.

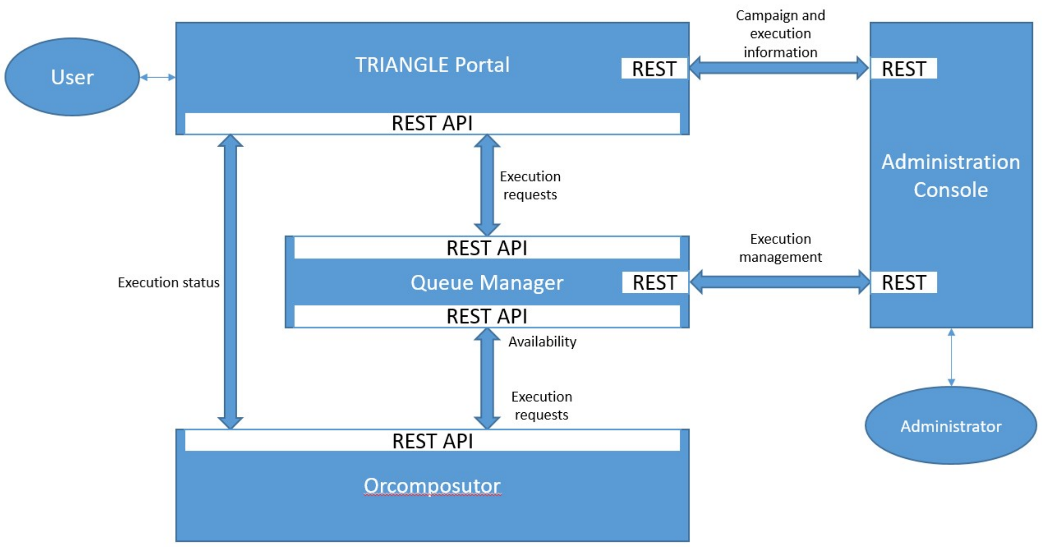
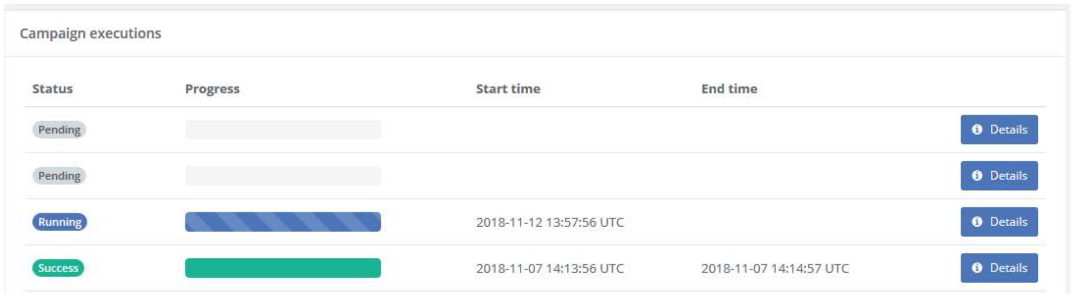


Figure 41 Testbed scheduling and management architecture

The Portal sends execution requests to the queue manager, which is acknowledged by creating a new campaign execution, which is set to the 'Pending' status. The queue manager then checks the availability of the testbed: If nothing is running the new campaign is automatically started, but if the testbed is not available then the campaign is kept in the queue until the Orcomposutor notifies that the current execution has been finished.


Campaign executions

Status	Progress	Start time	End time	
Pending				Details
Pending				Details
Running		2018-11-12 13:57:56 UTC		Details
Success		2018-11-07 14:13:56 UTC	2018-11-07 14:14:57 UTC	Details

Figure 42 Pending campaigns in the TRIANGLE portal

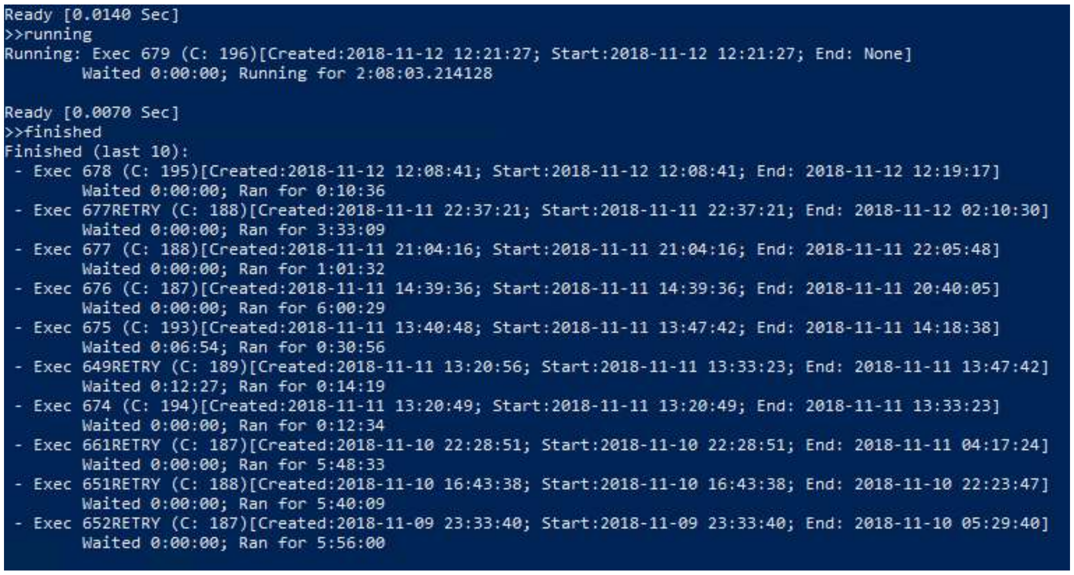
The communication between the components is performed via several REST APIs. In order to reduce complexity and execution delays, the Orcomposutor is still able to communicate directly with the Portal once a certain campaign execution starts: Only messages related to the scheduling of executions are passed through the queue manager.

4.7.1 Administration Console

The inclusion of this scheduling layer between the TRIANGLE Portal and the Orcomposutor allows to create a new administration interface for testbed management. The Administration Console communicates with the Portal and the queue manager using their REST APIs, and is able to modify the contents of the execution queue, while displaying useful information about current or past executions and campaigns.

The Administration Console is currently able to:

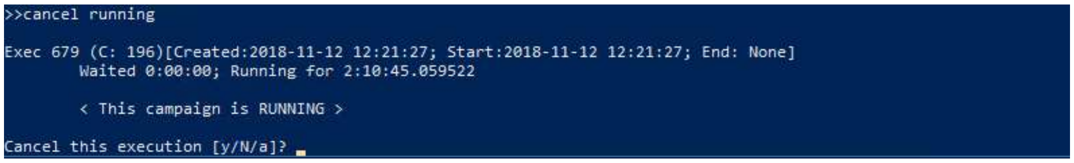
- Display information, such as creation, execution and wait times of queued, running or finished campaigns.



```
Ready [0.0140 Sec]
>>running
Running: Exec 679 (C: 196)[Created:2018-11-12 12:21:27; Start:2018-11-12 12:21:27; End: None]
        Waited 0:00:00; Running for 2:08:03.214128

Ready [0.0070 Sec]
>>finished
Finished (last 10):
 - Exec 678 (C: 195)[Created:2018-11-12 12:08:41; Start:2018-11-12 12:08:41; End: 2018-11-12 12:19:17]
        Waited 0:00:00; Ran for 0:10:36
 - Exec 677RETRY (C: 188)[Created:2018-11-11 22:37:21; Start:2018-11-11 22:37:21; End: 2018-11-12 02:10:30]
        Waited 0:00:00; Ran for 3:33:09
 - Exec 677 (C: 188)[Created:2018-11-11 21:04:16; Start:2018-11-11 21:04:16; End: 2018-11-11 22:05:48]
        Waited 0:00:00; Ran for 1:01:32
 - Exec 676 (C: 187)[Created:2018-11-11 14:39:36; Start:2018-11-11 14:39:36; End: 2018-11-11 20:40:05]
        Waited 0:00:00; Ran for 6:00:29
 - Exec 675 (C: 193)[Created:2018-11-11 13:40:48; Start:2018-11-11 13:47:42; End: 2018-11-11 14:18:38]
        Waited 0:06:54; Ran for 0:30:56
 - Exec 649RETRY (C: 189)[Created:2018-11-11 13:20:56; Start:2018-11-11 13:33:23; End: 2018-11-11 13:47:42]
        Waited 0:12:27; Ran for 0:14:19
 - Exec 674 (C: 194)[Created:2018-11-11 13:20:49; Start:2018-11-11 13:20:49; End: 2018-11-11 13:33:23]
        Waited 0:00:00; Ran for 0:12:34
 - Exec 661RETRY (C: 187)[Created:2018-11-10 22:28:51; Start:2018-11-10 22:28:51; End: 2018-11-11 04:17:24]
        Waited 0:00:00; Ran for 5:48:33
 - Exec 651RETRY (C: 188)[Created:2018-11-10 16:43:38; Start:2018-11-10 16:43:38; End: 2018-11-10 22:23:47]
        Waited 0:00:00; Ran for 5:40:09
 - Exec 652RETRY (C: 187)[Created:2018-11-09 23:33:40; Start:2018-11-09 23:33:40; End: 2018-11-10 05:29:40]
        Waited 0:00:00; Ran for 5:56:00
```

Figure 43 Campaign execution information

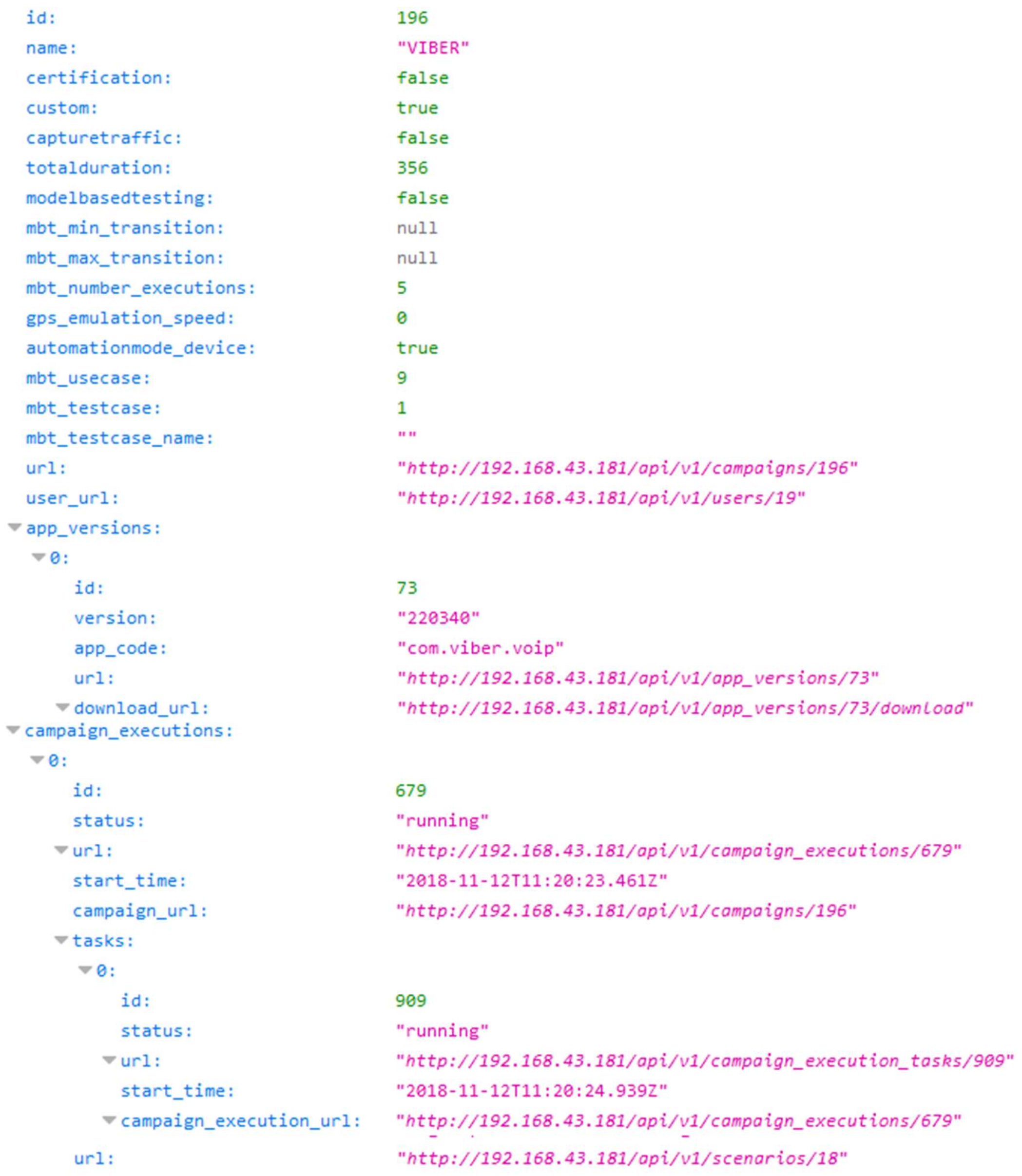
- Cancel running or queued campaign executions. The Administration console will remove canceled executions from the queue or stop them if running. The status of these executions will be set to 'Error' in the TRIANGLE Portal.



```
>>cancel running

Exec 679 (C: 196)[Created:2018-11-12 12:21:27; Start:2018-11-12 12:21:27; End: None]
       Waited 0:00:00; Running for 2:10:45.059522

       < This campaign is RUNNING >

Cancel this execution [y/N/a]? _
```

Figure 44 Cancelling a running campaign

- Display information about specific campaigns or campaign executions. This is currently done by displaying the information saved in the TRIANGLE Portal.



```
id:                              196
name:                            "VIBER"
certification:                   false
custom:                          true
capturetraffic:                  false
totalduration:                   356
modelbasedtesting:               false
mbt_min_transition:              null
mbt_max_transition:              null
mbt_number_executions:           5
gps_emulation_speed:             0
automationmode_device:           true
mbt_usecase:                     9
mbt_testcase:                    1
mbt_testcase_name:               ""
url:                             "http://192.168.43.181/api/v1/campaigns/196"
user_url:                        "http://192.168.43.181/api/v1/users/19"
app_versions:
  0:
    id:                          73
    version:                     "220340"
    app_code:                    "com.viber.voip"
    url:                         "http://192.168.43.181/api/v1/app_versions/73"
    download_url:                "http://192.168.43.181/api/v1/app_versions/73/download"
campaign_executions:
  0:
    id:                          679
    status:                      "running"
    url:                         "http://192.168.43.181/api/v1/campaign_executions/679"
    start_time:                  "2018-11-12T11:20:23.461Z"
    campaign_url:                "http://192.168.43.181/api/v1/campaigns/196"
    tasks:
      0:
        id:                      909
        status:                  "running"
        url:                     "http://192.168.43.181/api/v1/campaign_execution_tasks/909"
        start_time:              "2018-11-12T11:20:24.939Z"
        campaign_execution_url:  "http://192.168.43.181/api/v1/campaign_executions/679"
    url:                         "http://192.168.43.181/api/v1/scenarios/18"
```

Figure 45 Campaign information

5 Measurements and Data Collection

The testbed has integrated two tools, TestelDroid and DEKRA Performance Test Tool, which enable the instrumentation of mobile devices to collect data traffic information and OS API information. The testbed also includes a N6705B power analyzer to collect energy measurements from UEs. To collect measurements and information from apps which cannot be accessed in any other way, an instrumentation library will be provided to app developers. Finally, all the collected measurements will be sent to a central OML server.

5.1 TestelDroid Monitoring Tool

TestelDroid is an Android app envisaged to take advantage of the engineering features provided by current commercial smartphones for the development of advanced monitoring tools for mobile devices.

The features offered by TestelDroid are the following:

- Network information: Current operator, RAT (Radio Access Technology), cell identity, LAC (Location Area Code), RSSI (Radio Signal Strength Indicator), PSC (Primary Scrambling Code).
- Neighboring cell information: PSC, RSSI and type of network (not available for Samsung based phones, such as Samsung Galaxy S or Nexus S).
- GPS information: Longitude, latitude, altitude and speed.
- Traffic: Network traffic (monitoring mode displays only some information of the packet, such as protocol, IP source/destination or ports involved), using tcpdump. TestelDroid provides pcap format files containing the traffic captured.

  Besides monitoring and logging, TestelDroid allows:
- Connectivity tests: In order to diagnose connectivity issues
  - Ping a host (ping options are configurable)
  - Test if a port is open on a specified host
- Traffic test: Server-Client model, allows the transfer of an auto-generated file (size can be specified) between two devices. Speed is monitored on the server side and an average size is provided upon completion of the file transfer

To enhance its integration in the TRIANGLE testbed, a number of extensions have been added to TestelDroid. The extensions include support for Standard Commands for Programmable Instruments (SCPI), cOntrol and Management Framework (OMF) and OMF Measurement Library (OML). SCPI is the most widespread interface for measurement equipment control in many areas. OMF and OML extensions enable powerful orchestration framework languages that reduce the time required to define tests.

Table 11 TestelDroid configuration and control API

Command	**Purpose**	**Q**
**RST*	Reset the System	N
**IDN?*	Return a string with the name and version of the application	Y
**STB?*	Return a byte with the status, e.g.,: 0x00: idle, 0x01: running, 0xFF: error	Y

SETup:NETwork:RESTart	Restart the network (e.g.: using flight mode, searching network, etc.)	N
SETup:MEASurement:START	Start a measurement session (which includes starting the service under test)	Y
SETup:MEASurement:STOP	Stop a measurement session (which includes stopping the service under test)	Y
SETup:MEASurement:CONF: NETwork:ENable	Enable/Disable the NETWORK measurement (default on)	Y
SETup:MEASurement:CONF:TRAFfic: ENable	Enable/Disable the TRAFFIC measurement (default on)	Y
SETup:MEASurement:CONF:GPS:ENable	Enable/Disable the GPS reading (default off)	Y
SETup:MEASurement:CONF: NEIGHbour:ENable	Enable/Disable the NEIGHBOUR measurement (default off)	Y
SETup:MEASurement:CONF:PROFile:ENable	Enable/Disable the PROFILE measurement (default on)	Y
SETup:MEASurement:CONF:PROFile: SCENario	Add information about the context in which measurements are collected: vehicular, static, pedestrian or high-speed. This information is provided by the user of the tool. This information is very useful during the analysis and interpretation of the information collected by TestelDroid.	Y
SETup:MEASurement:CONF:PROFile:TECH	Define the network access technology, can be Mobile (generic and default), GSM, HSPA, LTE, UMTS,or Wi-Fi	Y
SETup:MEASurement:CONF:PROFile: CONFiguration	Add information about the context in which measurements are collected: Fixed-Fixed, Mobile-Mobile, Mobile-Fixed. This information is provided by the user of the tool. This information is very useful during the analysis and interpretation of the information collected by TestelDroid	Y
SETup:MEASurement:CONF:PROFile:SUBID	String identficating the SubId that will be used to generate the capture id, which has the following format YYYYMMDDHHSUBID	Y
SETup:MEASurement:CONF:PROFile:PEERID	String identifying the PeerId of the node, which can be 1 or 2, or NA to disable the use of PeerIds	Y
SETup:MEASurement:CONF:PROFile: COMMENTS	To add comments from the user. This information is provided by the user of the tool. This information is very useful during the analysis and interpretation of the information collected by TestelDroid	Y
RETrieve:MEAS:NETwork?	Returns the network file, the format might be:	Y

	INTEGER,ASCII_STRING,INTEGER,BINARY_STRING First integer is the length of the filename, the ascii_string is the filename, the next integer is the file size and the binary string is the content of the file. Network file format: <timestamp> <RAT> <Cell ID> <LAC> <RSSI> <MCC+MNC> <PSC>	
RETrieve:MEAS:GPS?	Returns the GPS file in the same format than the NETwork file. GPS file format: <timestamp> <latitude> <longitude> <speed>	Y
RETrieve:MEAS:BATTERY?	Returns the BATTERY file in the same format than the NETwork file. Battery file format: <timestamp> < µW > (available only when kernel has been modified)	Y
RETrieve:MEAS:NEIGHbour?	Returns the NEIGHBOUR file in the same format than the NETwork file. Neighbour file format: <timestamp> <PSC> <RSSI> <RAT>	Y
RETrieve:MEAS:PROFile?	Returns the PROFILE file in the same format than the NETwork file. Profile file format: <label>context_comment</label>	Y
RETrieve:MEAS:TRAFFIC?	Returns the TRAFFIC file in the same format that the NETWORK file Traffic forma file: pcap	Y

5.2 DEKRA Performance Tool

5G test scenarios will require high resolution for reporting target QoS KPIs. The TRIANGLE Testing Framework will provide up to layer 7 SDU packet resolution in the computation of data performance KPI thanks to the integration of the DEKRA Performance Tool.

This tool is composed of two components, Controller and Agents (data endpoints), and uses proprietary mechanisms to synchronize the Agents and provides accurate one-way measurements.

This tool includes a built-in traffic generator with the capability of generating constant rates, ramps, loops and statistical traffic patterns which is something of utmost importance for setting up the desired environment in terms of varying traffic loads (e.g., for measuring LTE-U impact on Wi-Fi networks).

Additionally, this tool has the ability to automate some mobile Apps on Android devices and measuring relevant QoE KPI such as YouTube buffering occurrences.

The summary of measurement capabilities provided to the TRIANGLE Testing Framework is summarized in Annex 6.

The DEKRA Performance Tool has been integrated into the TRIANGLE Testing Framework as TAP plugin (see section 5.2.135.2.10).

The DEKRA Performance Tool provides to the TRIANGLE Testing Framework the QoS and QoE measurements which are described in the following sections.

5.2.1 One-way Delay

This packet level measurement is an implementation of RFC 2679. This measurement is written in C for performance and portability across the range of supported mobile device platforms.

The measurement methodology proceeds as follows:

1. The system arranges that Source DEKRA-Agent and Destination DEKRA-Agent hosts are synchronized; that is, that they have clocks that are very closely synchronized with each other and each fairly close to the actual time.
2. At the Source DEKRA-Agent, the system selects Source and Destination IP addresses, and forms a test packet with these addresses. A test packet uses UDP transport protocol. The content of the test packet is random. The size of the packet (SDU) is parameter of the system and the packets inter departure time (i.e., time between consecutive packets) are parameters of the system.
3. At the Destination DEKRA-Agent, the system arranges to receive the packet.
4. At the Source DEKRA-Agent host, the system places a timestamp (t_1) in the prepared packet and sends it towards the Destination DEKRA-Agent host.
5. If the packet arrives within a reasonable period of time, the system takes a timestamp (t_2) as soon as possible upon the receipt of the packet. Note that the threshold of 'reasonable' is a parameter of the system.
6. Figure 46 shows in the protocol stack of the DEKRA-Agent hosts where the system measures the timestamps t_1 and t_2.

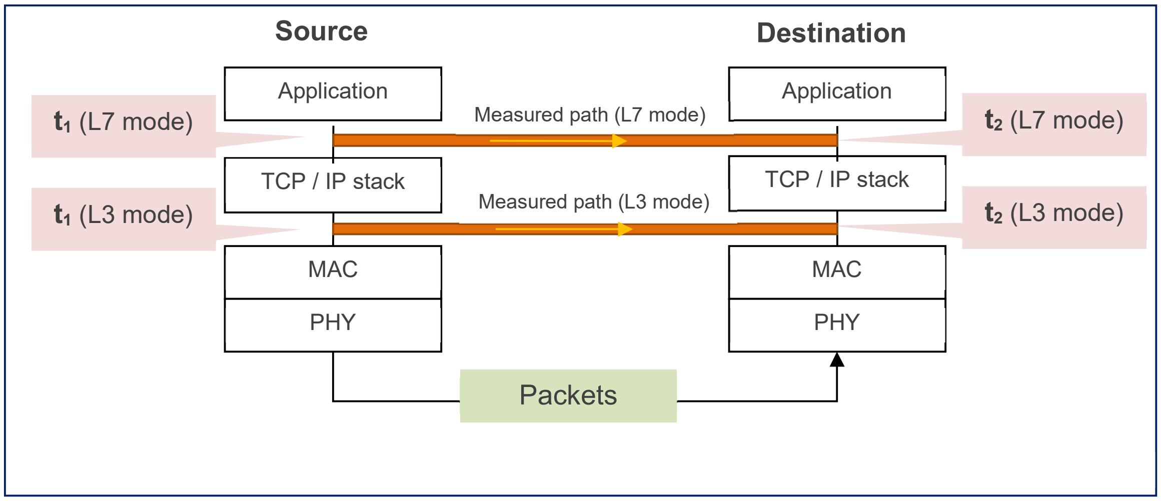


Figure 46 Timestamps for OWD measurements

7. By subtracting the two timestamps, an estimate of one-way delay can be computed for a give packet 'i':

$$\mathrm{OWDi} = \mathrm{t}_2 - \mathrm{t}_1 \qquad (1)$$

8. If the packet fails to arrive within a reasonable period of time, the one-way delay is taken to be undefined (informally, infinite).
9. The system groups the OWD_i samples into fixed periods of time (dT). The period "dT" is a parameter of the system. Default value is 1 second.
10. Given OWD [T, dT] = {OWD_1, OWD_2, …, OWD_N} the set of OWD_i samples within the partial interval [T, dT], the system computes the following instantaneous (i.e., per interval) statistics:

$$\text{Average OWD } [\mathrm{T}, \mathrm{dT}] = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=\mathrm{dT}}^{\mathrm{T}+\mathrm{dT}} \mathrm{OWD}_{\mathrm{i}} \qquad (2)$$

$$\text{Minimum OWD } [\mathrm{T}, \mathrm{dT}] = \text{min of } \{\text{OWD } [\mathrm{T}, \mathrm{dT}]\} \qquad (3)$$

$$\text{Xth Percentile OWD } [\mathrm{T}, \mathrm{dT}] = \mathrm{PCTL}_{\mathrm{X}}\{\text{OWD } [\mathrm{T}, \mathrm{dT}]\} \qquad (4)$$

11. Given OWD [t_{start}, t_{end}] = {OWD_1, OWD_2, … OWD_M} the set of OWD_i samples within the total measurement interval (t_{start}, t_{end}), the system computes the following statistics:

$$\text{Xth Percentile OWD } = \mathrm{PCTL}_{\mathrm{X}}\{\text{OWD } [\mathrm{t}_{\mathrm{start}}, \mathrm{t}_{\mathrm{end}}]\} \quad (5)$$

$$\text{PDF OWD } = \text{HISTOGRAM } \{\text{OWD } [\mathrm{t}_{\mathrm{start}}, \mathrm{t}_{\mathrm{end}}]\} \qquad (6)^{1}$$

$$\text{Average OWD } = \frac{1}{\mathrm{M}} \sum_{\mathrm{i}=\mathrm{tstart}}^{\mathrm{tend}} \mathrm{OWD}_{\mathrm{i}} \qquad (7)$$

Table 12 summarizes the One-way Delay KPIs reported by the TRIANGLE Testing Framework release 1.

Table 12 One-way Delay KPIs

KPI	**X-Axis**		**Y-Axis**	**Data Source**
OWD Average (t)	time	dT (ms)	Average OWD [T, dT]	(2)
OWD Minimum (t)	time	dT (ms)	Minimum OWD [T, dT].	(3)
OWD Median (t)	time	dT (ms)	50th Percentile OWD [d, dT]	(4)
OWD Percentile (t)	time	dT (ms)	0th Percentile OWD [d, dT] 5th Percentile OWD [d, dT] … 100 th Percentile OWD [d, T]	(4)
OWD CDF	%	*1%*	Xth Percentile OWD	(5)

[1] One OWD PDF plot is calculated for the OWD samples within the total measurement. For example, given a test 60 s long, where 10000 packets are correctly received, then the system obtains the following vector: OWD [0, 60] = {OWD_1, …., OWD_{10000}} and one PDF histogram is calculated for that vector of 10000 samples.

OWD PDF	frequency	Histogram OWD	(6)

5.2.2 One-way Delay Variation (Jitter)

This packet level measurement is an implementation of RFC 3393. This measurement is written in C for performance and portability across the range of supported platforms.

The methodology proceeds as follows:

1. Repeat steps 1 to 7 from the One-way delay measurement methodology to obtain the vector of samples OWD.
2. The system computes the IPDV samples as follows:

$$\mathrm{IPDV_i = OWD_i - \ OWD_{i-1}} \quad (1)$$

   $IPDV_i$ is undefined if either OWD_i or OWD_{i-1} is undefined (packet loss).

3. The system groups the $IPDV_i$ samples into fixed periods of time (dT). The period "dT" is a parameter of the system. Default value is 1 second.
4. Given IPDV [T, dT] = {$IPDV_1$, $IPDV_2$, …, $IPDV_N$} the set of $IPDV_i$ samples within the partial interval [T, dT], the system computes the following instantaneous (i.e., per interval) statistics:

$$\mathrm{Average\ IPDV\ [T, dT]} = \frac{1}{N}\sum_{i=dT}^{T+dT} \mathrm{IPDV_i} \quad (2)$$

$$\mathrm{Peak\ to\ Peak\ IPDV\ [T, dT] = max\ of\ \{OWD\ [T, dT]\} - \ min\ of\ \{OWD\ [T, dT]\}} \ (3)$$

$$\mathrm{Xth\ \ Percentile\ IPDV\ [T, dT] = PCTL_X\{IPDV\ [T, dT]\}} \quad (4)$$

5. Given IPDV [t_{start}, t_{end}] = {$IPDV_1$, $IPDV_2$, … $IPDV_M$} the set of $IPDV_i$ samples within the total measurement interval (t_{start}, t_{end}), the system computes the following statistics:

$$\mathrm{Xth\ \ Percentile\ IPDV\ \ = PCTL_X\{IPDV\ [t_{start}, t_{end}]\}} \quad (5)$$

$$\mathrm{Average\ IPDV\ \ =} \frac{1}{M}\sum_{i=tstart}^{tend} \mathrm{IPDV_i} \quad (6)$$

Table 13 summarizes the One-way Delay Variation KPIs reported by the TRIANGLE Testing Framework release 1.

Table 13 One-way Delay Variation KPIs

KPI	**X-Axis**		**Y-Axis**	**Data Source**
IPDV Average (t)	time	dT (ms)	Average IPDV [T, dT]	(2)
IPDV Peak to Peak (t)	time	dT (ms)	Peak to Peak IPDV [T, dT].	(3)
IPDV Percentile (t)	time	dT (ms)	0th Percentile IPDV [d, dT] 5th Percentile IPDV [d, dT] …	(4)

			100 th Percentile IPDV [d, T]	
IPDV CDF	%	*5%*	Xth Percentile IPDV	(4)
IPDV PDF	*frequency*		Histogram IPDV	(5)

5.2.3 One-way Packet Loss Rate

This packet level measurement is an implementation of RFC 2680. This measurement is written in C for performance and portability across the range of supported platforms.

The methodology proceeds as follows:

1. Repeat steps 1 to 4 from the One-way delay measurement.
2. If a packet 'i' arrives within a reasonable period of time, the system counts $PL_i = 0$. Otherwise, if a packet 'i' does not arrive within a reasonable period of time, the system counts $PL_i = 1$. Note that the threshold of 'reasonable' is a parameter of the system.
3. The system groups the PL_i samples into fixed periods of time (dT). The period "dT" is a parameter of the system. Default value is 1 second.
4. Given PL [T, dT] = {PL_1, PL_2, …, PL_N} the set of PL_i samples within the partial interval [T, dT], the system computes the following instantaneous (i.e., per interval) statistics:

$$\text{PL Rate}\,[\text{T}, \text{dT}] = \frac{1}{\text{N}}\sum_{\text{i}=\text{dT}}^{\text{T}+\text{dT}} \text{PL}_\text{i} \times 100 \qquad (1)$$

5. Given PL [t_{start}, t_{end}] = {PL_1, PL_2, … PL_M} the set of PL_i samples within the total measurement interval (t_{start}, t_{end}), the system computes the following statistics:

$$\text{PL Rate} = \frac{1}{\text{M}}\sum_{\text{i}=\text{tstart}}^{\text{tend}} \text{PL}_\text{i} \times 100 \qquad (2)$$

Table 14 summarizes the One-way Packet Loss Rate KPIs reported by the TRIANGLE Testing Framework release 1.

Table 14 One-way Packet Loss Rate KPIs

KPI	X-Axis		Y-Axis	Data Source
PL Rate (t)	time	dT (ms)	Average PL [T, dT]	(1)

5.2.4 One-way Packet Loss Distribution

This packet level measurement is an implementation of RFC 3357. This measurement is written in C for performance and portability across the range of supported platforms.

The methodology proceeds as follows:

1. Repeat steps 1 to 3 from the One-way packet loss measurement to obtain the PL vector.

2. Given *PL* [t_{start}, t_{end}] = {PL_1, PL_2, … PL_M} the set of PL_i samples within the total measurement interval (t_{start}, t_{end}), the system computes the following:
   a. Loss Distance LD_i: When a packet is considered lost (PL_j= 1), the system looks at its sequence and compares it with that of the previous lost packet (PL_k =1). The difference j-k is the *loss distance* between the lost packet and the previous lost packet,
   b. Loss Period LP_i: A loss period begins if PL_j = 1 and PL_{j-1} = 0.
   c. Noticeable Loss: A packet loss is "noticeable" if LD_i is no greater than delta, a positive integer, where delta is the "loss constraint".
3. Given PL, LD and LP vectors, the system compute the following:

$$\text{Noticeable Rate } (\%) = \frac{\text{Number of noticeable losses}}{\text{Total number of losses}} \text{ for a given "delta"} \quad (1)$$

$$\text{Loss Period Lengths} = \text{Number of packet lost in each loss period} \quad (2)$$

$$\text{Inter} - \text{loss Period Lengths} = \text{Distance between consecutive periods loss} \quad (3)$$

$$\text{Period Loss Duration} = \text{Duration of each loss period} \quad (4)$$

Table 15 summarizes the One-way Packet Loss Distribution KPIs reported by the TRIANGLE Testing Framework release 1.

Table 15 One-way Packet Loss Distribution KPIs

KPI	**X-Axis**		**Y-Axis**	**Data Source**
One-way Loss Noticeable Rate	Delta		Noticeable Rate (%)	(1)
One-way Consecutive Lost Packets	Length		Periods Loss Frequency	(2)
One-way Consecutive Lost Packets Distance	Length		Inter Periods Loss Frequency	(3)
One-way Consecutive Lost Packets Time (s)	time	dT (ms)	Period Loss duration (ms)	(4)

5.2.5 YouTube™

The DEKRA-Agent forces the mobile device to visualize a video from YouTube™ and measures the quality of experience of the video streaming.

The implementation of this measurement is driven by the official YouTube API which embeds a player on the DEKRA-Agent that most closely behaves like the YouTube native application[2].

[2] At the time this document has been revised.

A YouTube session consists of a number of individual YouTube playbacks.

1. DEKRA-Agent loads a full screen Web View component.
2. DEKRA-Agent gets a locally stored HTML test page. This HTML page contains Javascript code using YouTube iFrame API. It implements all required features for the measurement:
    a. It loads a YouTube player.
    b. It sets the size of the player to the size of the testing device screen which is equivalent to a landscape full screen playback.
    c. It plays the video clip (e.g., auto play).
    d. It captures the user events required to calculate the KPIs (quality changes and player states).
    e. DEKRA-Agent plays the video in the embedded player in the loaded web view.

Table 16 summarizes the KPIs reported by the TRIANGLE Testing Framework release 1 for the reference App YouTube™.

Table 16 YouTube™ KPIs

KPI	**Units**	**Description**
MOS	-	Estimation of the video quality as perceived by a user. Possible values: 1 (bad) to 5 (good). This estimation is based on the initial buffering, and the re-buffering index.
Playback Time	s	This is the actual duration of the playback from loading the player until the video clip ends. Playback Duration = Initial Buffering Time + Total Re-buffering Time + Other impairments Other impairments: - Video lagging due to testing device CPU over load. - iFrame API accuracy in rebufferings: Elapsed time from playback pauses until API event calls `onPlayerStateChange == PlayerState.BUFFERING`
Playback Size	MB	The amount of data downloaded by the device to play the video regardless of the bit rate used. Therefore, the same video clip may report different video sizes in different test iterations if the bit rate used has been also different.
Initial buffering	s	The period between the starting time of loading a video and the starting time of playing it.
Re-bufferings	-	When the buffered video data decreases to a low value, the playback will pause, and the player will enter into a re-buffering state. This KPI shows how many times this event happens.

		The following KPIs are derived from all the re-bufferings in a video playback: - Maximum re-buffering time (s) - Average re-buffering time (s) - Total re-buffering time (s) - Re-buffering Index Re-buffering duration is calculated on by capturing the Jasvascript events `onPlayerStateChange == PlayerState.BUFFERING` and `onPlayerStateChange == PlayerState.PLAYING.` The Re-buffering Index is computed from the number of re-buferings, the re-bufferings duration and the video's length. Possible values: 0 (good) to infinite (bad).
Video Quality Distribution	-	The playback quality of the video. The following KPIs are derived from a video playback: - First / Last Video Quality - % of Time in each Quality (histogram) - Most Used Video Quality - Average Video Quality[3]
Playtime	s	Over time representation of the video playback. X-Axis: Actual Time Y-Axis: Playback time

5.2.6 Spotify™

The DEKRA-Agent forces the mobile device to reproduce an audio track from Spotify™ and measures the quality of experience of the audio streaming.

1. DEKRA-Agent uses Spotify API to start a music track
2. DEKRA-Agent sets the playback rate with this API function:

   ```
   public voidsetPlaybackBitrate
   ```

   Spotify API: "*Set the bitrate of the player to specified value. This will take effect for the next chunk of audio that is streamed from the backend. The format or sample rate of the audio data does not change*"
3. Whenever the player state changes, the DEKRA-Agent captures the event calls: PLAY, TRACK END, etc.
4. Then, while the music track is playing on the device, the DEKRA-Agent captures the following event calls to measure the KPIs:

   ```
   int onAudioDataDelivered
   ```

   Spotify API: *"Called whenever the player receives audio data. The method is synchronous and therefore blocking*"

[3] Average Video Quality = (% time in 120p * 120 + % time in 240p ... + % time in 4k * 4k) /100.

5. Based on the API event calls presented above, the DEKRA-Agent calculates the following KPIs:
    a. Initial Buffering (s): First `onAudioDataDelivered` event - PLAY
        i. If either of the events is not captured, the KPI is reported as null (absence of measurement).
    b. Number of Re-bufferings:
        i. Let T be the time between consecutive `onAudioDataDelivered` events. DEKRA-Agent counts a re-buffering observation whenever T is longer than 200 ms. Thus, Re-buffering Time is given by T – 40 ms.
    c. Total Re-buffering Time (s): Sum of (Re-buffering Time observations)
    d. Playback Duration (s): TRACK END – PLAY.
    e. Re-buffering Index: Total Rebuffering Time / (TRACK END - First `onAudioDataDelivered` event).

Table 17 summarizes the KPIs reported by the TRIANGLE Testing Framework release 1 for the reference App Spotify™.

Table 17 Spotify™ KPIs

KPI	**Units**	**Description**
Initial buffering	s	The period between the starting time of loading the audio track and the starting time of playing it.
Re-bufferings	-	When the buffered audio data decreases to a low value, the playback will pause, and the player will enter into a re-buffering state. This KPI shows how many times this event happens. The following KPIs are derived from all the re-bufferings in a audio playback: Maximum re-buffering time (s) Average re-buffering time (s) Total re-buffering time (s)
Track Size	MB	The amount of data downloaded by the device to play the audio regardless of the bit rate used.
Re-buffering index	-	This KPI is computed from the number of re-buferings, the re-bufferings duration and the audio's length. Possible values: 0 (good) to infinite (bad).
MOS	-	Estimation of the audio quality as perceived by a user. Possible values: 1 (bad) to 5 (good). This estimation is based on the initial buffering, and the re-buffering index.
Playback Duration	s	This is the actual duration of the playback from loading the player until the audio track ends.

5.2.7 Facebook™

The DEKRA-Agent forces the mobile device to perform Facebook operations and measures the quality of experience.

This measurement uses the Android Facebook API and uses Facebook test accounts which must be created by the user only once before running the first test on that device.

Table 18 summarizes the KPIs reported by the TRIANGLE Testing Framework release 1 for the reference App Facebook™.

Table 18 Facebook™ KPIs

KPI	**Units**	**Description**
Time to post a comment	s	Elapsed time since users click "post a comment" and the comment is posted in their Facebook account.
Time to post a image	s	Elapsed time since users click "post an image" and the comment is posted in their Facebook account.
Time to post a video	s	Elapsed time since users click "post a video" and the comment is posted in their Facebook account.
Operations successful rate	%	Percentage of operations successfully completed.

5.2.8 Web Browsing

The DEKRA-Agent forces the mobile device to download a web page from a destination web server (target IP server) for each test measurement.

A Web browsing session consists of a number of individual web page downloads. The following figure shows a Web Browsing iteration identifying the Setup and Session Time.

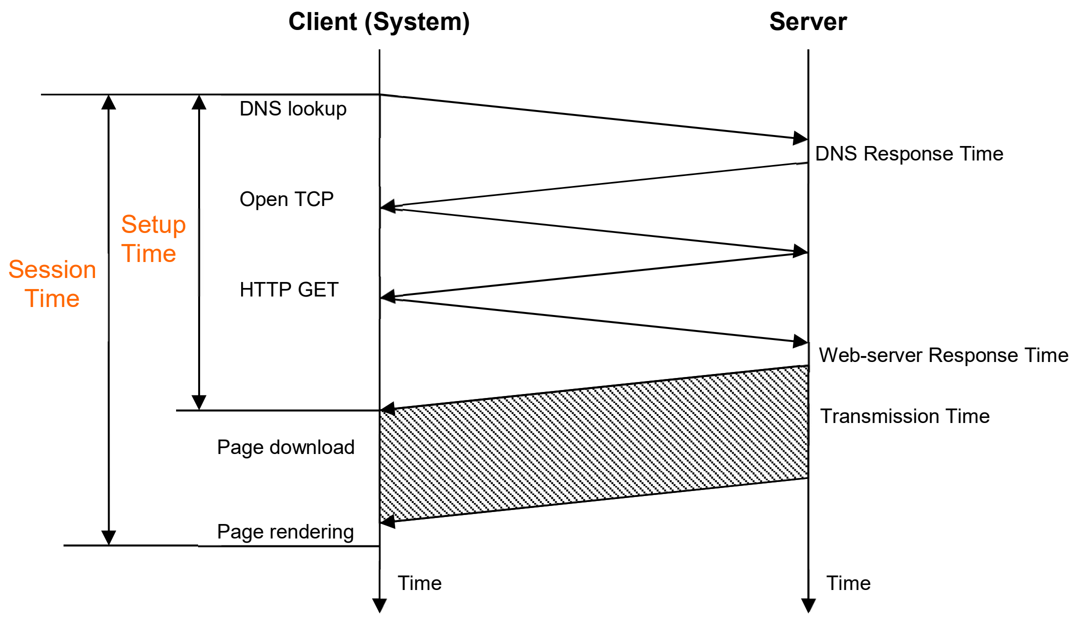


Figure 47 Web Browsing measurements

Table 19 summarizes the KPIs reported by the TRIANGLE Testing Framework release 1 for the reference App Web Browser.

Table 19 Web Browsing KPIs

KPI	**Units**	**Description**
Setup time	s	Time period needed to access the web page, from user entering the URL and hitting “Return”, to the point of time to receive the first byte of the web page
Session time	s	Time period needed to successfully complete the data transfer, from user entering the URL and hitting “Return”, to the point of time to receive the last byte of the web page
Mean data rate	Mbit/s	The average data transfer rate measured throughout the entire connect time to the service. The data transfer shall be successfully terminated

5.2.9 File Transfer

The DEKRA-Agent forces the mobile device to transfer (download or upload) a data file from a destination file storage server (target IP server) for each test measurement.

In both the download and upload modes the test uses a single TCP connection to perform the transfer.

Table 20 summarizes the KPIs reported by the TRIANGLE Testing Framework release 1 for the reference App File Transfer.

Table 20 File Transfer KPIs

KPI	Units	Description
Setup time	s	Time period needed to access the data service, from user entering the URL and hitting "Return", to the point of time to receive the first byte of the file
Session time	s	Time period needed to successfully complete the data transfer, from the point of time to receive the first byte of the data file, to the point of time to receive the last byte
Mean data rate	Mbit/s	The average data transfer rate measured throughout the entire data file transfer. The data transfer shall be successfully terminated

5.2.10 WLAN Access Point Automation

In the scope of Task 3.6 Integration with the underlying infrastructure, DEKRA has developed a module to facilitate the integration of Wi-Fi access points in the TRIANGLE testbed.

Table 21 shows the list of commands available in the TRIANGLE testing framework.

Table 21 WLAN AP commands available in TRIANGLE

Name	Possible values
Set Channel Width	{20, 40, 80, 20_40, 20_40_80}
Set Channel Numer	Any valid channel number
Turn On/Off	Turn on/off Wi-Fi radio
Set Transmit Level	{0, 100} %

The module developed by DEKRA uses Telnet to control the WLAN Access Point. Then, it is suitable for automating any WLAN Access Point provided that it exposes a Telnet interface.

The implementation consists in a parser of a descriptor file which contains the list of Telnet commands in human readable language for a specific WLAN Access Point model. This way if another WLAN Access Point used different command set, the module would be still valid and only updating the descriptor file would be needed.

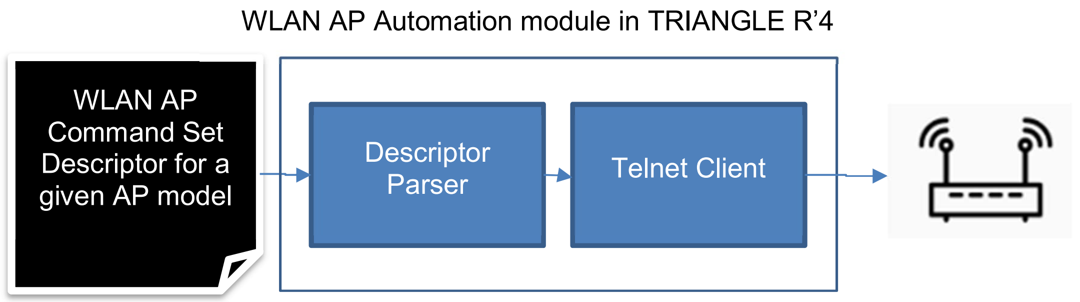

WLAN AP Automation module in TRIANGLE R'4
WLAN AP Command Set Descriptor for a given AP model
Descriptor Parser
Telnet Client

Figure 48 TRIANGLE WLAN AP Automation

The descriptor file for the WLAN Access Point used for the validation of this module is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<Commands>
        <WidthCommandParam>
                <20>0x1</20> <!--Channel width 20 MHz-->
                <20_40>0x3</20_40> <!--Channel width 20/40 MHz-->
                <20_40_80>0x7</20_40_80> <!--Channel width 20/40/80 MHz-->
        </WidthCommandParam>
        <SetChannel>
                <Description>Specifies a channel number</Description>
                <Op>admin</Op> <!--Your AP username-->
                <Op>12345</Op> <!--Your AP password-->
                <Op>wl -i eth1 radio off</Op>
                <Op>wl -i eth1 channel CHANNEL</Op>
                <Op>wl -i eth1 radio on</Op>
                <Op>exit</Op>
        </SetChannel>
        <TurnOn>
                <Description>Turn radio On</Description>
                <Op>admin</Op> <!--Your AP username-->
                <Op>12345</Op> <!--Your AP password-->
                <Op>wl -i eth1 radio on</Op>
                <Op>exit</Op>
        </TurnOn>
        <TurnOff>
                <Description>Turn radio Off</Description>
                <Op>admin</Op> <!--Your AP username-->
```

```
            <Op>12345</Op> <!--Your AP password-->
            <Op>wl -i eth1 radio off</Op>
            <Op>exit</Op>
       </TurnOff>
       <TransmitLevel>
            <Description>Specifies the transmit power level for the current
operating radio channel on the access point (%)</Description>
            <Op>admin</Op> <!--Your AP username-->
            <Op>12345</Op> <!--Your AP password-->
            <Op>wl -i eth1 pwr_percent POWER</Op>
            <Op>exit</Op>
       </TransmitLevel>
</Commands>
```

For the validation of the WLAN AP automation module, we have used the configuration of the TRIANGLE testbed which uses the LWIP feature. Table 22 summarizes the scenarios which has been conducted in the preparation of this deliverable. In all scenarios, there was an LTE link up and transmitting data throughout the entire test whereas the WLAN configuration was varied by using the implemented WLAN AP automation module. The imposed traffic load was 200 Mbit/s between the laptop client and a data endpoint on the EPC computer.

Table 22 WLAN Access Point automation and LWIP scenarios

Name	Test steps
Test 1	WLAN On, WLAN Off, WLAN ON
Test 2	WLAN TX Power: 1, 10, 25, 50, 100
Test 3	Channel 1, Channel 11

The test setup used for the validation of the WLAN AP automation feature with LWIP is shown in the Figure 49 .

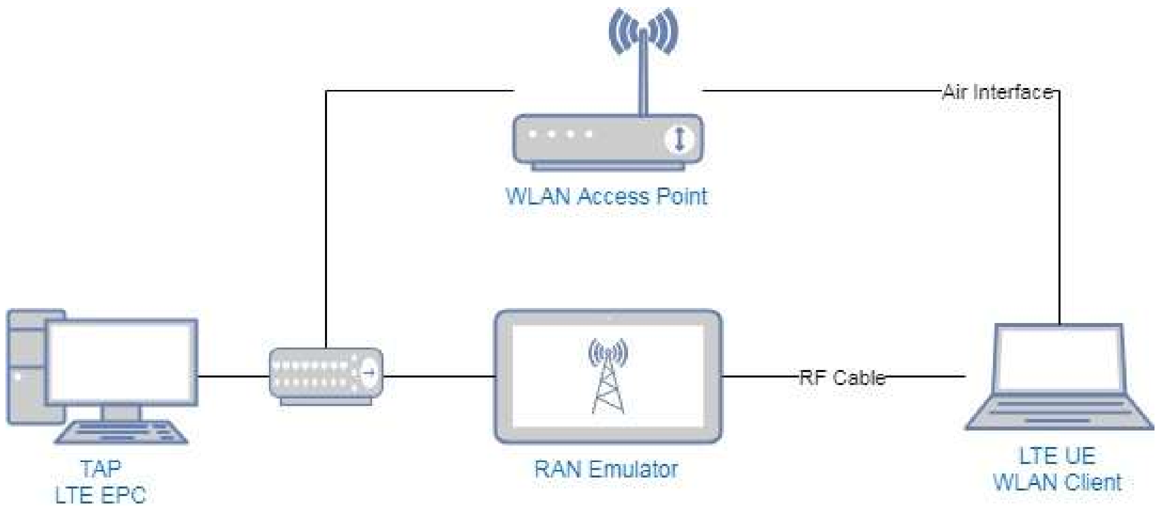


Figure 49 TRIANGLE configuration for WLAN AP automation feature validation

Annex 7 presents the more relevant results obtained during the validation of the WLAN AP automation feature on the LWIP based TRIANGLE testbed configuration.

5.2.11 Virtual Reality Application Testing Capability

The goal of VR applications is to emulate a natural and fluid interaction between the user and a virtual world, which will demand network resources. How far from natural and fluidity will determine the quality of experience perceived by VR users

The solution implemented supports testing VR apps on Android and iOS devices. Details available in section 11.2.

Annex 8 provides details of the implementation of the Remote-Control Interface service. That specification provides a complete view of the measurement capabilities available in Release 3 of the TRIANGLE testbed.

5.2.12 Other Features

In the scope of DRA test specification [D2.2_AP6], Content Distribution Streaming Reference App use case and more specifically Content Stall KPI, reporting the percentile curve (a.k.a CDF) as KPI summarization was specified for the TRIANGLE Mark scoring. The ability to collect the measurements for Content Stall KPI resides in the DEKRA Performance Tool component. In Release 2, even though the DEKRA Tool internally measured every Content Stall instance and its duration, only average and maximum values were exposed to the TRIANGLE testbed. In Release 3, all the Content Stall instances as measured by the DEKRA Tool are now exposed and made available to the testbed so that the ETL component can compute the required KPI summarization (i.e., percentile curve) which is necessary to provide support for the TRIANGLE mark scoring process.

In the scope of RES test specification [D2.2 AP5], reporting the device GPU usage was specified. In Release 3 the DEKRA Performance Tool is able to collect GPU Usage from Android devices.

That new measurement capability has been integrated into the TRIANGLE test bed thus completing the full coverage of the RES tests specification.

The device interface that the DEKRA Tool uses for reading the GPU load is "/sys/class/kgsl/kgsl-3d0/gpubusy", which reports the total and busy cycles DEKRA Tool uses to calculate the GPU usage: `/sys/class/kgsl/kgsl-3d0/gpubusy` reports two integers g0and g1. The actual load as a percentage can be calculated as (g0/g1*100).

5.2.13 DEKRA TAP plugin

This section describes the implementation of the TAP plugin for the integration of the DEKRA Performance Tool in the TRIANGLE testbed.

DEKRA Performance Tool Interface

The plugin implementation relies on the interface exposed by the tool, the Remote Control (RC) Sever. The TAP plugin is therefore an implementation of a RC Client.

The RC Server can be accessed with the following parameters:

Table 23 DEKRA Tool RC Server channel

Item	Setting
LAN IP Address	IP address of the Performance Tool
Protocol	TCP
Port	11500
End of Sentence	'\n' (line feed)

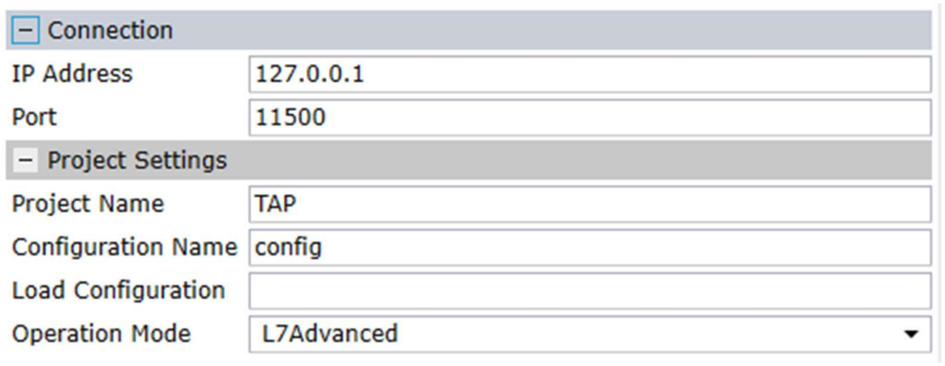
The RC Client (i.e., the TAP plugin) shall open a connection to that service and send commands. All commands imply a response from the RC Server that the RC User shall read right after sending the command.

Instrument

Table 24 shows the operations implemented for TAP Instrument management.

Table 24 DEKRA Tool TAP Instrument

Operation	Description
Open	Creates the socket with the DEKRA Tool RC Server
Close	Closes the socket with the DEKRA Tool RC Server
Settings	DEKRA Tool RC Server settings: IP address, port, operation mode, and project/session name
End of Sentence	'\n' (line feed)



Connection	
IP Address	127.0.0.1
Port	11500
Project Settings	
Project Name	TAP
Configuration Name	config
Load Configuration	
Operation Mode	L7Advanced

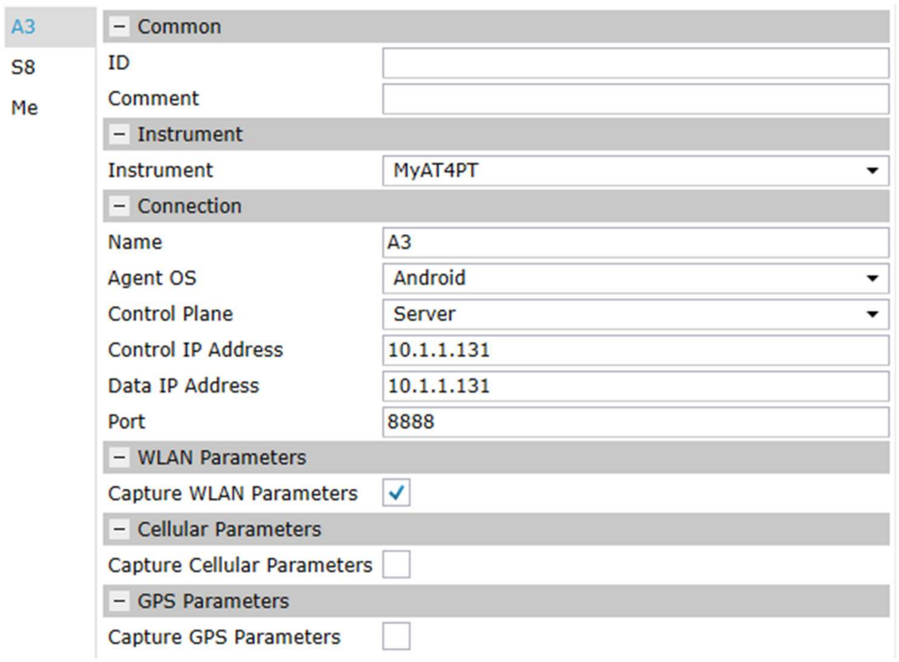
Figure 50 DEKRA Tool TAP Instrument

DUT (TACS4-Agent)

Table 25 shows the operations implemented for TAP DUT management.

Table 25 DEKRA Tool TAP DUT

Operation	Description
Definition	Defines the Agents which participate in the test. Adding a DUT does not necessarily implies its usage.
Configuration	Defines the configuration of the TACS4-Agent: IP address, Port, etc.
Open/Close	There is no method for open/close. TAP by default initiates the DUT before the Instrument.
Add Agent	This is the step which adds and configure a Agent to the configuration of the RC Server. The only parameter of this operation is the DUT itself. The step reads all in the information stored in the TAP DUT and builds the RC Server configuration commands.



A3
S8
Me

− Common	
ID	
Comment	
− Instrument	
Instrument	MyAT4PT
− Connection	
Name	A3
Agent OS	Android
Control Plane	Server
Control IP Address	10.1.1.131
Data IP Address	10.1.1.131
Port	8888
− WLAN Parameters	
Capture WLAN Parameters	✓
− Cellular Parameters	
Capture Cellular Parameters	
− GPS Parameters	
Capture GPS Parameters	

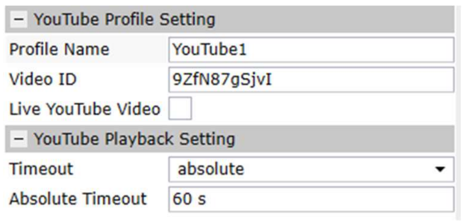
Figure 51 DEKRA Tool TAP DUT

Test Step

Table 26 shows the operations implemented for TAP Test Steps management.

Table 26 DEKRA Tool TAP Test Steps

Operation	Description
Configuration	The configuration of the test steps is similar in all the tests except for the input parameters. For example, below is the input parameters for YouTube; • Video Id • Timeout
Execution	The test steps have basically three methods and a constructor. The constructor initializes the input parameters with default values. The methods pre-plan and post-plan are used to do operation before and after start the test plan. These methods are not used in this plugin. The method "run" is used to execute the test step itself. This methods implements the configuration of the test based on the actual input parameters.



− YouTube Profile Setting	
Profile Name	YouTube1
Video ID	9ZfN87gSjvI
Live YouTube Video	
− YouTube Playback Setting	
Timeout	absolute
Absolute Timeout	60 s

Figure 52 DEKRA Tool TAP Test Step (YouTube test)

Run/Stop Test

Both Run and Stop Test steps have one single input parameter the TAP Instrument itself.

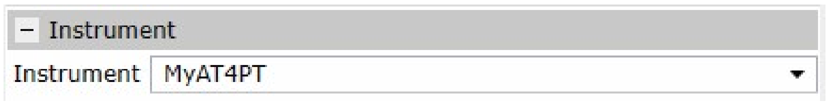


Figure 53 DEKRA Tool TAP Run/Stop Test

The step Test Run is blocking and block the TAP Tets Plan until the step finishes.

Get Results

The DEKRA TAP plugin reports the measurements with one second resolution. This feature is available because the RC server implements a function called "Get Vector".

Additionally, the DEKRA TAP Plugin also reports the system measurements collected from the test phone (e.g., CPU usage, battery status) with one second resolution as well.

Table 27 shows an extract of the RC server specification which has been used to implement the DEKRA TAP plugin get results procedures.

Table 27 DEKRA Tool RC Server Get Results

Operation	Description
Get <x> Vector	This function returns the list of pairs (timestamp, <x>) as measured throughout the test session, where <x> is the type of measurement: Thorughput, One way delay, pachet loss, or jitter. Example: RESULT:OWDGETVECTOR 2016-05-02 16h 20m 10s, MyUDPFLOW,averaged OK: 0.067,42.123,1.069,42.325,2.071,45.322,3.075,46.123
Get RASM Vector	This function returns the list of pairs (timestamp, phone parameter) as measured throughout the test session. Example (fro WLAN RSSI): RESULT:RASMGETVECTOR 2016-05-02 16h 20m 10s, Agent1, wlan.rssi OK: 0.067,-42,1.069,-42,2.071,-45,3.075,-46
Get YOUTUBE	Returns the YouTube KPIs from a specific Agent. Possible KPIs: Possible values:

- ib:Initial buffering in seconds
- rb_index: Re-buffering index (0-inf)
- rb_avg: Average re-buffering time (s)
- rb_max: Maximum re-buffering time (s)
- rb_total: Total re-buffering time (s)
- rb_number: Number of re-bufferings
- size: Playback size in MB
- MOS: MOS (1-5)
- duration: Playback duration in seconds
- thoughput: Average throughput in Mbit/s
- vq_first: First video quality
- vq_ last: Last video quality
- vq_mode: Most used video quality
- vq_avg: Average video quality
- vq_144: % time in 144p
- vq_240: % time in 240p
- vq_360: % time in 360p
- vq_480: % time in 480p
- vq_720: % time in 720p
- vq_1080: % time in 1080p
- vq_2k: % time in 1440p
- vq_4k: % time in 2160p

Example:

RESULT: YOUTUBEGETKPI 2016-05-02 16h 20m 10s, Agent1,MyYou,1,ib

OK: 1.123,2.123,,3.245

Error Handling

All the commands implemented in the DEKRA Tool RC Server handle the error cases and return error code and message. These codes and messages are transparently propagated up to the TAP GUI via the DEKRA TAP Plugin. The DEKRA TAP plugin does not implement any additional error handling. It just forwards the error coming up from the RC Server.

FAIL is reported whenever the error does not prevent the execution of the TAP test plan. ERROR is whenever the error prevents the execution of the TAP test plan (e.g., the DEKRA is unable to connect to the Agent running on the test phone).

5.3 Power Analyzer

The Keysight N6705B DC power analyzer 4-slot mainframe holds up to 600 W of total power. The N6705B is a highly integrated instrument that combines up to four advanced DC power supplies, digital multi-meter (DMM), oscilloscope, arbitrary waveform generator and data logger. It provides an interface, with all sourcing and measuring functions available from the front panel. In addition, the instrument can be remotely controlled. The aforementioned power analyzer is able to measure the current going into the UE. We extened the capabilities of the device to support IEEE 1588 for time synchronization between the eNodeB emulator and the power analyzer. This section provides a brief overview of the capabilities of the power analyzer. For a more exhaustive view of the commands the reader is referred to the user manual [47].

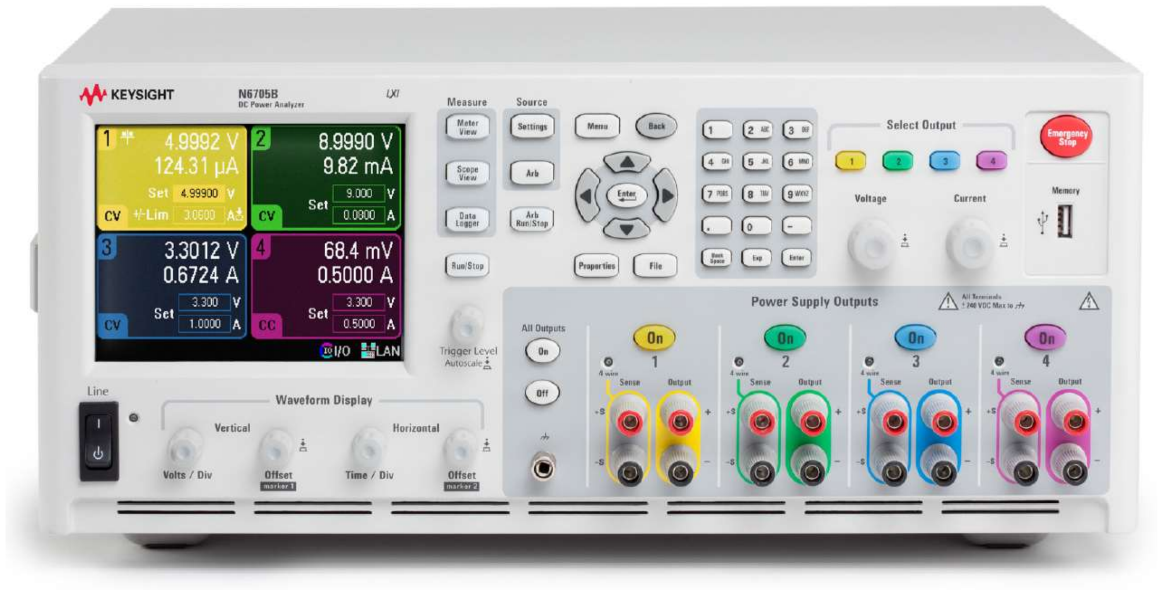
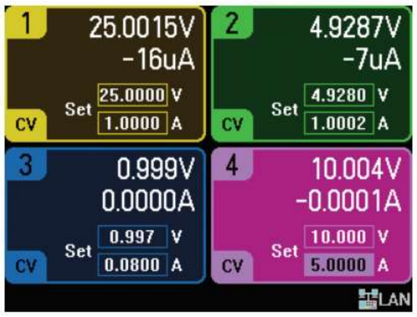


Figure 54 Keysight N6705B DC Power Analyzer

5.3.1 Voltmeter/Ammeter: Meter View

Each DC power module in the Keysight N6705 DC power analyzer has a fully integrated voltmeter and amperimeter to measure the actual voltage and current being sourced out of the DC output into the UE.



```
1  25.0015V          2  4.9287V
   -16uA                -7uA
   Set 25.0000 V        Set 4.9280 V
CV     1.0000 A     CV      1.0002 A
3  0.999V            4  10.004V
   0.0000A              -0.0001A
   Set 0.997 V          Set 10.000 V
CV     0.0800 A     CV      5.0000 A
                                 LAN
```

Figure 55 Meter View; all 4 outputs can be viewed simultaneously

5.3.2 Oscilloscope: Scope View

Each DC power module in the Keysight N6705 DC power analyzer has a fully integrated digitizer to capture the actual voltage-versus-time and current-versus-time being sourced from the DC output into the UEDUT. The digitized data appears on the large colour display just like an Oscilloscope.

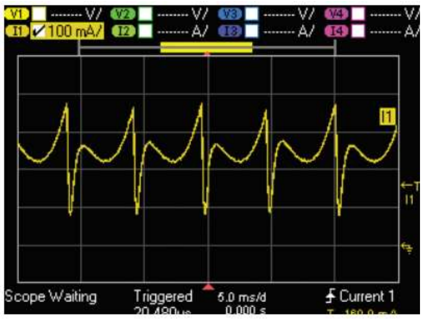


Figure 56 Scope View; voltage and current traces are displayed

5.3.3 Data Logger View

The Keysight N6705 DC power analyzer can also function as a data logger. Using the measurement capability built into each DC power module, the N6705 can continuously log data to the large colour display and to a file. Data can be simultaneously logged on all four DC outputs.

The following table summarizes the data logging specifications:

Table 28 N6705 DC power analyzer data logging specifications

	Standard data logging	Continuous data logging
Sample interval range	75 milliseconds to 60 seconds	20* µs to 60 s *Add 20 µs for each additional parameter (Voltage, Current, Min, or Max)
Sample rate	50 kHz	50 kHz

5.3.4 Control and Analysis Software

The software for the DC power analyzer complements the front panel of the N6705B mainframe, offering advanced functionality and PC control as shown in Figure 57. One showcase application is battery drain analysis for IoT devices.

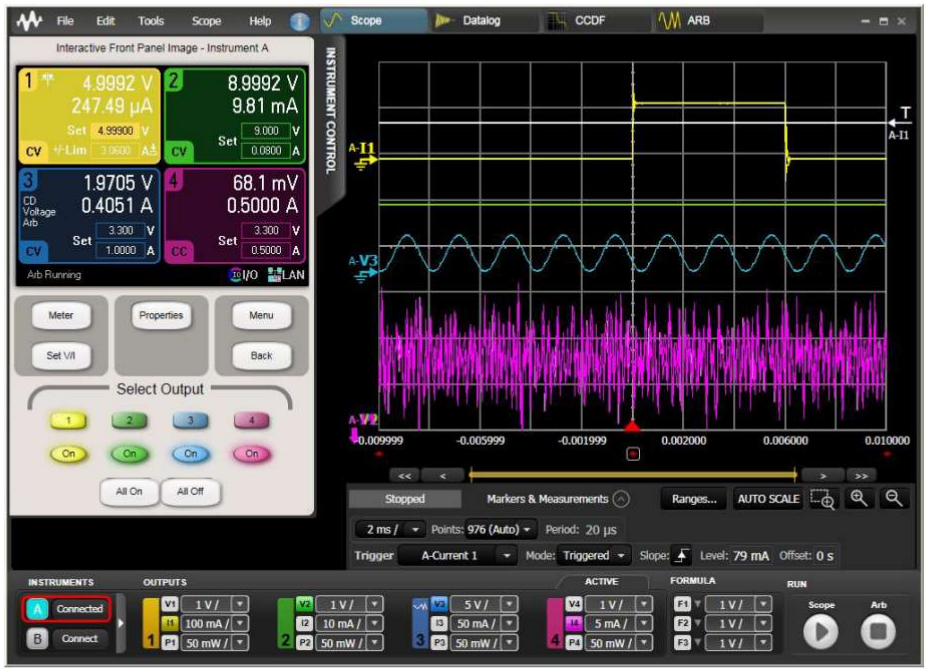

File Edit Tools Scope Help
Scope Datalog CCDF ARB
Interactive Front Panel Image - Instrument A
INSTRUMENT CONTROL
1 4.9992 V 247.49 µA Set 4.99900 V +Lim 3.0600 A CV
2 8.9992 V 9.81 mA Set 9.000 V 0.0800 A CV
3 1.9705 V 0.4051 A CD Voltage Arb Set 3.300 V 1.0000 A CV
4 68.1 mV 0.5000 A Set 3.300 V 0.5000 A CC
Arb Running I/O LAN
Meter Properties Menu
Set V/I Back
Select Output
1 2 3 4
On On On On
All On All Off
A-I1
A-V3
A-V2
T A-I1
-0.009999 -0.005999 -0.001999 0.002000 0.006000 0.010000
Stopped Markers & Measurements Ranges... AUTO SCALE
INSTRUMENTS OUTPUTS ACTIVE FORMULA RUN
A Connected
B Connect
V1 1 V/ I1 100 mA/ P1 50 mW/
V2 1 V/ I2 10 mA/ P2 50 mW/
V3 5 V/ I3 50 mA/ P3 50 mW/
V4 1 V/ I4 5 mA/ P4 50 mW/
F1 1 V/ F2 1 V/ F3 1 V/
Scope Arb
2 ms/ Points: 976 (Auto) Period: 20 µs
Trigger A-Current 1 Mode: Triggered Slope: Level: 79 mA Offset: 0 s

Figure 57 Control and Analysis Software for N6705B Screenshot

5.4 Apps Instrumentation

The TRIANGLE testbed provides several means for extracting information needed to calculate KPIs in apps. Some KPIs can be calculated by measuring the time between two UI events or actions, e.g. the presence of a specific element in the screen, or when a certain user action is performed in the context of an app user flow.

However, to compute KPIs associated to internals actions in the apps, the apps have to be "instrumented" to provide the required information. This app instrumentation must be done by the App developer before submitting the app to the TRIANGLE testbed for testing. This app instrumentation must be both easy to use and lightweight enough, to avoid interfering with the measurements.

The Application Instrumentation Library (Instrumentation Library or just Library for short) is a library provided by the TRIANGLE project to app developers, in order to facilitate how to extract measurements from inside their applications. The measurements performed through the Instrumentation Library are stored along other measurements gathered during a test case execution. This Library provides the necessary measurement points for running the test specifications defined within the TRIANGLE project, and computing the corresponding KPIs and metrics. In addition, the Library allows app developers to log additional measurements outside of the ones defined within the project, and store them with the rest of the measurements. The measurements that app developers will be able to get from the Portal will include both “custom” and “standard” measurements. This library is available only for Android applications. The same library can also be used in Unity applications for Android. Some parts are written in a generic manner, to be applicable to other future library implementations. This section describes the Library contents, and how to use it inside an application. It also describes the current internal format of the messages produced by the Library, although this information is internal and subject to change.

5.4.1 Instrumentation Library for Android

For Android apps, measurements are written to the system-wide log, called logcat [6]. All apps running in an Android device can write to this log, and the messages are timestamped on the device. The contents of the logcat can be retrieved online, e.g. while the Android device is connected to a computer through USB, or offline. Messages can be filtered by their tag (set by the app that sends them) and priority (e.g. verbose, info, warning and error).

To univocally identify the data that is being written as part of the data required for a certain KPI, the app will write to the log with a very specific tag and message format. This enables filtering and parsing the required data from the testbed.

TRIANGLE provides a library that app developers can use to indicate the measurement points inside the application. The measurements collected at these points are used to calculate the KPIs. This library takes care of giving the proper tag and format to the log messages, with a developer friendly interface. This library is optimized for speed and memory, e.g. object allocation is minimized by using static methods and objects whenever possible.

The library provides a set of abstract classes in the TRIANGLE portal (eu.TRIANGLE_project.instrumentation.kpis) package, which enables app developers to provide measurements. To make their intended usage clearer for app developers, the measurements have been grouped into classes, and organized into sub-packages, according to the KPIs which require them. The measurements required by a KPI are represented by a single abstract class. A KPI typically requires more than one measurement, which we call KPI measurement points in TRIANGLE. Each class provides static methods that must be called by the developer to provide data for a specific measurement point. These methods can take zero or more parameters, depending on what data must be provided for each measurement point. A method with no parameters can also output useful information, such as the presence of an event and its timestamp.

For instance, a KPI that involves measuring the time to download content from the app servers would be supported by a DownloadContentTime abstract class. This KPI may have three measurement points: the content size, and time when the download started and finished. Therefore, this class provides three methods to provide this information, which the developer has to call from the app code when the appropriate condition is met. In addition, there are overloaded versions of the methods that indicate the start or end of the download, with an additional timestamp parameter. If no timestamp is given, it is calculated when the method is called. The explicit version is useful if the developer must obtain the required timestamp from a particular source.

Figure 58 shows how the KPI classes are organized for the user experience KPIs that were identified in deliverable D2.1, section 5.1.4. The KPIs for each of the app categories are contained in separate packages, with a “shared” package containing KPI classes that can be applied to more than one category.

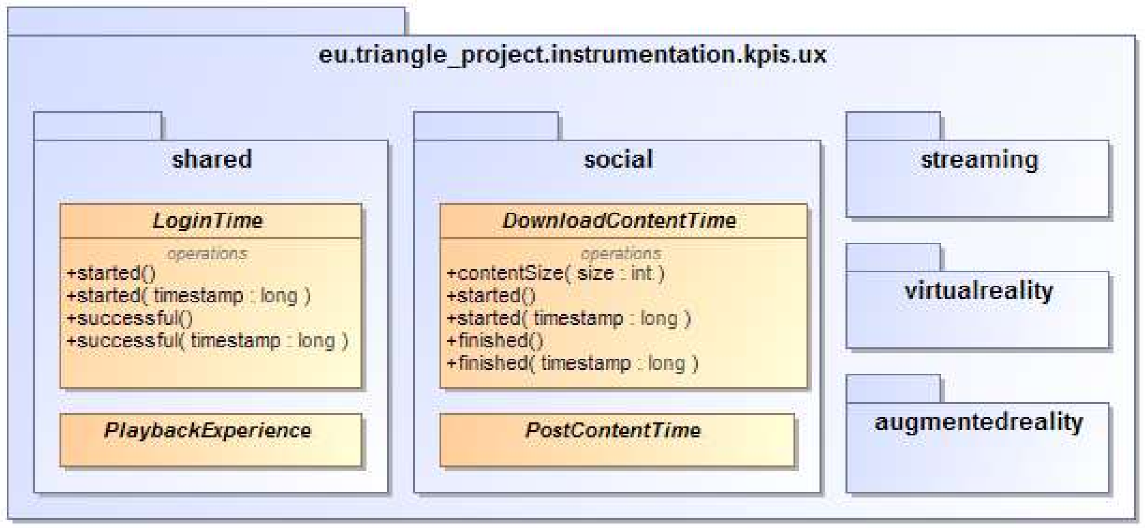

eu.triangle_project.instrumentation.kpis.ux
shared
LoginTime
operations
+started()
+started(timestamp : long)
+successful()
+successful(timestamp : long)
PlaybackExperience
social
DownloadContentTime
operations
+contentSize(size : int)
+started()
+started(timestamp : long)
+finished()
+finished(timestamp : long)
PostContentTime
streaming
virtualreality
augmentedreality

Figure 58 Overview of organization of instrumentation classes for a subset of KPIs

The KPIs themselves belong to a particular app feature, e.g. “login” or “post picture”. The measurements also belong to these features, and can be used in one or more KPIs for that feature. Finally, the features are grouped by the use case to which they belong, e.g. “live streaming services” or “social networking”. Each measurement may have zero or more arguments that must be filled in by the user. These arguments can be of any of the four following types:

- Boolean
- Integer
- Floating point
- String

TRIANGLE has defined a set of “standard measurements”, which are used to compute the KPIs defined for the TRIANGLE test cases. The package/class hierarchy of the instrumentation library provides a clear path to the appropriate measurement, so that it is possible to find the appropriate method/function to call easily. The general structure is:

- Use cases
    - Features
        - Measurements

Additionally, the library provides means to app developers to include additional measurements, called “custom measurements”. These measurements are parsed and stored alongside the rest of the measurements, but they are not included as part of any standard KPI computation. To distinguish them from regular measurements, all these measurements are organized into a special use case called “Custom”. When logging a custom measurement, the Library user can define to which feature and measurement it belongs. This affects the classification of the measurements, when stored in the measurements database. In addition, the user can provide zero or one arguments for a custom measurement

Annex 3 provides a list of all the measurements supported by the Instrumentation Library.

5.4.2 TAP Support for Instrumentation Library

A TAP plugin has been designed to extract measurement data written by the Instrumentation Library, and make it available as a result. Together with the OML (See Section 5.5) plugin for TAP, this enables cross referencing KPI data from the instrumentation with data from other tools, in the same OML database.

Table 29 shows the instruments provided by the TAP plugin. A generic IDutLogParserInstrument interface defines common methods to extract measurement data from DUT logs.

Table 29 Measurement parsing TAP plugin instruments

Instrument	**Setting**	**Description**
(IDutLogParser-Instrument)		Interface that must be implemented by instruments that support parsing measurement data from the log of a DUT.
Logcat Parser *(LogcarParser-Instrument)*		Instrument that implements IDutLogParser for parsing measurements from Android's logcat.
	ADB / ADB path	Path to the ADB executable in the local file system that will be used to interact with the logcat.

Table 30 shows the main test steps provided by the TAP plugin. The two main steps are setting up which measurements will be parsed, and performing the actual parsing. By default, all measurements will be monitored, but the subset that is required for a particular KPI can be selected. The ParseDutLogStep performs the actual log parsing for the previously configured measurements.

Table 30 Measurement parsing TAP plugin test steps

Test step	**Setting**	**Description**
Setup measurement parsing *(MeasurementPasrsing-SetupStep)*		Select which measurements will be parsed from the log of a DUT, through an IDutLogParserInstrument.
	DUT / DUT log parser	Select the IDutLogParserInstrument that will be used to parse the DUT log.
	DUT / Device ID	ID of the device whose log will be parsed.
	Measurements / Parse all	If enabled, the test step will try to detect and parse all possible measurements that can be generated by the instrumentation library. If disabled, select a KPI whose measurement points will be parsed in the next setting.
	Measurements / KPI	Select a KPI whose measurement points will be parsed.
Parse DUT log *(ParseDutLogStep)*		Perform the actual parsing of the log of a DUT, to extract the previously configured measurements.
	DUT / DUT log parser	Select the IDutLogParserInstrument that will be used to parse the DUT log.

	Action / Action	Possible values: Activate, Deactivate, Single Shot. If Single Shot is selected, the current contents of the log will be examined to parse the measurements. The other two values can be used to start log parsing in the background, so that other test steps can be executed in between.
Setup Logcat regex parsing (LogcatRegexParsing-SetupStep)		Configures a custom measurement parsing rule that can be used to extract data from logcat with a custom format not part of the standard measurement points.
	DUT / Logcat parser	LogcatParserInstrument that will be used to parse the custom measurements.
	DUT / Device ID	ID of the device whose log will be parsed
	Result / Name	Name of the result that will be published when a match is found.
	Logcat filter / Tag	Filter the contents of the logcat so that only messages with the given tag are considered.
	Logcat filter / Priority	Filter the contents of the logcat so that only messages with the given priority (e.g. verbose, info, error) are considered.
	Regex / Regex	The regular expression (regex) that will try to be matched in the filtered logcat messages. This regex may have a capturing groups. When a message matches the regex, the values of these capturing groups will be published as results, with the names configured below.
	Regex / Result names	Comma separated list of result names to be given to the value from each capture group in a regex match.

Specific steps to perform ad-hoc parsing will be considered. LogcatRegexParsingSetupStep allows the user to parse messages that match a given regular expression. The capture groups of the match will be published as results of the test step, with the names given in the step settings.

5.4.3 Measurement Calculation without Instrumentation Library

In case it is not possible to use the Instrumentation Library on your application (for example, because it has been developed using the Android NDK or it is not possible to include external libraries), app developers can still instrumentalize their applications and take advantage of the automatic measurement calculation provided by the TRIANGLE testbed. This is possible by writing messages that follow the same format as the messages generated by the instrumentation library, and generating them at the same situations in which a method from the library would be included.

For example, the following snippet could be used for generating the required measurement point at the end of an FTP download in an Android NDK application:

```
#define LOGI(...) ((void)__android_log_print(ANDROID_LOG_INFO, "ftp.Native", __VA_ARGS__))
#define INST(...) ((void)__android_log_print(ANDROID_LOG_INFO, "TriangleInstr", __VA_ARGS__))

// [...]

if (downloadFtp(url_c, port_c, user_c, pass_c, buffer_c, binding_c, &speed)) {
  LOGI("FTP download completed, speed %.2f kbytes/s", speed);
  INST("Hs\tDownload\tFile Download - End\t%i\ttrue", transferId);
} else {
  LOGI("FTP download failed);
  INST("Hs\tDownload\tFile Download - End\t%i\tfalse", transferId);
}
```

Figure 59 Snippet for generating measurement points with the same format that the instrumentation library

Where “\t” corresponds to the Tab character. The message that corresponds to each of the supported measurement points can be seen on the 'Measurement point methods’ section on the documentation of the specific Instrumentation Library for each operating system, as “Generated message”.

5.5 OMF Measurement Library (OML)

All the control and measurement tools used during a test may generate significant amounts of data and results. This data is useful for aggregating different measurements and events, and measuring KPIs that span across tools. The TRIANGLE testbed uses the OML measurement framework [7] to centralize the collection of measurements and other data generated during the test.

5.5.1 OML Architecture

OML follows a client-server architecture, where several OML clients collect measurements and sends them to one or more OML servers using a custom OML protocol. OML clients can use one of the existing client libraries for several programming languages, instead of doing their own protocol implementation.

The measurements from an OML client are grouped into measurements points. Data from each measurement point can be filtered at the client, to perform some processing before forwarding the measurements.

In the TRIANGLE testbed, all measurements are sent to a central OML server which collects and stores them. This OML server uses a PostgreSQL database server as a backend to store the measurements, not related to the database managed by TAP. The OML server stores the data from each test in a separate database, in the same PostgreSQL database server. Before starting a test, OML clients must be configured with the same test ID, so that the OML server stores all their measurements in the same database.

The data from each measurement point is stored in a separate table in the database. Measurement points produce data as key-value pairs. The measurement point table will contain one column per each key used in the measurements, plus additional metadata, such as the ID of the client, and timestamps at both the client and the server. Each measurement sent by the client will be stored in a row on the corresponding table. The same measurement point ID can be used by more than one OML client. The measurements sent from the two clients will be stored on the same table, but they can be distinguished by the client ID stored on each row. Two other tables will be created with additional metadata per test.

Figure 60 shows the high-level architecture of the OML framework with an example. Two OML clients are sending measurements to a single OML server. The measurements from one client are grouped into two separate measurement points (MPA and MPB). All the measurements can be filtered before they are actually sent to the OML server. The measurements from each test will be stored in a separate database, in the same PostgreSQL database server.

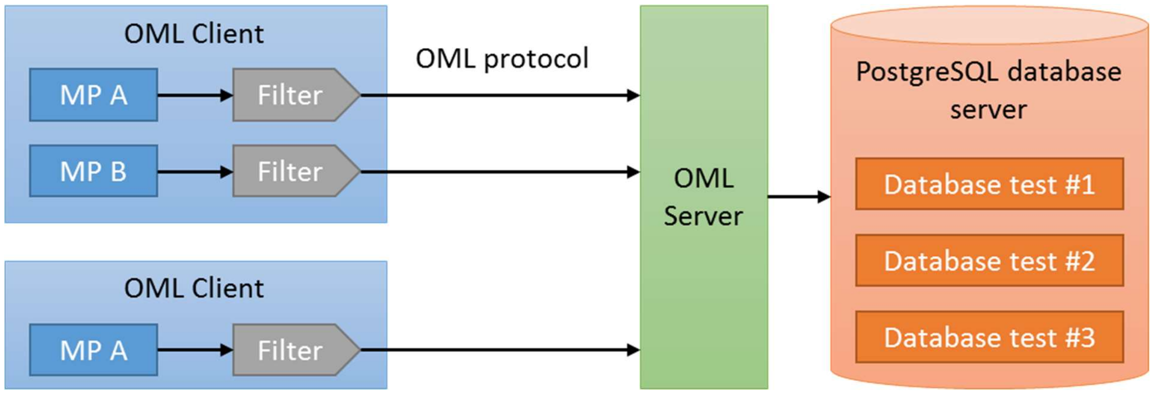
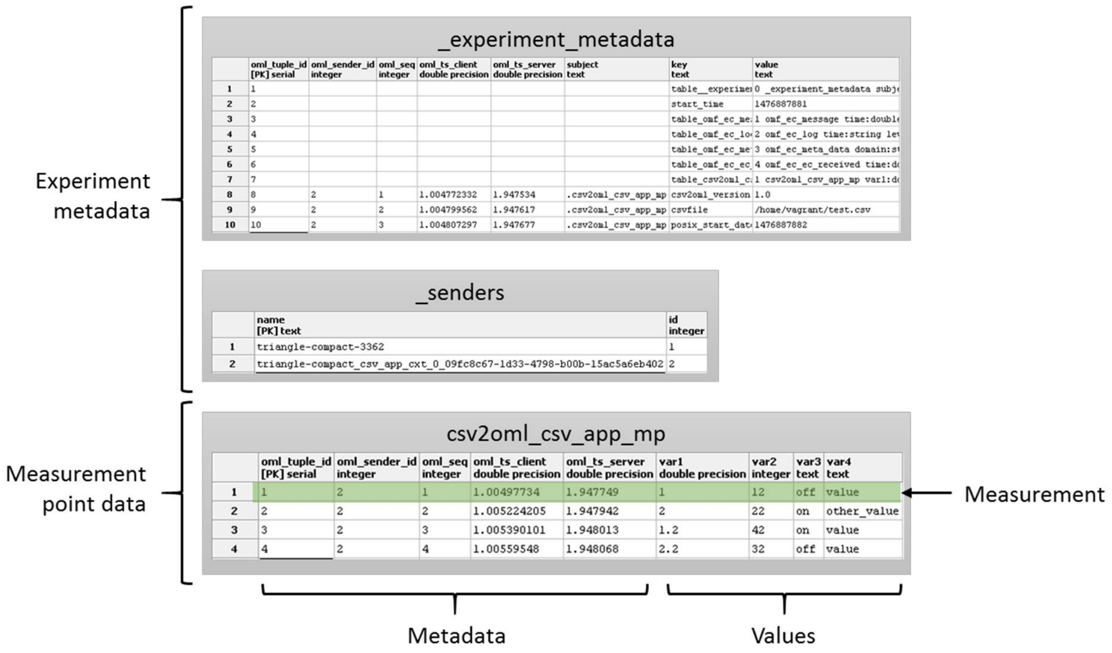


Figure 60 OML architecture

Figure 61 show an example database created by the OML server in the PostgreSQL database server for storing measurements from a test. This test contains a single measurement point, called csv2oml_csv_app_mp, whose measurements are stored in a table of the same name. The first five columns of that table store metadata associated with each measurement, while the other four store the actual values of each measurement. All the measurements have been written by the same client, whose ID is shown in the oml_sender_id column.



_experiment_metadata

	oml_tuple_id [PK] serial	oml_sender_id integer	oml_seq integer	oml_ts_client double precision	oml_ts_server double precision	subject text	key text	value text
1	1						table__experime	0 _experiment_metadata subj
2	2						start_time	1476887881
3	3						table_omf_ec_me	1 omf_ec_message time:double
4	4						table_omf_ec_lo	2 omf_ec_log time:string lev
5	5						table_omf_ec_me	3 omf_ec_meta_data domain:st
6	6						table_omf_ec_ec	4 omf_ec_ec_received time:do
7	7						table_csv2oml_c	1 csv2oml_csv_app_mp var1:do
8	8	2	1	1.004772332	1.947534	.csv2oml_csv_app_mp	csv2oml_version	1.0
9	9	2	2	1.004799562	1.947617	.csv2oml_csv_app_mp	csvfile	/home/vagrant/test.csv
10	10	2	3	1.004807297	1.947677	.csv2oml_csv_app_mp	posix_start_dat	1476887882

_senders

	name [PK] text	id integer
1	triangle-compact-3362	1
2	triangle-compact_csv_app_cxt_0_09fc8c67-1d33-4798-b00b-15ac5a6eb402	2

csv2oml_csv_app_mp

	oml_tuple_id [PK] serial	oml_sender_id integer	oml_seq integer	oml_ts_client double precision	oml_ts_server double precision	var1 double precision	var2 integer	var3 text	var4 text
1	1	2	1	1.00497734	1.947749	1	12	off	value
2	2	2	2	1.005224205	1.947942	2	22	on	other_value
3	3	2	3	1.005390101	1.948013	1.2	42	on	value
4	4	2	4	1.00559548	1.948068	2.2	32	off	value

Figure 61 OML database for an experiment with a single measurement point

OML also provides utilities to share data collected by tools that do not implement OML functionality. The csv2oml command line tool sends the contents of a CSV file to an OML server, using the headers of each column as the name of each measurement component.

5.5.2 TAP Support for OML

OML measurements have to be configured and performed as part of a TAP test plan that performs a test. This section describes the design for a TAP plugin to support OML measurements in TAP. This plugin includes several test steps and instruments, as usual.

Table 31 shows the instrument provided by the TAP plugin. This instrument handles the connection to an OML server, and thus the user needs to provide the host and port where that server is running. The server keeps track of the test ID, so that the measurements collected during the execution of a TAP test plan are stored in the same database. It also shows an interface that has been defined for new instruments: ICsvInstrument, Instruments that implement this interface declare that they produce a CSV file as a result of their normal operation. The interface declares only one property, with the path to the CSV file that will be generated.

Table 31 OML TAP plugin instruments

Instrument	Setting	Description
OML Server (OmlServerInstrument)		Handles connection to OML server. Keeps track of the test ID, so that measurements are stored on the same database.
	Server / Host	Host name or IP of OML server.
	Server / Port	Port of OML server.
(ICsvInstrument)		Interface for new TAP instruments that provide CSV files. Instruments that implement it,

Table 32 shows the two test steps provided in the TAP plugin. OmlSetupStep performs basic OML configuration. SendCsvStep sends the rows of a CSV file as individual measurements to the OML server. This file can be one found on the local file system, or one provided by an instrument that implements the ICsvInterface. This step is helpful to add support for tools that generate CSV files as results, without having to implement a full TAP plugin.

Table 32 OML TAP plugin test steps

Test step	Setting	Description
OML Setup (OmlSetupStep)		Performs initial setup for the OML measurement collection.
	OML / Server	OmlServerInstrument which will be set up.
	Test / Custom name	If enabled, the user can set a custom name for the new test, which will be the name of the database where measurements will be stored. If disabled, a unique test name will be automatically generated.
Send CSV to OML		Sends the contents of a CSV file to an OML server.

(SendCsvStep)		
	OML / Server	OmlServerInstrument to which measurements will be sent.
	CSV / Source	Selection between: File, Instrument. Select the source of the CSV file: a file in the file system, or a CSV generated from a compatible ICsvInstrument.
	CSV / File path	Path to a CSV file in the file system, whose contents will be sent to the OML server.
	CSV / Instrument	An ICsvInstrument that will provide the CSV file that will be sent to the OML server.
	CSV / Has Headers	If true, the first line of the CSV file will be interpreted as a header. The values of this header will be used as the names for each column. If false, column names will be assigned automatically.
	Measurements / Measurement point	Name of the measurement point defined for this CSV file.

The plugin also provides a TAP result listener that sends the results published by TAP test steps to an OML server, as shown in Table 33. A result listener is the best option for publishing results from plugins which follow TAP conventions and publish results through the TAP API.

Table 33 OML TAP plugin result listeners

Result listener	**Setting**	**Description**
OML Result Listener *(OmlResultListener)*		Result listener that sends the results received from TAP test steps to an OML server. The name of the ResultTable published by a test step will be used as the measurement point name.
	OML / Server	OmlServerInstrument to which measurements will be sent.

Figure 62 shows the main classes which compose the implementation of the TAP plugin for OML. Only one instrument has been defined, OmlServerInstrument, to represent the connection to a particular OML server. This instrument holds the name of the test, which is used to identify the database where all measurements of a TAP test plan execution are sent to.

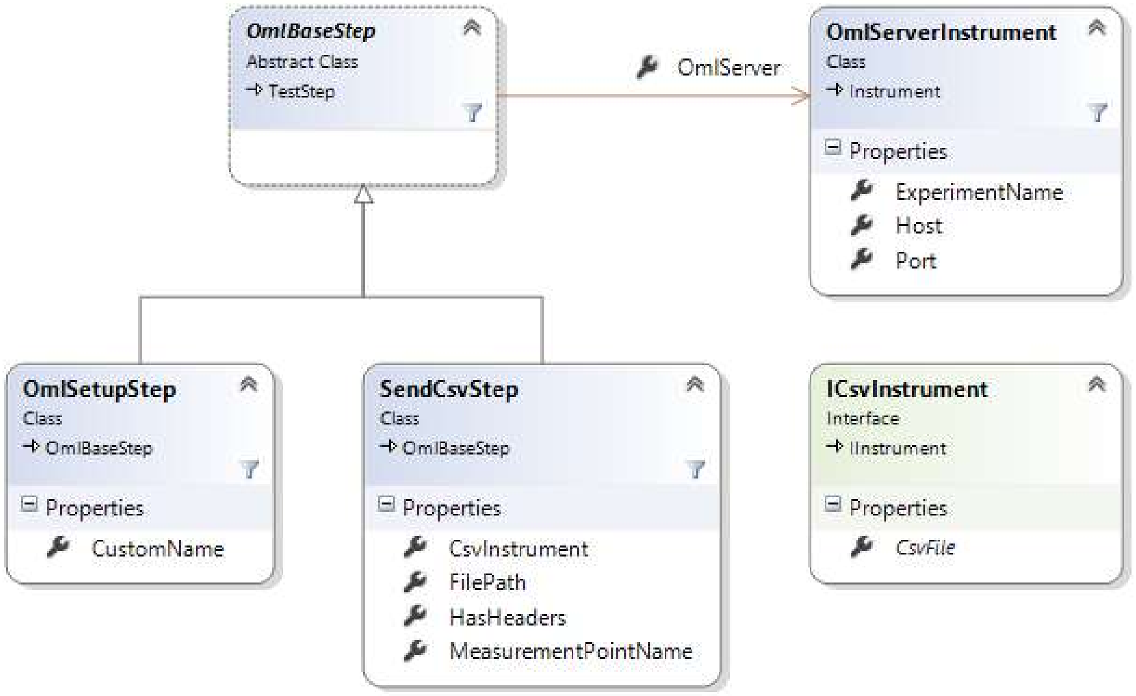
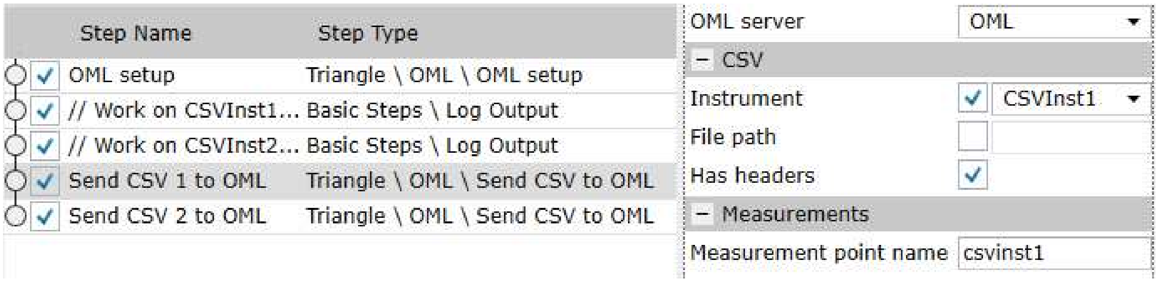


Figure 62 Main classes of TAP plugin for OML

Figure 63 shows an example of a TAP test plan using the test steps and instruments included in the OML TAP plugin. The test plan uses an OmlServerInstrument, which collects measurements, and two instruments that implement the ICsvInstrument interface. The first step is to configure the name of the OML test, which is followed by the steps that make use of both ICsvInstruments. It finalizes with two test steps that send the CSV files generated by both instruments. The Instrument selector allows the user to select only those instruments that implement the required interface.



	Step Name	Step Type
✓	OML setup	Triangle \ OML \ OML setup
✓	// Work on CSVInst1...	Basic Steps \ Log Output
✓	// Work on CSVInst2...	Basic Steps \ Log Output
✓	Send CSV 1 to OML	Triangle \ OML \ Send CSV to OML
✓	Send CSV 2 to OML	Triangle \ OML \ Send CSV to OML

OML server	OML
− CSV	
Instrument	✓ CSVInst1
File path	
Has headers	✓
− Measurements	
Measurement point name	csvinst1

Figure 63 Example of TAP using OML TAP plugin

5.6 KPIs Computation

The measurements stored in the OML database serve as the source material for extracting and computing the KPI values. A specialized ETL (Extract, Transform, Load) performs this task.

Each test is executed to measure enough data to compute a set of KPIs. Therefore, the ETL tool needs as input which of the KPIs defined for the TRIANGLE Testing Framework can be computed from the experiment or test. This information was produced by the Orcomposutor, along with each of the test plans it generated. The computed KPIs are stored in a database different from the OML database.

For experimentation campaigns, the main body of the workflow ends here. The computed KPIs will be available for the user in the Portal, along with the raw measurements.

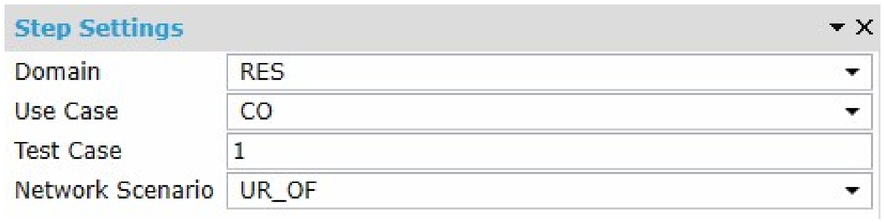
A generic data management framework (TRIANGLEKpi.Core) provides the basic functionality required for the generation of the KPIs based on the raw data obtained during the testplan execution.

A set of TAP steps (grouped on the 'Tap.Plugins.TRIANGLEKpi' plugin) makes use of the core package for calculating the different KPIs defined for each of the available domains.

5.6.1 Tap.Plugins.TRIANGLEKpi

The TRIANGLEKpi TAP plugin contains a set of test steps that can be used for generating the set of KPIs defined for each domain, using the results generated by the previous campaign execution as source. This plugin also defines an additional Loader based on the interface defined on the Core package: TapDataLoader can generate the initial StructuredData for the Pipelines by querying a set of data from the TAP's result database.

Prior to the execution of the KPI extraction steps the results generated by a testplan execution by using the 'Test Case Labeler' step must be labeled. This step is executed at the beginning of each campaign run and provides the basic information that TapDataLoader requires for loading the desired results.

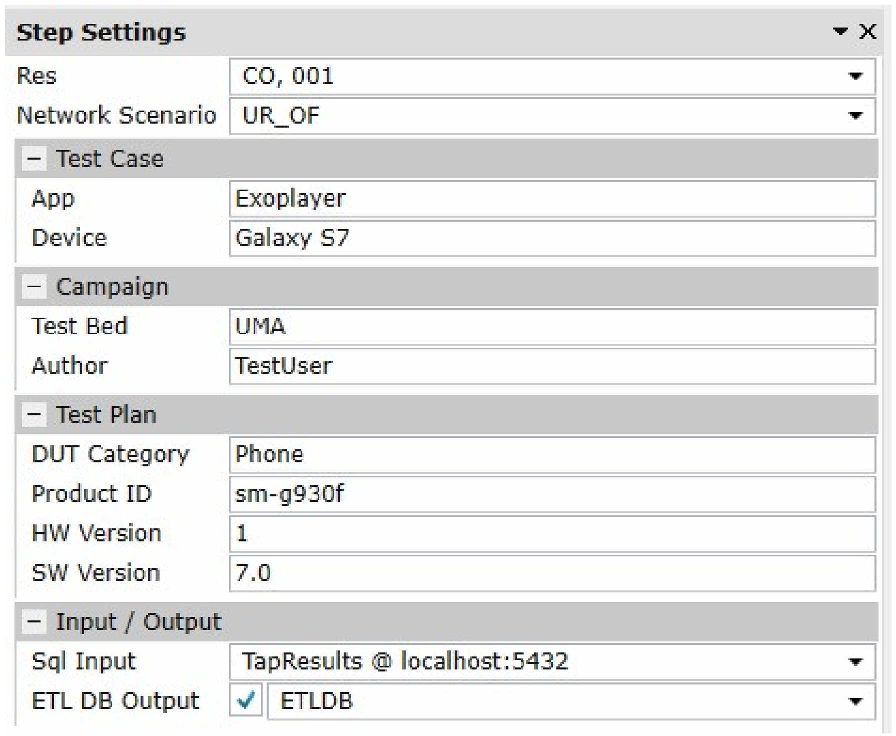


Domain	RES
Use Case	CO
Test Case	1
Network Scenario	UR_OF

Figure 64 Configuration parameters on the Test Case Labeller step

Each step exposes a similar set of configuration parameters:

- The basic settings include the Test Case id and network scenario of the testplan.
- The Test Case, Campaign and Test Plan sections can be used to define associated metadata for the generated KPIs.
- The Input / Output section defines the input database and optional ETL database: The steps can save the generated KPIs and metadata directly into an ETL database, or only as standard TAP results.



Step Settings

Res	CO, 001
Network Scenario	UR_OF
Test Case	
App	Exoplayer
Device	Galaxy S7
Campaign	
Test Bed	UMA
Author	TestUser
Test Plan	
DUT Category	Phone
Product ID	sm-g930f
HW Version	1
SW Version	7.0
Input / Output	
Sql Input	TapResults @ localhost:5432
ETL DB Output	✓ ETLDB

-

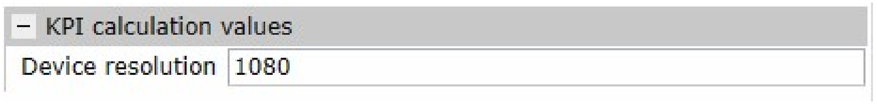
Figure 65 Configuration parameters on the RES step. All steps include the same basic settings.

Other features developed to support post–processing measurements into atomic KPIs for new domains are depicted in the following sections.

Application User Experience (AUE)

The AUE domain part of the TriangleKPI plugin processes all user-experience measurements as described in the test specifications. It calculates the respective KPIs using multiple Processors, for example Mode (for Video Resolution measurements), Average (for Time to Load the first Media Frame) or CutOff Radio.

The test step notably calculates KPIs based on the resolution used by the application (for example, during video playback or gaming). In these cases, the calculation is performed using 4k as the maximum resolution by default, but it is possible to specify the maximum resolution of the DUT for generating these MOS values using more realistic information.



KPI calculation values	
Device resolution	1080

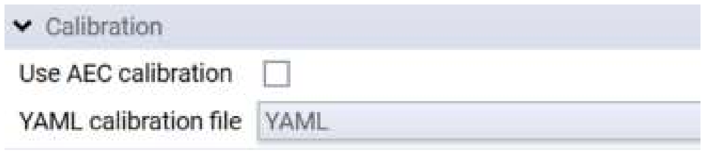
Figure 66 Additional settings on the AUE step

Each of the provided steps defines a custom Pipeline that is used to generate the desired KPIs

Application Energy Consumption (AEC)

The AEC domain takes as input Voltage and Current measurements, capturing the consumption of the device under test. Typically, a phone is powered directly through the Power Analyzer, rather than through its internal battery, which is disconnected. On another power analyser channel, the power leakage over USB is logged as well. Combined, the sum of the power over the two channels represent the total consumption of the device while running the test case.

To determine the consumption of the application itself, the measurements captured earlier need to be offset with calibration values, measured when the phone is in an idle state (see D3.6), without any application running. These calibration values are device-specific, as well as testcase specific, as the testcase determines whether the device screen is ON, which logging agent is running, etc.



Calibration

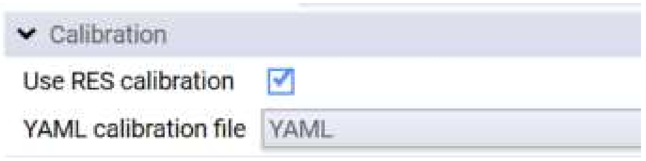
Use AEC calibration	☐
YAML calibration file	YAML

Figure 67 Use of calibration for AEC domain measurements post-processing

Application Resource Usage (RES)

The RES test step from TriangleKPI performs post-processing on measurements logged by the DEKRA agent running in parallel with the test application. This agent logs among other metrics the device's CPU, GPU and RAM usage while the application flow is being executed. These measurements are then averaged and post-processed into KPIs.

In the same way as for the AEC domain, to determine the application-specific KPIs rather than the device-specific KPIs, a substraction of calibration values need to be performed at the post-processing stage. This is done automatically by the test step when the calibration file is provided.



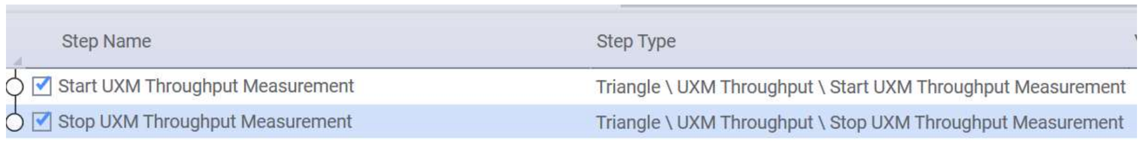
Calibration

Use RES calibration	✓
YAML calibration file	YAML

Figure 68 Use of calibration for RES domain measurements post-processing

Application Network Resources Usage Domain (NWR)

In order to generate measurements for this domain, an additional UXM plugin has been developed, called UXM Throughput. It communicates with the UXM at the beginning and the end of each measurement iteration, and records the overall downlink and uplink IP data transferred during the iteration. These measurements are written to the TAP Result Listener and picked by the post-processing steps from the TRIANGLEKPI package.



Step Name	Step Type
✓ Start UXM Throughput Measurement	Triangle \ UXM Throughput \ Start UXM Throughput Measurement
✓ Stop UXM Throughput Measurement	Triangle \ UXM Throughput \ Stop UXM Throughput Measurement

Figure 67 New test steps to capture IP throughput at each iteration

The new KPIs made available by this domain are Total DL IP traffic and Total UL IP traffic, and allow to assign a mark which quantifies how "mobile data-hungry" a given user flow is.

Application Reliability (REL)

As described in the Test Specification, the application reliability domain introduces three new atomic KPIs:

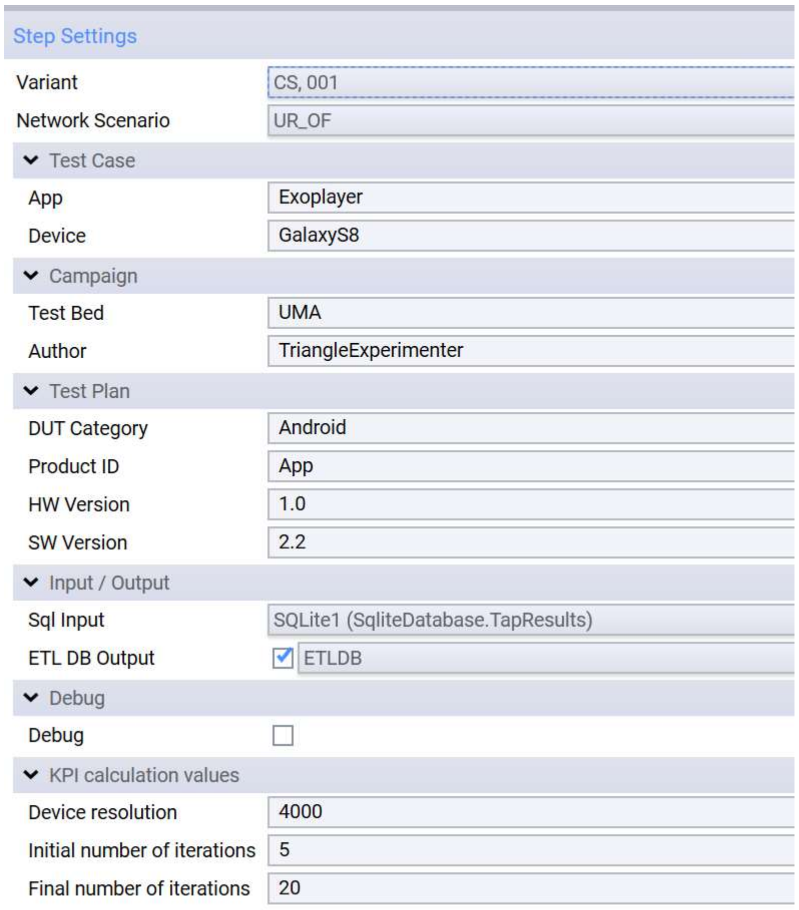
- Feature availability
- Feature auto-recovery after failure
- Performance degradation

The TRIANGLEKPI TAP plugin calculates the Degradation KPI based on test results across multiple iterations, for example by comparing an average result from the first 5 iterations to the average result from the last 20 iterations.

The plugin is supporting post-processing of multiple KPIs, especially all time-centric measurements (search time, access time, load time), resolution (for content streaming use cases), success rate and throughput. Implementation-wise, this has been achieved by adding new Processors which return the Degradation KPI from per-iteration measurements.

The particularity of this KPI lies in its definition of evaluation values (kpi_min, kpi_max and kpi_type), as it is not straightforward to determine whether a high value of Degradation is a result improvement or a regression. This depends whether a source KPI is of type “Higher is better” (such as resolution) or “Lower is better” (such as access time). Currently the evaluation values are tuned in consequence.

Note that to generate meaningful application reliability KPIs, a test with a large number of iteration needs to be ran (at least 25, according to test specifications). However, the post-processing TAP step within TRIANGLEKPI is more flexible as it offers the initial number and final number of iterations as test step parameters.



Step Settings

Variant	CS, 001
Network Scenario	UR_OF
Test Case	
App	Exoplayer
Device	GalaxyS8
Campaign	
Test Bed	UMA
Author	TriangleExperimenter
Test Plan	
DUT Category	Android
Product ID	App
HW Version	1.0
SW Version	2.2
Input / Output	
Sql Input	SQLite1 (SqliteDatabase.TapResults)
ETL DB Output	✓ ETLDB
Debug	
Debug	☐
KPI calculation values	
Device resolution	4000
Initial number of iterations	5
Final number of iterations	20

Figure 69 Example of Reliability KPIs post-processing test step

5.7 Metrics and Mark Computation

Figure 70 shows the work flow of the measurements and results in the TRIANGLE testbed. All the measurements collected during the testing campaign are stored in the main data base of TAP, in CSV files and in an OML data base. TAP data base is the main entry point for the post-processing process to compute the KPIs and the TRIANGLE mark. Once the KPIs are calculated the ETL framework will map these KPIs into MOS values for each of the domains considered in the project and will provide a global TRIANGLE mark based on the aggregation of these MOS values. CSV files contain raw measurements and are also provided to the experiments through the TRIANGLE Portal. The OML data base also contains raw results providing a persistent storage and enabling their organization and sharing. The Web visualizer of the results is based on the OML data base.

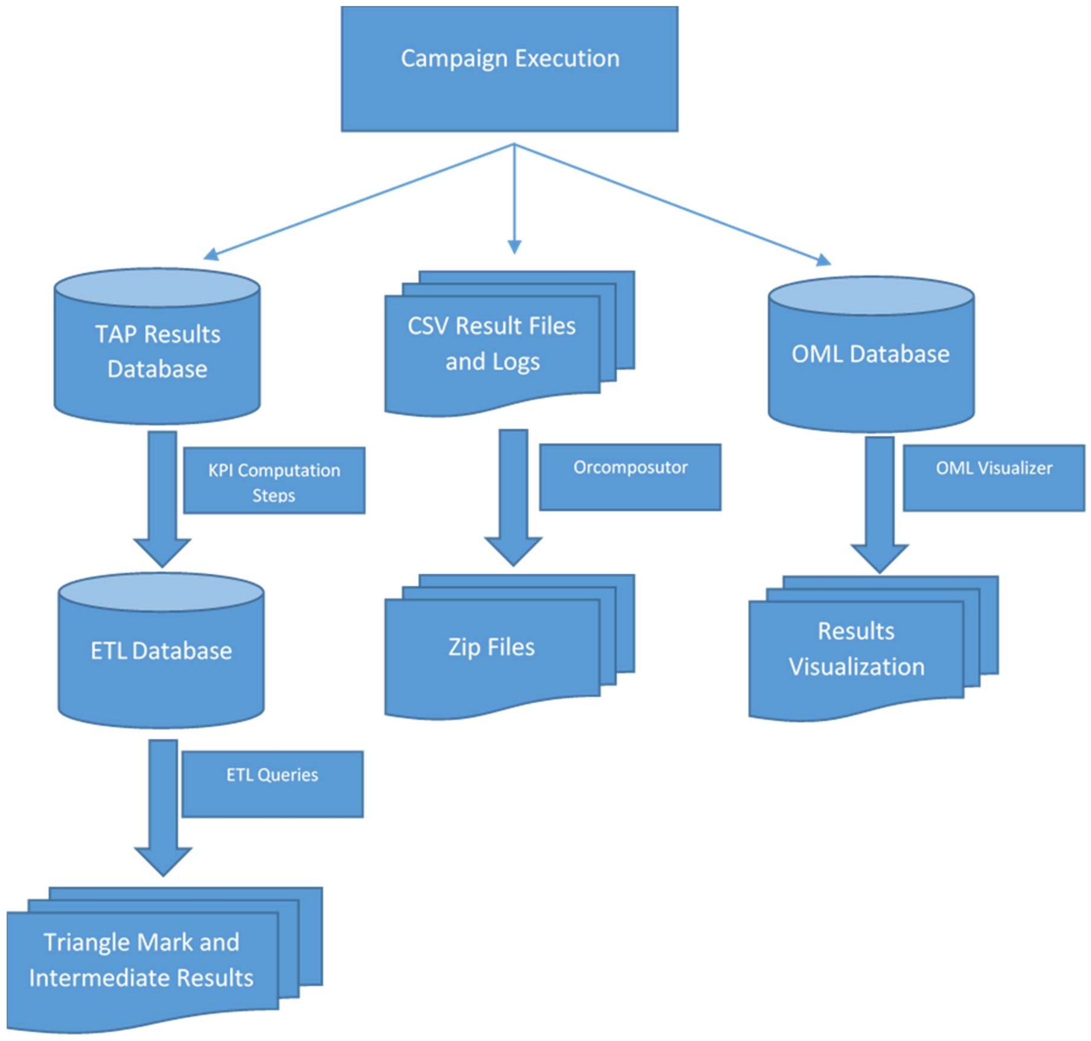
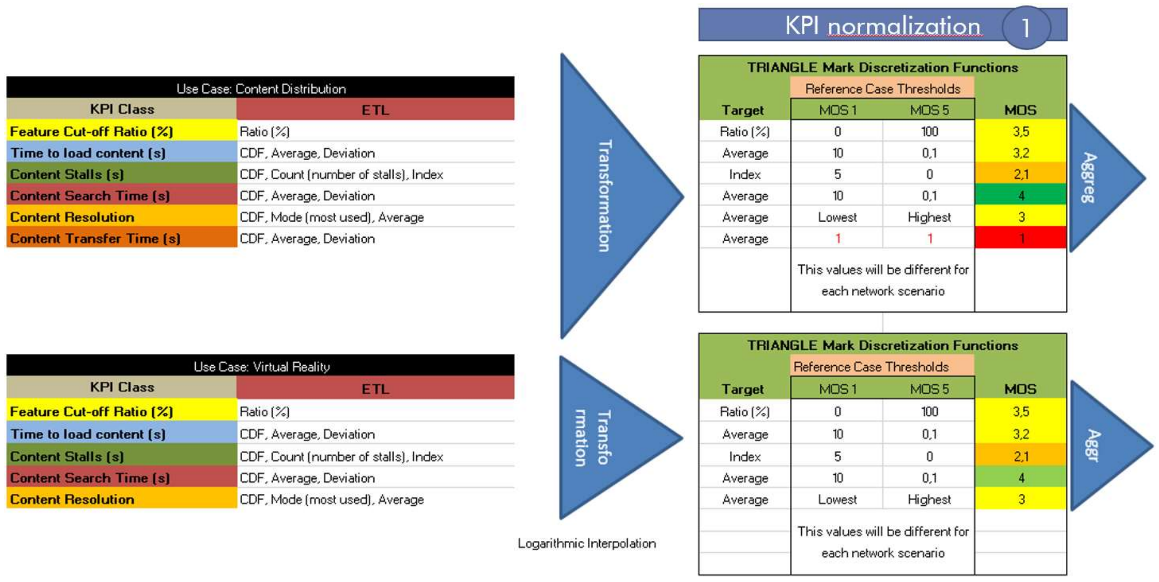


Figure 70 Post-processing and reporting tools

For the post-processing of the results and the generation of the TRIANGLE Mark we are following the ETL (Extract, Transform, and Load) process. The calculation of the TRIANGLE Mark is divided into five different steps, where each step performs the normalization or aggregation of the values generated by the previous step in the chain:

Raw data > KPIs > KPI MOS > Scenario MOS > Domain MOS > use case MOS > TRIANGLE Mark



Use Case: Content Distribution

KPI Class	ETL
Feature Cut-off Ratio (%)	Ratio (%)
Time to load content (s)	CDF, Average, Deviation
Content Stalls (s)	CDF, Count (number of stalls), Index
Content Search Time (s)	CDF, Average, Deviation
Content Resolution	CDF, Mode (most used), Average
Content Transfer Time (s)	CDF, Average, Deviation

TRIANGLE Mark Discretization Functions

Target	Reference Case Thresholds MOS 1	Reference Case Thresholds MOS 5	MOS
Ratio (%)	0	100	3,5
Average	10	0,1	3,2
Index	5	0	2,1
Average	10	0,1	4
Average	Lowest	Highest	3
Average	1	1	1
	This values will be different for each network scenario		

Use Case: Virtual Reality

KPI Class	ETL
Feature Cut-off Ratio (%)	Ratio (%)
Time to load content (s)	CDF, Average, Deviation
Content Stalls (s)	CDF, Count (number of stalls), Index
Content Search Time (s)	CDF, Average, Deviation
Content Resolution	CDF, Mode (most used), Average

TRIANGLE Mark Discretization Functions

Target	Reference Case Thresholds MOS 1	Reference Case Thresholds MOS 5	MOS
Ratio (%)	0	100	3,5
Average	10	0,1	3,2
Index	5	0	2,1
Average	10	0,1	4
Average	Lowest	Highest	3
	This values will be different for each network scenario		

Figure 71 Post-processing step 1

This division in steps allows us to separate the process in different modules that perform a part of the calculation. With this separation we can easily introduce modifications on any of the steps without affecting the process as a whole. The steps are:

- KPI normalization: The KPIs obtained from the test campaign execution are normalized into a single MOS value, so that the following aggregations are not affected by the different units used for each measurement.
- Scenario MOS aggregation: This step aggregates all the KPIs for each of the scenarios of the campaign into a single MOS value.
- Domain MOS aggregation: In this step the Scenario MOS values for each Domain (calculated in the previous step) are aggregated into a single value.
- Use Case MOS aggregation: The values calculated in the previous step for each Use Cases are aggregated.
- TRIANGLE Mark calculation: The MOS values for each of the Use Cases are aggregated one last time into a single value.

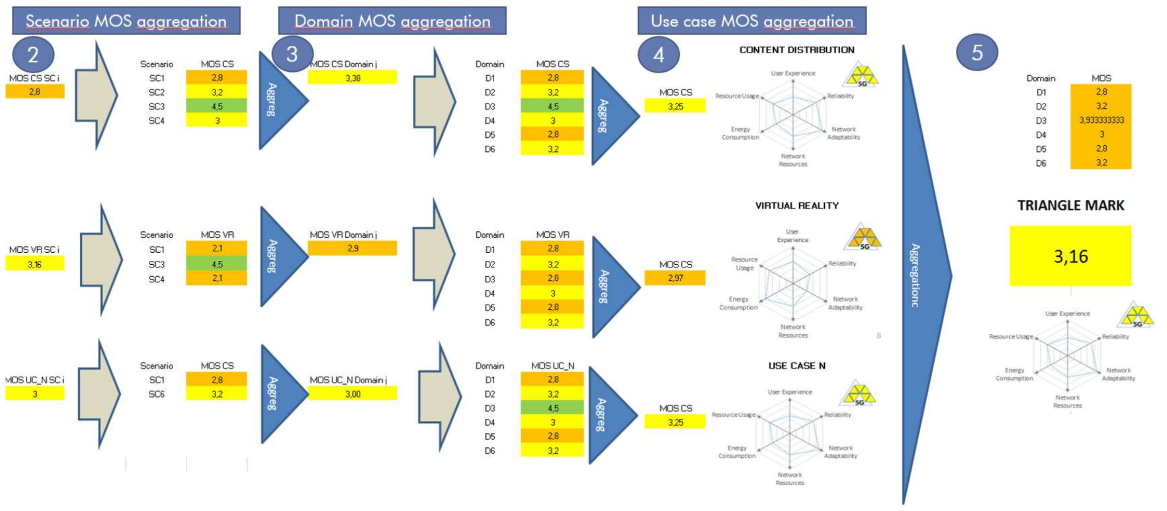


Figure 72 Post-processing steps 2, 3, 4 and 5.

5.7.1 Transforming KPIs into synthetic-MOS

The transformation of KPIs into QoE scores is the most challenging step in the TRIANGLE framework. The execution of the test cases will generate a significant amount of raw measurements about several aspects of the system. Specific KPIs can then be extracted through statistical analysis: mean, deviation, cumulative distribution function (CDF), or ratio.

The KPIs will be individually interpolated in order to provide a common homogeneous comparison and aggregation space. The interpolation is based on the application of two functions, named Type I and Type II. By using the proposed two types of interpolations, the clear majority of KPIs can be translated into normalized MOS-type of metric (synthetic- MOS), easy to be averaged in order to provide a simple, unified evaluation.

Type I

This function performs a linear interpolation on the original data. The variables minKPI and maxKPI are the worst and best-known values of a KPI from a reference case. The function maps a value, v, of a KPI, to v' (synthetic-MOS) in the range [1-to-5] by computing the following formula

$$v' = \frac{v - min_{KPI}}{max_{KPI} - min_{KPI}}(5.0 - 1.0) + 1.0$$

This function transforms a KPI to a synthetic-MOS value by applying a simple linear interpolation between the worst and best expected values from a reference case. If a future input case falls outside the data range of the KPI, the new value will be set to the extreme value minKPI (if it is worse) or maxKPI (if it is better).

Type II

This function performs a logarithmic interpolation and is inspired on the opinion model recommended by the ITU-T in [20] **[**20]for a simple web search task. This function maps a value, v, of a KPI, to v' (synthetic-MOS) in the range [1-to-5] by computing the following formula

$$v' = \frac{5.0 - 1.0}{\ln((a * worst_{KPI} + b)/worst_{KPI}))} \cdot (\ln(v) - \ln(a * worst_{KPI} + a)) + 5$$

The default values of a and b correspond to the simple web search task case (a = 0,003 and b = 0,12) [20][22] and the worst value has been extracted from the ITU-T G1030. If during experimentation a future input case falls outside the data range of the KPI, the parameters a and b will be updated accordingly. Likewise, if through subjective experimentation other values are considered better adjustments for specific services, the function can be easily updated.

Once all KPIs are translated into synthetic-MOS values, they can be averaged with suitable weights. In the averaging process, the first step is to average over the network scenarios considered relevant for the use case. This provides the synthetic-MOS output value for the test case. If there is more than one test case per domain, which is generally the case, a weighted average is calculated in order to provide one synthetic-MOS value per domain. The final step is to average the synthetic-MOS scores over all use cases supported by the application. This provides the final score, i.e., the TRIANGLE mark.

5.8 Support for testbed characterization and calibration

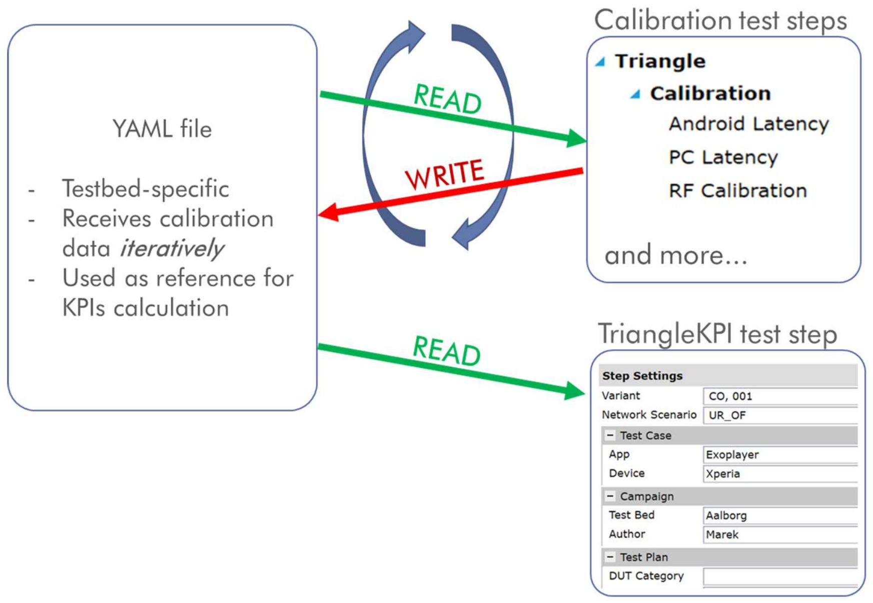
In order to enable testbed calibration, a series of TAP plugins and processes have been added to the testbed.

5.8.1 Calibration YAML file workflow

The calibration YAML file is a plaintext file with YAML syntax. There is one file per TRIANGLE testbed, and it contains all calibration and compensation values required to run the testbed in a meaningful and repeatable manner.

This file is accessed from TAP as a new instrument, which points to the file, and a series of functions have been implemented to read from and write back to the file.

The file is populated through running a TRIANGLE calibration campaign, where calibration and compensation values will be iteratively inserted into this file. The file contains compensation values relative to the testbed (such as its latency), relative to each device connected to the testbed (such as its idle state power and resource usage consumption, RF cabling compensation values, and other).



Step Settings

Variant	CO, 001
Network Scenario	UR_OF
− Test Case	
App	Exoplayer
Device	Xperia
− Campaign	
Test Bed	Aalborg
Author	Marek
− Test Plan	
DUT Category	

Figure 73 Iterative population of the calibration YAML file

At the beginning of a test campaign, the TAP template will initialize the testbed with compensation values extracted from the YAML file relative to the UE which is being currently tested. This improves the comparability of test results between devices.

5.8.2 TAP plugins for testbed characterization & calibration

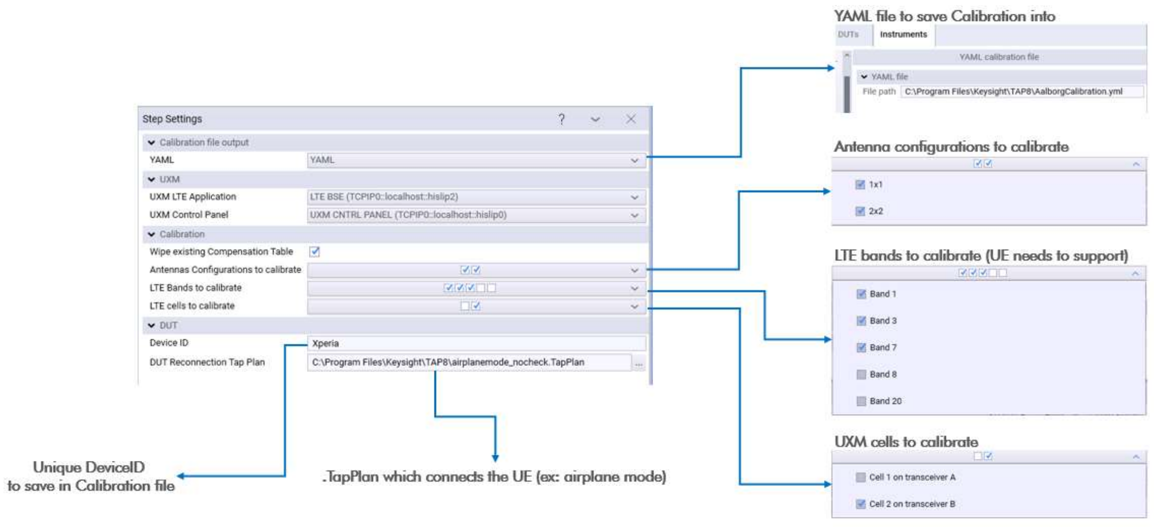
Cabling loss compensation

The DUTs used in the TRIANGLE project have to be modified to allow connectivity with various external instruments (e.g., UXM, power analyser) via cables. This may lead to a difference in the desired RF signal power and the actual RF signal power reaching the modem due to the cabling loss and connection loss. The cabling loss needs to be measured and compensated during the calibration procedure.

The measurement of the cable loss is performed in downlink, per DUT RX antenna and per LTE band. The calibration is based on UE-reports based on UXM (RSRP reports).

This is achieved via a TAP test step which:

- Generates RF DL compensation per connector, per band, per cell
- Writes the values in Calibration YAML file, for device under test
- Applies to UXM Control Panel


YAML file to save Calibration into
Antenna configurations to calibrate
LTE bands to calibrate (UE needs to support)
UXM cells to calibrate
Unique DeviceID to save in Calibration file
.TapPlan which connects the UE (ex: airplane mode)
DUTs
Instruments
YAML calibration file

Step Settings

Calibration file output
- YAML: YAML

UXM
- UXM LTE Application: LTE BSE (TCPIP0::localhost::hislip2)
- UXM Control Panel: UXM CNTRL PANEL (TCPIP0::localhost::hislip0)

Calibration
- Wipe existing Compensation Table: ✓
- Antennas Configurations to calibrate: ✓✓
- LTE Bands to calibrate: ✓✓✓☐☐
- LTE cells to calibrate: ☐✓

DUT
- Device ID: Xperia
- DUT Reconnection Tap Plan: C:\Program Files\Keysight\TAP8\airplanemode_nocheck.TapPlan

YAML file
- File path: C:\Program Files\Keysight\TAP8\AalborgCalibration.yml

- ✓ 1x1
- ✓ 2x2

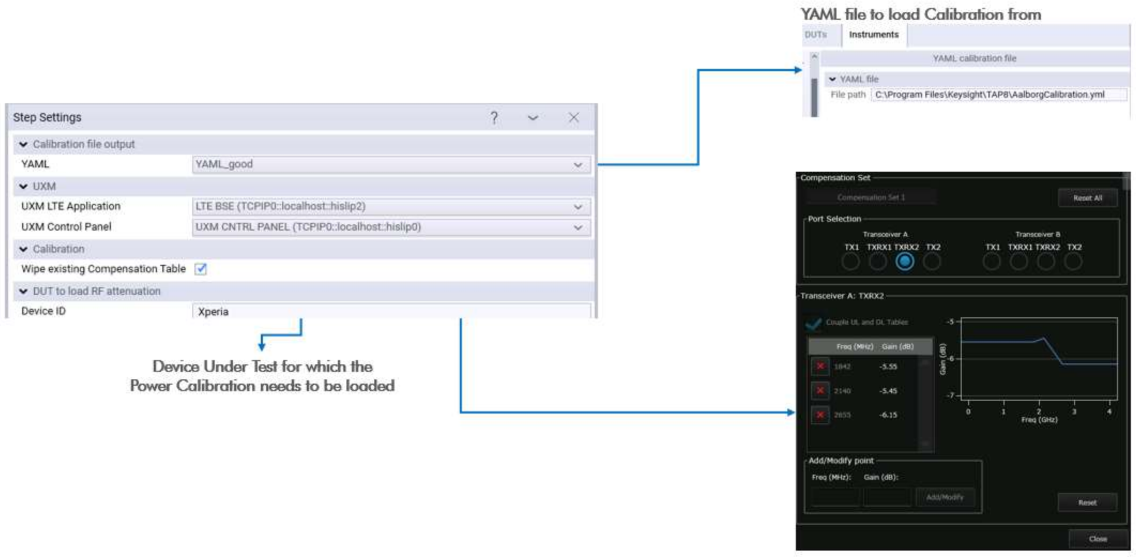
- ✓ Band 1
- ✓ Band 3
- ✓ Band 7
- ☐ Band 8
- ☐ Band 20

- ☐ Cell 1 on transceiver A
- ✓ Cell 2 on transceiver B

Figure 74 RF DL compensation plugin

Once the cabling loss characterization is performed for a device, at the beginning of a test campaign, a compensation of cabling loss test step is executed. This test step reads DL RF calibration from YAML file for a specific device under test, and then applies to UXM Control Panel the offset values for all previously calibrated connectors, bands, transceivers.

This approach guarantees identical received pilot power for all LTE devices in the testbed, which compensates for the soldered RF modifications of the devices.



Step Settings

Calibration file output
- YAML: YAML_good

UXM
- UXM LTE Application: LTE BSE (TCPIP0::localhost::hislip2)
- UXM Control Panel: UXM CNTRL PANEL (TCPIP0::localhost::hislip0)

Calibration
- Wipe existing Compensation Table: ✓

DUT to load RF attenuation
- Device ID: Xperia

YAML file
- File path: C:\Program Files\Keysight\TAP8\AalborgCalibration.yml

Transceiver A: TXRX2

Freq (MHz)	Gain (dB)
1842	-5.55
2140	-5.45
2655	-6.15

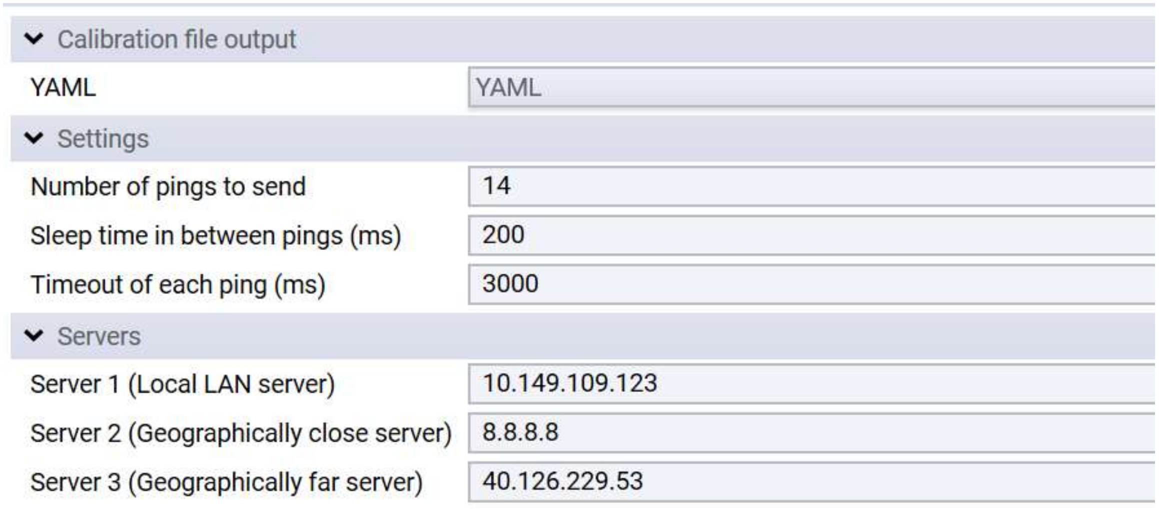
Figure 75 Offsetting the DL RF power values in UXM control panel

Testbed Latency

In order to characterize the latency that different devices under test (DUT) will experience in the testbed, the round trip time (RTT) needs to be measured at multiple levels, such as the delay to reach known servers from the private LAN containing the testbed, the latency in-between the different instruments and VMs within the TRIANGLE LAN, as well as additional latency brought in by TRIANGLE network impairments.

Latency can be evaluated from the perspective of a test PC (running TAP), as well as from the perspective of an Android DUT connected to the testbed.

To facilitate the measurement of the delay, a calibration TAP test has been developed and the corresponding TAP test steps and TAP plugins have been developed. Once the latency is measured, the values are saved in the Calibration YAML file.

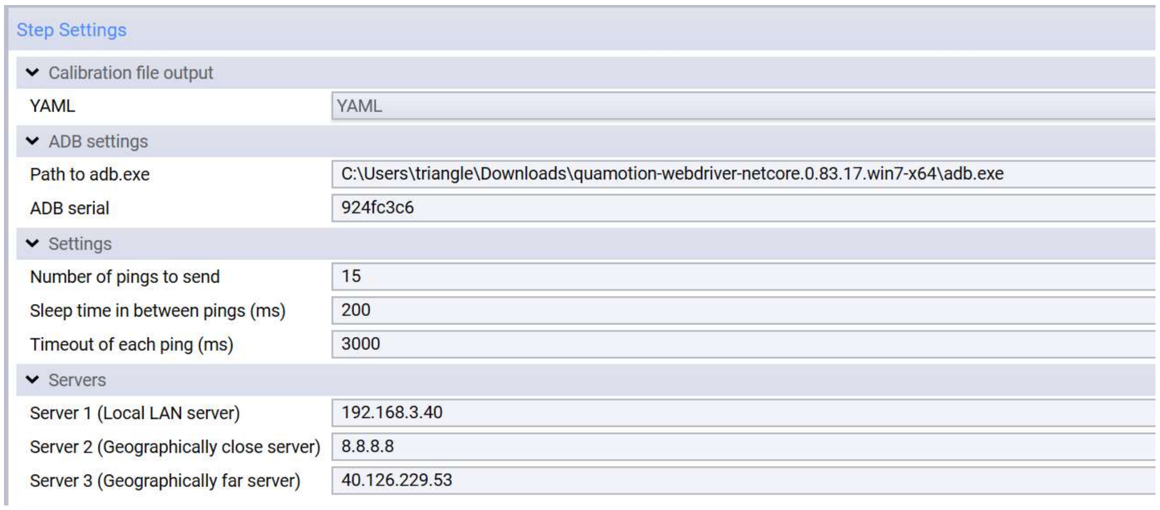


Calibration file output	
YAML	YAML
Settings	
Number of pings to send	14
Sleep time in between pings (ms)	200
Timeout of each ping (ms)	3000
Servers	
Server 1 (Local LAN server)	10.149.109.123
Server 2 (Geographically close server)	8.8.8.8
Server 3 (Geographically far server)	40.126.229.53

Figure 76 Step settings for the PC latency characterization test step

DUT Latency characterization

In addition to the testbed latency measurements described in the previous section, an additional latency measurement is performed from the perspective of the DUT. The UE-perceived latency may differ due to the IP stack on the device.



Step Settings

Calibration file output	
YAML	YAML
ADB settings	
Path to adb.exe	C:\Users\triangle\Downloads\quamotion-webdriver-netcore.0.83.17.win7-x64\adb.exe
ADB serial	924fc3c6
Settings	
Number of pings to send	15
Sleep time in between pings (ms)	200
Timeout of each ping (ms)	3000
Servers	
Server 1 (Local LAN server)	192.168.3.40
Server 2 (Geographically close server)	8.8.8.8
Server 3 (Geographically far server)	40.126.229.53

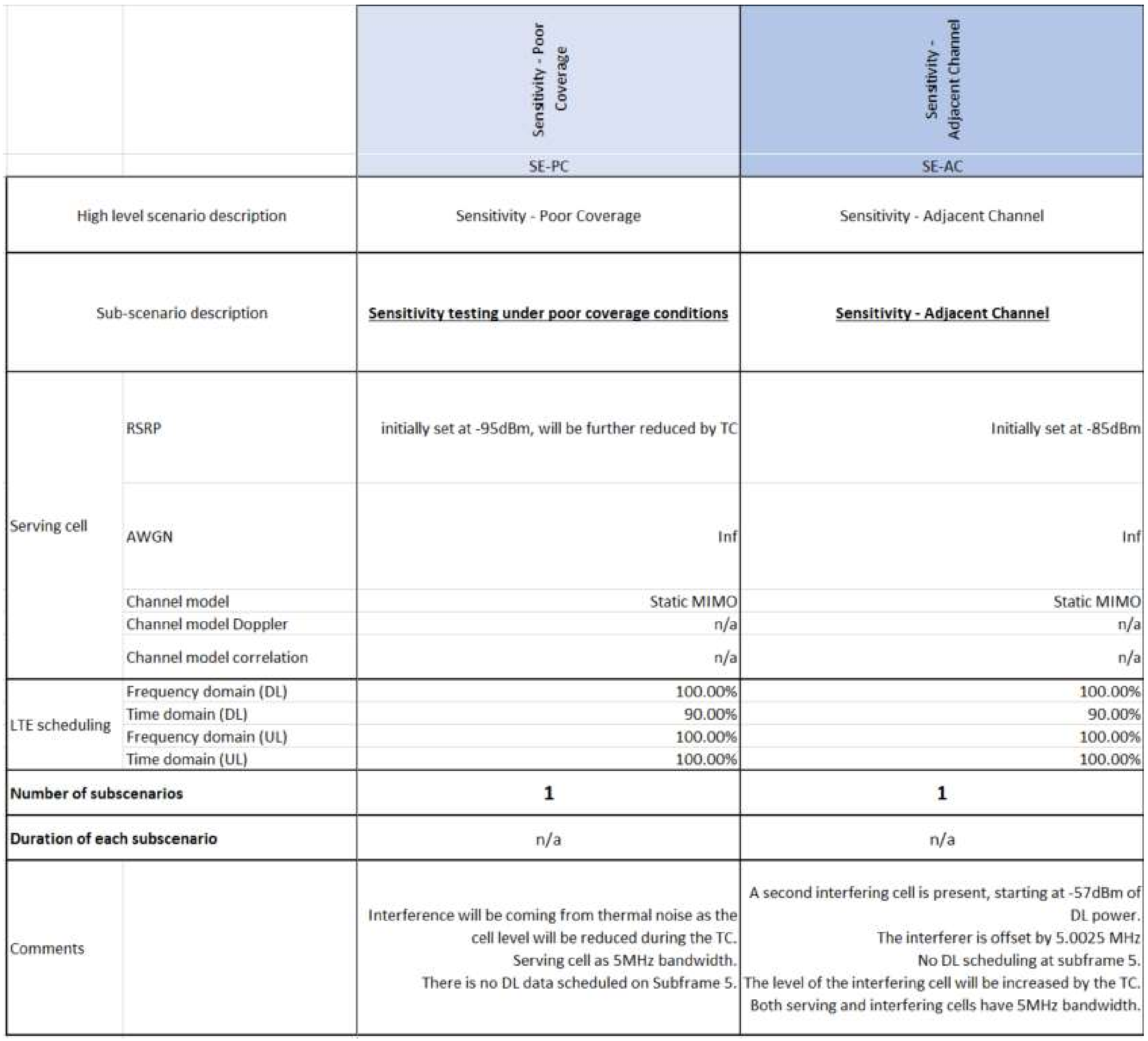
Figure 77 Step settings for the Android latency characterization test step

The DUT latency is measured pointing to the same servers as the testbed latency measurements. A series of pings are being sent from the UE to the servers and the average RTT calculated. Once measured, the UE latency results are saved in the Calibration YAML file. These UE latency values can then be used to offset additional latency values applied in network impairments, if a TRIANGLE experimenter wishes a fixed latency for his tests.

5.9 Network scenarios for Device Radio Performance domain

5.9.1 Scenario Definition

In order to support measurements of the new domain Device Radio Performance (RFP), two new network scenarios have been implemented in TAP and included in the testbed.



		Sensitivity - Poor Coverage	Sensitivity - Adjacent Channel
		SE-PC	SE-AC
High level scenario description		Sensitivity - Poor Coverage	Sensitivity - Adjacent Channel
Sub-scenario description		**Sensitivity testing under poor coverage conditions**	**Sensitivity - Adjacent Channel**
Serving cell	RSRP	initially set at -95dBm, will be further reduced by TC	Initially set at -85dBm
	AWGN	Inf	Inf
	Channel model	Static MIMO	Static MIMO
	Channel model Doppler	n/a	n/a
	Channel model correlation	n/a	n/a
LTE scheduling	Frequency domain (DL)	100.00%	100.00%
	Time domain (DL)	90.00%	90.00%
	Frequency domain (UL)	100.00%	100.00%
	Time domain (UL)	100.00%	100.00%
Number of subscenarios		**1**	**1**
Duration of each subscenario		n/a	n/a
Comments		Interference will be coming from thermal noise as the cell level will be reduced during the TC. Serving cell as 5MHz bandwidth. There is no DL data scheduled on Subframe 5.	A second interfering cell is present, starting at -57dBm of DL power. The interferer is offset by 5.0025 MHz No DL scheduling at subframe 5. The level of the interfering cell will be increased by the TC. Both serving and interfering cells have 5MHz bandwidth.

Figure 78 New network scenarios for RFP domain testing

These network scenarios focus on stressing the device's receiver sensitivity under background thermal noise for the Poor Coverage network scenario, and under neighbour cell interference for the Adjacent channel scenario.

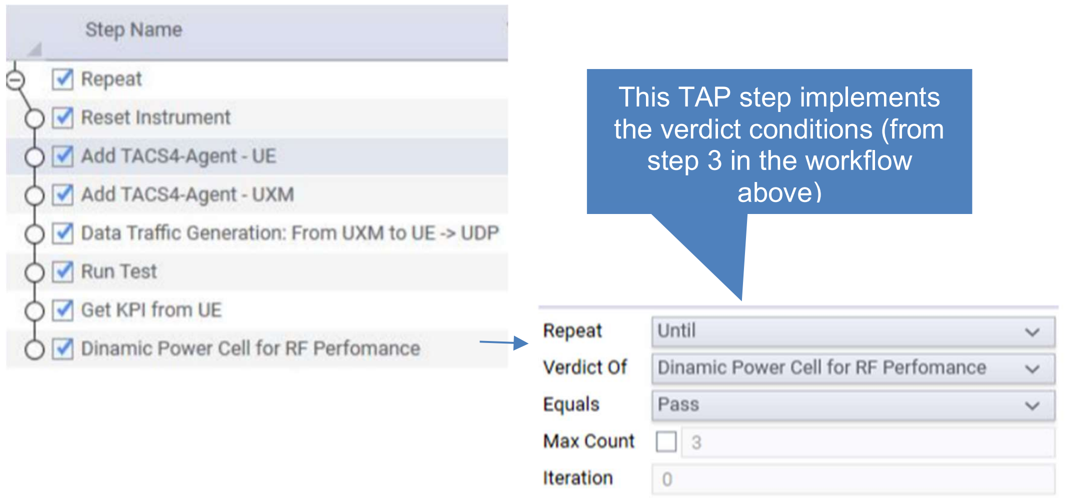
5.9.2 TAP template for sensitivity test cases

In order to implement the RFP test cases RFP*/HS/001* (Sensitivity) and RFP*/HS/002* (Adjacent Channel Selectivity) a sequential TAP template has been implemented. In a sequential template there is no parallel execution of other measurements (such as the DEKRA Performance Tool) as done in other test plans templates. This should not be surprising as sensitivity and adjancent channel selectivity is independent of the rest of the components.

The RFP test cases workflow is as follows:

1. Start Loop
2. @RAN Emulator: Set Network Scenario
    A. Sensitivity Test Case: Set serving cell transmission power level
    B. Adjacen Channel Test Case: Set adjacent cell transmission power level
3. @Performance Tool: Measure average throughput for 30 seconds
4. @TAP: Evaluation of test case conditions:
    A. If the throughput is above 1.2 x *Target value*, the power level is decreased in 1 dB
    B. If the throughput is equal or below 1.2 x *Target value*, the power level is decreased in 0.2 dB
    C. If the throughput is below the Target vale, the loop is broken and the tets case ends
5. Wait for 10 seconds and jump to step 2.

Figure 79 shows the TAP template which implements the RFP test cases:


Step Name
This TAP step implements the verdict conditions (from step 3 in the workflow above)

Repeat
Reset Instrument
Add TACS4-Agent - UE
Add TACS4-Agent - UXM
Data Traffic Generation: From UXM to UE -> UDP
Run Test
Get KPI from UE
Dinamic Power Cell for RF Perfomance

Repeat	Until
Verdict Of	Dinamic Power Cell for RF Perfomance
Equals	Pass
Max Count	☐ 3
Iteration	0

Figure 79 TAP Template for RFP Test Cases

The DEKRA Performance Tool TAP plugin has also been upgraded to report a scalar measurement for the DUT average throughput. Note that in other templates, the throughput is reported as a vector which contains samples for each time record and it is the ETL which aggregates that vector to provide averages and other statistics (e.g. for the TRIANGLE Mark computation). However, in the RFP test case that average throughput is needed in the test case at runtime in order to evaluate the verdict conditions and set the right transmission power value in the network scenario.

Table 34 shows a sample result for *RFP/HS/001* (Sensitivity) test case.

Table 34: Validation test result RFP/HS/001

Parameter	Value
Initial Throughput	6 Mbit/s
Target Throughput	1 Mbit/s
Sensitivity (KPI)	-120 dBm
Loop count	40
Test Case Iteration Duration	600 s

6 Radio Access Network (RAN)

This section describes the elements that compose the RAN of the TRIANGLE testbed.

6.1 eNodeB Emulator

The TRIANGLE testbed has integrated the UXM Wireless Test Set from Keysight as the radio access network (RAN) of the testbed. The UXM is a flagship mobile network emulator that provides state of the art test features.

The Keysight E7515A UXM Wireless Test set is capable of emulating a 2G, 3G, 4G and NB-IOT base stations. The UXM is a highly-integrated instrument created for functional and RF design validation in 4G and NB-IOT. It provides the integrated capabilities needed to test the newest designs, delivering LTE-Advanced Pro data rates up to 1 Gbps. In addition, the UXM allows functional test by emulating a wide range of complex network operations, such as LTE intra-RAT and LTE inter-RAT mobility, WLAN offload and end to end VoLTE. The UXM also provides protocol messaging.

The number of features and capabilities of the UXM is very extensive. The purpose of this section is to highlight its most relevant features for the testbed. For a complete reference of capabilities and documentation about such capabilities, the reader is referred to [9].

Feature	**Description**
Basic Cell Configuration	The UXM provides an API to control most of the network parameters. Among the basic aspects that can be configured, the following are highlighted: - Duplex mode: FDD or TDD - Downlink and Uplink bandwidth - Most of the FDD and TDD LTE bands are supported - TDD frame configuration
RF Channel Conditions	The UXM is capable of emulating different channel conditions in terms of signal levels, fading profiles, and noise and interference profiles. The following propagation conditions are supported: - Static - Extended Pedestrian A (EPA) - Extended Vehicular A (EVA) - Extended Typical Urban Model

Mobility	The UXM is capable of emulating the following mobility scenarios: - Intra-System Handover - Inter-System Handover - Roaming
Cell Load	The UXM is capable of emulating different cell loads by controlling the number of resources allocated to the user.
Dual Connectivity	Within the scope of the TRIANGLE project the Testing framework shall be able to support dual connectivity between 3GPP radio access nodes and between 3GPP and non-3GPP radio access nodes.
Supported Formats	GSM/UMTS/LTE/LTE-A/NBIOT

6.2 RF switches

TRIANGLE offers several reference devices in the testbed. TRIANGLE users may select a given device at any moment. Therefore, these devices should be available to the testbed at all times. This poses a considerable challenge as the RAN emulator only has 4 RF connectors. Obviously, wiring all the devices to the RF ports of the RAN emulator could create instabilities, e.g., one rogue device connecting to the network instead of the desired one. To avoid such issues and for the sake of a more stable system TRIANGLE uses an RF switch. This switch is capable of creating a 1 to 1 mapping, meaning that at a given time only a device is connected to the RAN emulator.

TRIANGLE is using the Keysight L7104A component, which is an electro-mechanical switch providing isolation and 0.03 dB insertion loss repeatability. The RF switch is shown in Figure 80.

Figure 80 Keysight RF Switch Product Number L7104A

The RF switch is controlled by a LXI-compliant 11713C attenuator/switch driver, which provides remote or front-panel drive control. This controller provides means to programmatically establish wired connections with the desired device under test. The controller is shown in Figure 81.

Figure 81 Keysight 11713C LXI-Compliant Attenuator/Switch Driver

7 Evolved Packet Core (EPC)

The core network available in the TRIANGLE testbed is provided by Polaris Networks. It is a core network emulator which provides carrier grade performance for up to 2000 clients. The platform supports positive and negative behaviours, remote sniffing of all the interfaces of the network elements, introduction of traffic impairments in the transport interfaces and support for multiple instances of the following EPC entities:

- MME
- SGW
- PGW
- PCRF
- HSS
- ePDG
- ANDSF

The configuration and creation of these components can be done by means of a GUI interface, accessing a programmable API written in TCL, and partially with a C++ API developed in the FLEX project.

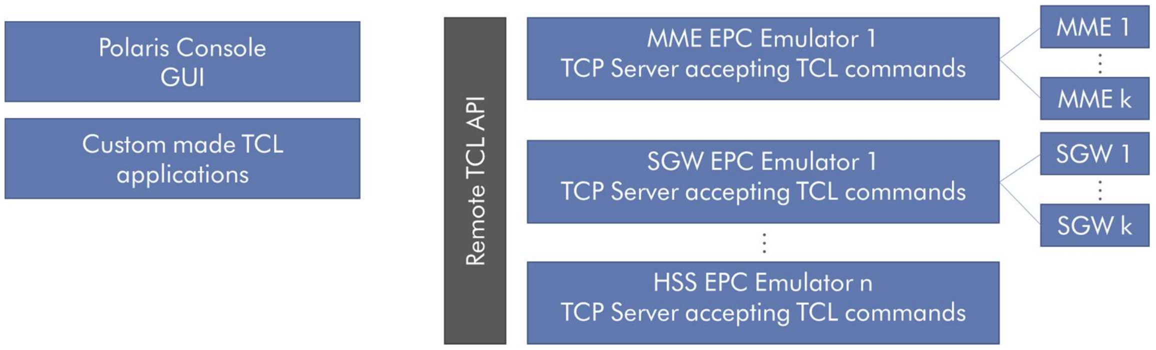

Polaris Console GUI
Custom made TCL applications
Remote TCL API
MME EPC Emulator 1 TCP Server accepting TCL commands
SGW EPC Emulator 1 TCP Server accepting TCL commands
HSS EPC Emulator n TCP Server accepting TCL commands
MME 1
MME k
SGW 1
SGW k

Figure 82 Polaris Networks EPC GUI and TCL API

7.1 Features and Configurable Parameters

The following table depicts the list of available features and configurable parameters by the user of the entities that compose the Polaris Networks EPC emulator.

Table 35 MME configuration parameters

MME Features	**Details**
Standard Interfaces	S1-MME, S11, S10, S6a, S13, S3, SBc, SGs, S102, M3, Sm
Protocols/Interfaces with Configurable Behaviour	S1 Setup, S1AP, NAS, GTv2, SGsAP, Diameter
Procedures	Attach, Paging, SMS, Detach, Location Report, Handover, etc.
Capabilities	Partial Path Failure, PGW Restart Notification, Modify Access Bearer Request, Network Triggered Service Restoration, Relay eNB, IMS

	Voice over PS session, Diameter Proxy Agent Type 2, Extensive Diameter Validation, IPv6/IPv4, Roaming, Configuration of protocol policies.
S1AP Procedures with Configurable Behaviour	Setup Request, Reset, eNB Configuration Update, eNB Configuration Transfer, Initial UE Message, Uplink NAS Transport, UE Context Release Request, Handover Required, Handover Notify, Handover Cancel, Path Switch Request, Initial Context Setup Response, Handover Request ACK, E-RAB Setup Response, E-RAB Modify Response, E-RAB Release Response, UE Context Modification Response
NAS (EMM and ESM) Procedures with Configurable Behaviour	Attach Request/Complete, Detach Request/Accept, TA Update Request/Complete, Security Mode Complete/Reject, Service Request, Authentication Response/Failure, Activate default/dedicated EPS bearer accept/reject, Modify EPS bearer context accept/reject, PDN connectivity request
GTPv2c Procedures with Configurable Behaviour	Create/Update/Delete bearer request, identification request/response, context request/response/acknowledge, MBMS session start/update/stop,
SGsAP Procedures with Configurable Behaviour	SGsAP location update ack/reject, SGsAP paging request, SGsAP release request, SGsAP alert request
Diameter Procedures with Configurable Behaviour	Cancel-location-request, insert/delete-subscriber-data-request, reset-request

Table 36 PGW configuration parameter

PGW Features	**Details**
Standard Interfaces	S5-c, S5-u, S8-c, S8-u, SGi, S2a, S6b, Gx
Protocols/Interfaces with Configurable Behaviour	GTPv2-C, PMIPv6, Diameter, GTPv1-U
Functionalities	Traffic generation, traces, apn, protocol configuration
GTPv2C Procedures with Configurable Behaviour	Create Session Request, Delete Session Request, Modify Bearer Request, Modify Bearer Command, Delete Bearer Command, Bearer Resource Command
PMIPv6 Procedures with Configurable Behaviour	Proxy Binding Update, Binding Revocation, Heart Beat
Diameter Procedures with Configurable Behaviour	Re-Auth Request, Session Termination Request
GTPv1U Procedures with Configurable Behaviour	QCI demands.
PGW Features	Details
Standard Interfaces	S5-c, S5-u, S8-c, S8-u, SGi, S2a, S6b, Gx

Table 37 PCRF configuration parameters

PCRF Features	**Details**
Standard Interfaces	Gx, Rx, S9, Gxx

Functionalities	Services Creation/Modification/Deletion, IMS, Roaming
Diameter Procedures with Configurable Behaviour	Authorisation and Authentication Request, Credit Control Request, Session Termination Request

Table 38 SGW configuration parameters

SGW Features	**Details**
Standard Interfaces	S11/S4-c, S1/S4/S12-u, S5/S8-c,S5/S8-u, Gxc
Protocols/Interfaces with Configurable Behaviour	GTPv2-c, PMIPv6, GTPv1-U
Capabilities	Partial Path Failure, PGW Restart Notification, Modify Access Bearer Request, Network Triggered Service Restoration, Diameter Proxy Agent Type 2
GTPv2C Procedures with Configurable Behaviour	Create/Delete/Modify/Update Session Request, Release/Modify Access Bearer Request, Create Indirect Data Forwarding Tunnel Request
PMIPv6 Procedures with Configurable Behaviour	Binding Revocation, Heart Beat
GTPv1U Procedures with Configurable Behaviour	QCI demands.

Table 39 HSS configuration parameters

HSS Features	**Details**
Standard Interfaces	S6a, Zh, Cx
AAA interfaces	S6b, STa, SWd, SWm
SPR Functionality	Service definition (Service data flow, piggybacked bearer creation, QCI, Priority Preemption, UL-GBR, UL-MBR, DL-MBR, DL-GBR), AMBR, GBR, Charging, QCI, Priority,
Subscribers	Subscription groups, Subscription Profiles
Diameter Procedures with Configurable Behaviour	User-Authorisation-Request, Server-Assignment-Request, Location-Info-Request, Multimedia-Auth-Request, Authentication-Information-Request, Update-Location-Request, Purge-UE-Request, Notify-Request
HSS Features	Details

Table 40 ePDG configuration parameters

ePDG Features	**Details**
Standard Interfaces	SWu, S2b, SWm
Protocols configuration	IKEv2, PMIP, Diameter
IKEv2 Configurable Behaviour	IKE_SA_INIT, IKE_AUTH, INFORMATIONAL

PMIPv6 Behaviour	*Configurable*	Binding Revocation, Heart Beat
Diameter Behaviour	*Configurable*	Re-Auth Request, Abort Session Request

Table 41 ANDSF configuration parameters

ANDSF Features	**Details**
Standard Interfaces	S14, Zh, Ub
Rules	ISRP Rules (flow based, service based, non-seamless offload rules), ISMP Rules (access networks, time of day conditions, validity areas)
Non 3GPP Access	WLAN, 3GPP2, WiMax, Geo.
ANDSF Features	Details
Standard Interfaces	S14, Zh, Ub

7.2 Measurement and Behaviour

The EPC deployment can provide the signalling available in between all the components of the network. This feature can be supported by directly sniffing in the interfaces but also with an automatic system provided by the Minimization of Drive Test features (see TS 32.422) that can send the messages in XML format to an external server.

Additionally, the EPC modules themselves can provide statistics regarding the procedures associated with each protocol. All the EPC modules provides information on the received messages as well as failure/success counts for each procedure.

Table 42 HSS Statistics

Interface	**Procedures**
S6a/S6d	Update location, canel location, authentication information retrieval, insert subscriber data, delete subscriber data, purge ue, reset, notify and ME identify check.
Cx	User authorization, authentication information retrieval, server assignment, user location information retrieval, registration termination, push profile.
SLh	Routing info.
Sh	User data, profile update, subscribe notifications.
S6b	Authorization, abort session, session termination.
STa and SWm	Authentication Authorization, abort session, session termination.

In the case of the MME the system also provides statistics on the subscribers in the different mobility states and number of connections in the different interfaces. In the case of general procedures there is also some information regarding the minimum, maximum and average time to finish some procedures such as attach, detach, tracking area update, S1-handover, X2-handover, service request, paging, pdn connection, dedicated bearer activation, etc.

Table 43 MME Statistics

Interface	Procedures
NAS	Attach, network/UE initiated detach, security, service request, TAU, ESM information, GUTI reallocation, etc.
S1AP	S1 Setup/Reset, eNB configuration update, MME configuration update, initial context setup, UE context modify, UE context release, e-RAB setup, e-RAB modify, e-RAB release, HO preparation, etc.
GTPv2-C	Create session, delete session, modify bearer, release access bearer, delete bearer, create/delete indirect data forwarding tunnel, get identification, create context, forward relocation, forward relocation complete, etc.
Diameter	Update/cancel location, authentication information, insert/delete subscriber data, purge UE, reset, notify, ME identity check, mobile terminated location report.
M3AP	M3 setup, session start/stop.
LCSAP	Location service, reset.
SBcAP	Write replace warning, stop warning.
SGsAP	Paging for non-EPS services, location update for non-EPS services, Non-EPS alter, IMSI Detach from EPS/non-EPS services, VLR/MME/HSS failure, MME information, tunnelling of NAS messages, service request.
S102AP	S102 session establishment/termination, S102 tunnel redirection

The PCRF also provides some timing statistics for procedures such as IP-CAN session establishment/termination, service activation/modification/deletion, S9 sub-session establishment/termination, AF session establishment/termination, gateway control session establishment/termination.

Table 44 PCRF Statistics

Interface	Procedures
Gx, Gxx and S9	Credit control, re-authorization
Rx	Authentication authorization, re-authorization, abort session, session termination.

The PGW provide timing stats for PDN connection/disconnection and dedicated bearer activation/modification/deactivation and also statistics regarding the throughput and number of packets in the S5/S8 and S2a/S2b interfaces per user and RAB id.

Table 45 PGW Statistics

Interface	Procedures
GTPv2-C	Session creation/deletion, bearer modification, UE requested bearer operation, network initiated dedicated bearer creation/update/deletion.
Gx	Credit control, re-authorization
S6b	Authentication authorization, re-authorization, abort session, session termination.
PMIPv6-C	Create/delete session.

The SGW provides information on the data traffic per user and per bearer in the S1-U/S4-U/S12-U and S5-U/S8-U interfaces. It also provides information on the control packets of the data plane such as echo request/response, error indication, SEH notification or end packet.

Table 46 SGW Statistics

Interface	Procedures
GTPv2-C	Create/Delete session, modify bearer, release access bearer, downlink data notification, create/delete indirect data forwarding tunnel, create/update/delete bearer, change notification, update/delete PDN connection set, modify access bearer, PGW restart notification.
PMIPv6-C	Create session/delete session.
Diameter	Credit control, re-authorization.

And the testbed can also provide sniffed packets for all the interfaces that interconnect the core network both internally and externally.

7.3 TRIANGLE EPC Plugin

The TRIANGLE testbed can deploy automatically a functional EPC to be used in the different experiments by means of the “EPC plugin”, implemented by TRIANGLE project. This plugin allows users to created fixed topology scenarios (regular deployments of MME, SGW, PGW, PCRF, HSS and ePDG). The standard architecture that is deployed with the EPC plugin is depicted in the following figure:

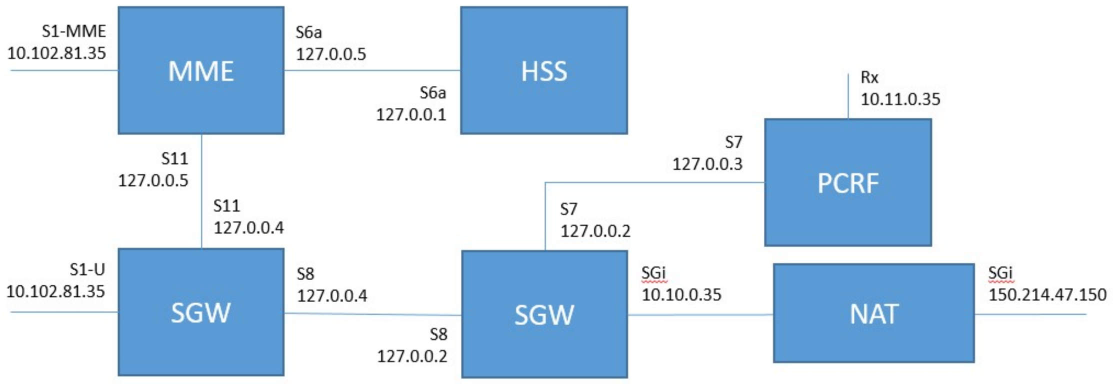


Figure 83 EPC architecture

The interfaces which are not shown in the figure are setup in different loopback interfaces as per the following table.

Table 47 EPC Elements loopback IPs and not depicted interfaces

Element	IP loopback	Interfaces not depicted
MME	127.0.0.5	S10, M3,S13, S3, SBc, SGs, S102, Sm, SLg, SLs, Sv
SGW	127.0.0.4	Gxc
PGW	127.0.0.2	S5, S2A, S2B, Gx
HSS	127.0.0.1	Zh, Cx, SLh, Sh, STa, SWd, SWm
PCRF	127.0.0.3	Gx, S9, Sxx

The configuration can be adapted to any user willing to test any particular interface or component in the network.

Besides automatically deploying the EPC, the EPC plugin is also able to trigger certain procedures on the network:

- MME Detach IMSI, triggers a detach procedure for a given IMSI. The detach message will be of type 1, cause 0 and indicate that a reattach is required. This is useful to obtain multiple attach samples in order to analyse both the behaviour and time consumed by the procedure.
- MME UE Context Release IMSI, which triggers a UE context release with cause group 3 and cause value 0.
- MME Paging IMSI, tells the MME to initiate a paging procedure for the given IMSI.
- PCRF Create Dedicated Bearer. The command will create a dedicated radio bearer matching a service with the following parameters (that have to be provided by the user):
    - IMSI
    - UE IP
    - QCI (Quality Class Indicator)
    - Maximum Bit Rate for uplink and downlink

    - Guaranteed Bit Rate for uplink and downlink.
- PCRF Release Dedicated Bearer. The command will release a dedicated bearer matching the following parameters:
    - IMSI
    - UE IP
    - QCI

More details on the implementation of the system are provided in D4.1 and in the internal deliverable “EPC SCPI Server”.

8 Transport

8.1 SDN

The transport layer consists of three independent network domains interconnected through a virtualized routing environment and managed by several SDN components. On the one hand this approach allows to quickly test new configurations without the time consuming task of routing new physical lines between the networks and, on the other hand, it does not limit the future expansion of the test facilities as the virtualized network equipment can connect to their physical counterpart, and thus any component can be moved to another location and only those routers and switches on the edge have to be made aware of the new addresses. Figure 84 shows the three domains considered owned by two different actors, the mobile operator and the backhaul operator:

- RAN or Access network of the mobile operator, which includes the base stations and the network equipment.
- The EPC of the mobile network.
- The distribution network between the two above.

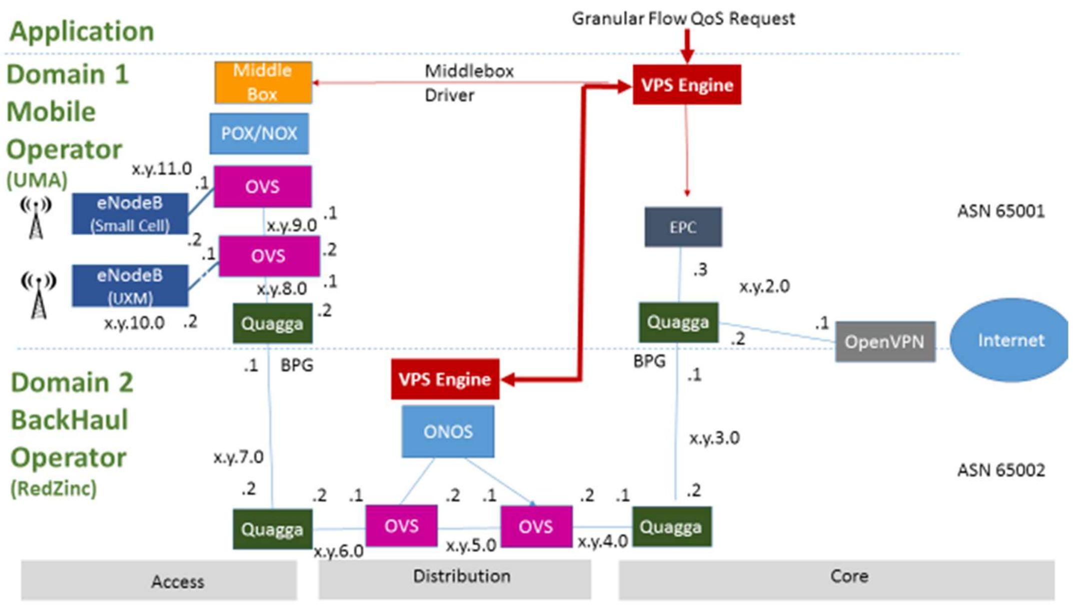


Figure 84 Software defined network deployment at TRIANGLE testbed

The devices inside each domain are isolated from each other, effectively belonging to different networks. The interconnection is provided by the following three components:

- OpenvSwitch, a fully compliant OpenFlow implementation for virtual switches, designed to be run in a virtualized environment but that can also be used with real network equipment.
- Quagga, a software router implementing several routing and discovery algorithms as well as advanced network functionality like multipath Border Gateway Protocol (BGP) or Intermediate-System to Intermediate-System (IS-IS) routing.

- ONOS, a network orchestrator or SDN controller which provides a transparent interface to monitor and manage a distributed deployment in different subnetworks. Different instances running in different networks can be interconnected to present a unified interface to control and visualize all the resources and traffic flows from a central location. It also provides a high level API to inject new rules for specific links based on the source or target address, or the type of traffic.

The initial deployment consists on Virtual Machines hosted on a Linux KVM hypervisor environment, with isolated virtual networks configured in the host machine. The different subnets are not linked with each other and the host's only role is to provide a gateway to the public internet for the VMs, but it cannot be used to break that isolation. The way to interconnect the networks is the equivalent of a real environment: each router or switch needs an additional port (i.e. a virtualized Ethernet device) for every network it belongs to.

The low-level configuration of each VM, that is, the IPs and the number of networks it connects to, is expected to remain stable for the duration of the project so it has been made by hand. The configuration of the transport network is one of the cornerstones of the setup, so it has to allow automatization.

Once deployed, the Quagga routers should be able to work without direct intervention from the operator. As it implements discovery algorithms like BGP and OSPF it will detect other routers in the networks it is connected to and will establish and maintain the routes between the hosts in each network.

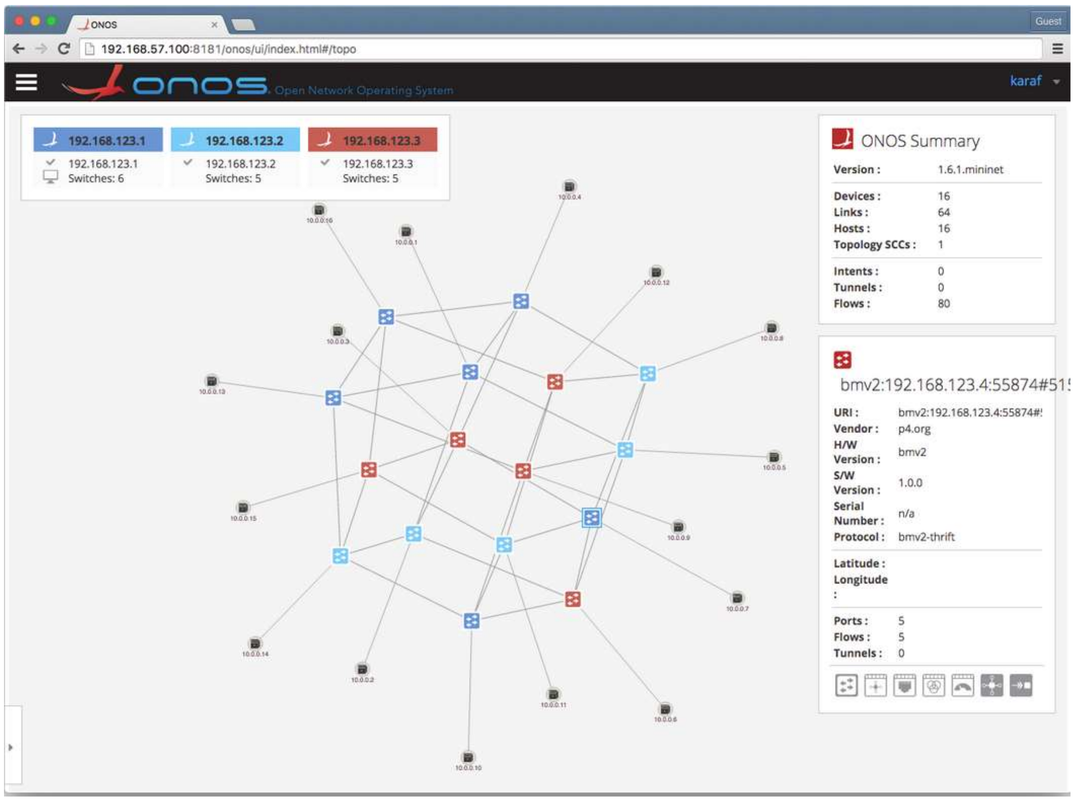
OpenvSwitch works out-of-the-box as a learning switch which makes a discovery search using the ARP protocol when a new host sends packets through it, but it also has advanced features that can be configured using the OpenFlow protocol. Some of the switching capabilities expected to be used in the project are:

- Creation of data plane and control plane as differentiated flows in each switch. It will allow to process both types of traffic with different priorities or even through different routes.
- User traffic identification and matching against rules to provide different QoS to the data coming from the UE.
- Transparent mirroring of certain traffic flows to allow advanced interconnection schemas between the EPC and groups of users.

OpenvSwitch can be configured by command line (or by means of a script) specifying the flow rules to be installed or modified, but this approach can be cumbersome in a dynamic environment as it would be necessary to create the commands on the fly.

Another way to configure the switch is to set up a *controller node* in each OvS instance. This way, when a packet without a matching rule arrives to the switch, it will ask that controller what actions it should perform for that packets and whatever follows in the same flow.

The ONOS orchestrator will take the role of the switch controller, as it provides a high-level Java API to inject OpenFlow rules as a result of a petition from any switch or router associated with it. It also provides a web interface to monitor the network and to install and remove rules for different instances of the switches and routers, as can be seen in Figure 85.



ONOS Summary

Version :	1.6.1.mininet
Devices :	16
Links :	64
Hosts :	16
Topology SCCs :	1
Intents :	0
Tunnels :	0
Flows :	80

bmv2:192.168.123.4:55874#51

URI :	bmv2:192.168.123.4:55874#
Vendor :	p4.org
H/W Version :	bmv2
S/W Version :	1.0.0
Serial Number :	n/a
Protocol :	bmv2-thrift
Latitude :	
Longitude :	
Ports :	5
Flows :	5
Tunnels :	0

Figure 85 ONOS Interface

8.2 Emulated impairments

The testbed will offer the possibility of integrating artificial impairments in the interfaces of the core network and the application servers. To do so, two different alternatives are being explored: impairments introduced by the EPC emulator and impairments introduced by external applications.

The impairments introduced by the EPC emulator can be set in any of the interfaces of each of the components of the EPC and it enables the definition of the following impairments:

- Limit of packets, which is the number of packets to which the impairment will be applied. If the limit is set to 0, it will be applied to all the packets.
- Delay Parameters, which includes, delay, jitter, delay correlation, delay statistical distribution, percentage or reordered packets, percentage of reorder correlation and gap.
- Loss Parameters, distribution type, percentage of lost packets, percentage of correlation.
- Corruption and Duplication, percentage and correlation for both.

To access this functionality EPC emulator offers a graphical interface and a TCL script.

The other possibility to integrate emulated impairments in interfaces not related to the EPC is the use of external applications. The main explorations by the TRIANGLE project have been the use of mininet and dummynet [4] (the userspace version is able to process 6 million packets per second with simple filtering). The main issue with these types of approaches is the performance that can be offered in real environments, in the preliminary test carried out by the UMA team mininet can offer up to 100 Mbps when connected to equipment outside the emulation environment, and the integration can be done using command line parameters. There are still ongoing tests efforts with dummynet, and the integration with the testbed will be done based on command line parameters.

8.3 Virtual Path Slice Engine

RedZinc's VPS Engine, VELOX, provides an API for 3rd party application developers so that services between two endpoints with a specified bandwidth reservation can be requested without the specific knowledge of how the network itself is deployed or how many Autonomous Systems need to be involved in order to establish the service. The API also allow the listing of running services and services available for request. Unique API keys are generated on demand and bound to developers so that all services can be correctly charged, adding the capability to simulate a financial layer in the testbed.

8.3.1 VPS engine usage

In order to use the VELOX API an application must:

- Create a TCP connection to known IP address/port (provided by local operator)
- Write Request (as a single text line, new line ends a request)
- Read Response (sent as a single line)
- Connections are terminated on the VELOX side after sending the response

All Requests must use the API key generated by the local operator VELOX.

8.3.2 Usage considerations

API Key

The API Key provided will always be a standard UUID in human readable format without dashes.

Example: ECE335024E3E466CA98BF5014D5C7D86

IPv4 vs IPv6

VELOX Supports both IPv4 and IPv6 services, but does not allow IPv4 mixed with IPv6, in requests that have both source and destination addresses, both must be of the same IP version. All versions of IPv6 abbreviation are supported.

Security

Currently the system considers a safe connection already exists between the Operator and the 3rd Party.

8.3.3 Usage example

In this scenario the client application is considered to be any application that uses the VELOX API to access VELOX services.

Modify and Run requests are considered optional since not all circumstances will require their use.

While the List request is the first to be executed it is not mandatory before every Trigger request, the use of the List request should be done as deemed necessary in order to check for any service library changes.

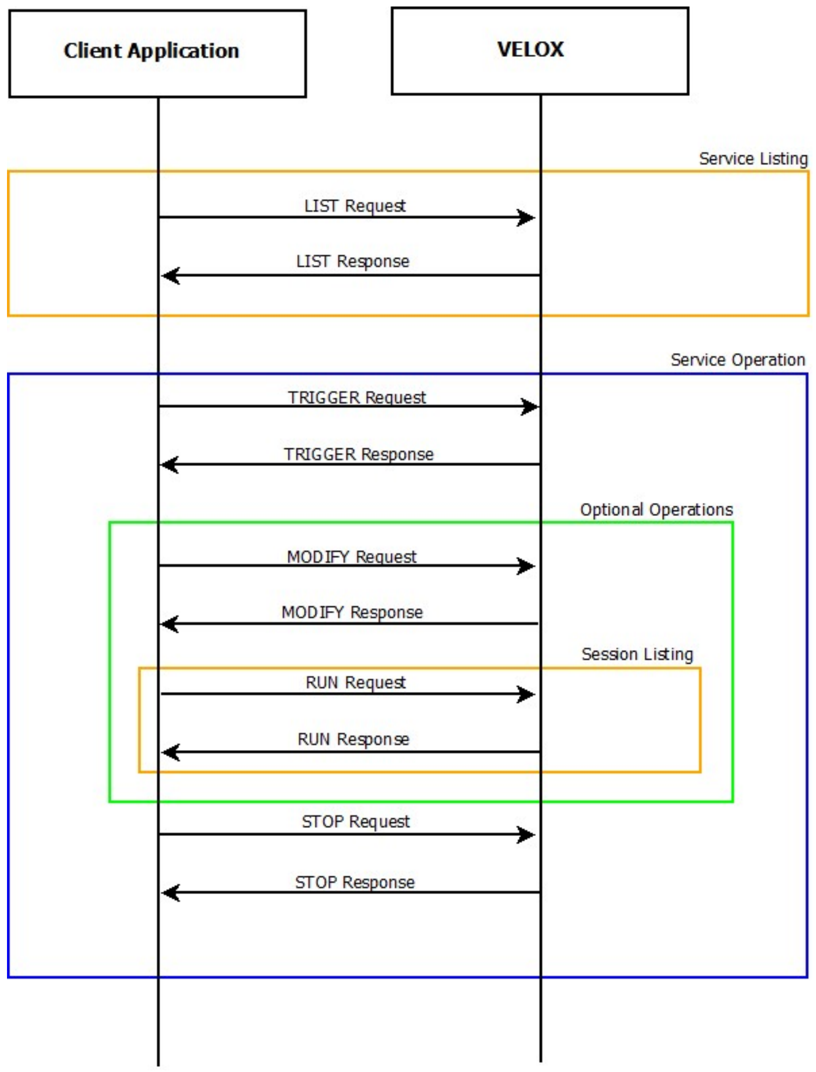


Figure 86 VPS engine usage example

A detailed description of VPS engine API is provided in Appendix 3.

8.4 TRIANGLE Configurable Traffic Shaping API

This section highlights a new functionality added in the Release 4 of the TRIANGLE testbed, which offers to the end-user a new traffic filtering and shaping API and introduces packets scheduling conditions similar to live networks, especially for heavy downlink-starved traffic models.

Typically, in the TRIANGLE testbed, the packet flow between an application content server and the UE experiences a bottleneck in the LTE DL connection between the UXM and the UE. This is by design, caused by the network scenarios, to represent other cellular users as well as realistic propagation conditions. When the LTE link is the bottleneck, it leads the UXM to buffer packets in downlink, which can unfortunately lead to very high one-way delays. If the application server keeps streaming data (notably in UDP type of traffic) and the LTE link's capacity does nott increase, the downlink packets will stay buffered until the end of the application flow, which impacts downlink reception delay.

The new API makes it possible to:

- apply a filter on a certain type of traffic (very flexible filtering, can capture any kind of protocol)
- traffic-shape the filtered traffic to guarantee latency at the expense of packet loss
- leave all non-filtered traffic untouched (this can be used for the control path)

A standalone executable is also provided which leverages this API and controls the data flow in a way to guarantee at maximum 500ms of one-way delay yet keeps the LTE link capacity saturated to avoid capacity loss.

8.4.1 Implementation Details

Location in the Testbed

The Impairments Python Server (originally developed by UMA) has been extended with a new API to control a new type of impairments:

Traffic Shaping

- A set of packet filters and bucket filters can be added at the EPC network interface
- Controlled via new custom SCPI commands
  TRA:DEF:TSH:CLN
  TRA:DEF:TSH:ENA 1,192.168.3.11
  TRA:DEF:TSH:PAR 10mbit,500ms,10kb

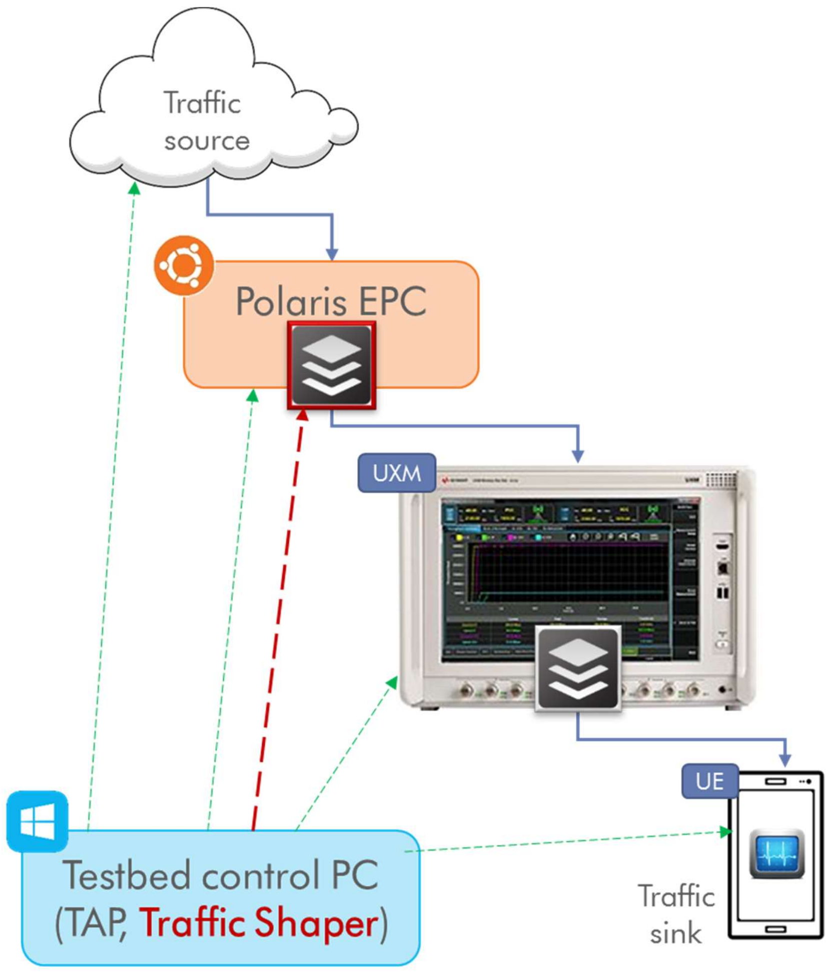


Figure 87 Location of the Traffic Shaper within the testbed

To take advantage of the new API to control the traffic, a new executable is running in the background on a Windows machine

- Periodically sends pings to the UE
- Estimates the UXM buffer status and LTE link capacity
- Sends SCPI commands to the impairments server to adjust the flow parameters of the buffer at the EPC

The API can also be leveraged in a TAP plugin, or exposed to experimenters.

Linux Filters

A new virtual network adapter is configured to receive traffic from the physical adapter, and perform filtering and traffic shaping.

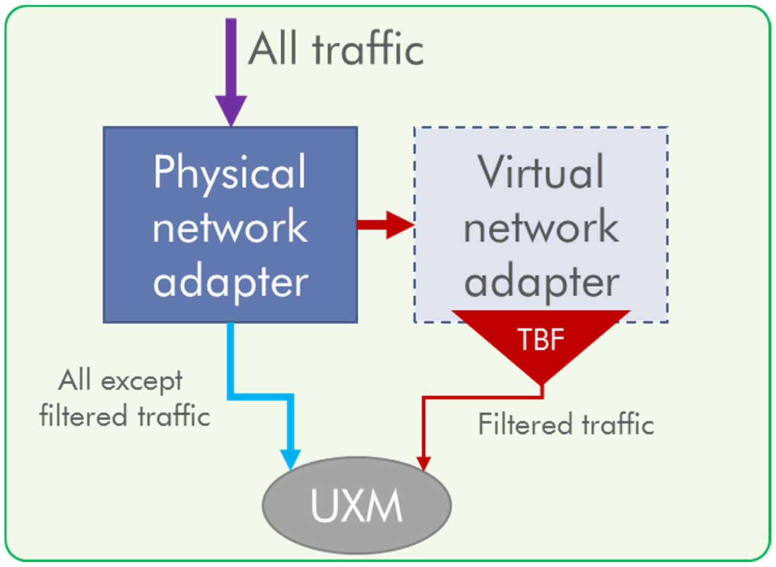
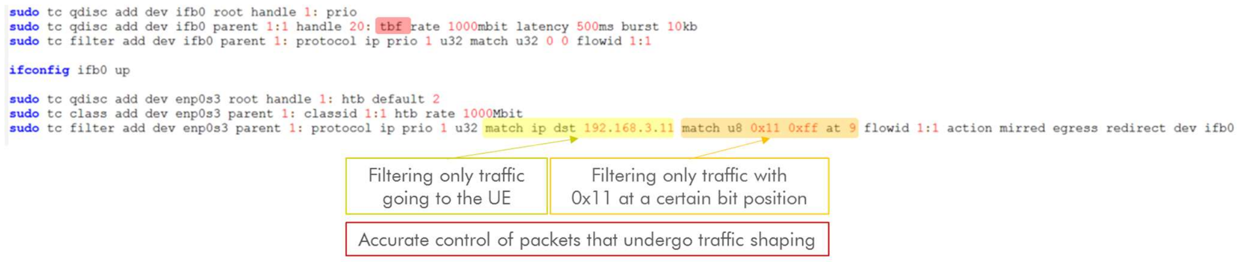


Figure 88 Virtual network adapter with Linux packet filters

The commands below are being sent by the API to initialize the filters:



```
sudo tc qdisc add dev ifb0 root handle 1: prio
sudo tc qdisc add dev ifb0 parent 1:1 handle 20: tbf rate 1000mbit latency 500ms burst 10kb
sudo tc filter add dev ifb0 parent 1: protocol ip prio 1 u32 match u32 0 0 flowid 1:1

ifconfig ifb0 up

sudo tc qdisc add dev enp0s3 root handle 1: htb default 2
sudo tc class add dev enp0s3 parent 1: classid 1:1 htb rate 1000Mbit
sudo tc filter add dev enp0s3 parent 1: protocol ip prio 1 u32 match ip dst 192.168.3.11 match u8 0x11 0xff at 9 flowid 1:1 action mirred egress redirect dev ifb0
```

Filtering only traffic going to the UE
Filtering only traffic with 0x11 at a certain bit position
Accurate control of packets that undergo traffic shaping

Figure 89 Configuration of the Linux network interfaces by the Traffic Shaper API

Traffic Shaper standalone application algorithm

In order to tightly control the filtered flow, a tracking algorithm has been designed to simultaneously maintain the UXM buffer full yet prevent it from filling beyond the LTE's link capacity.

It relies on a control path over ICMP between the standalone application and the UE.

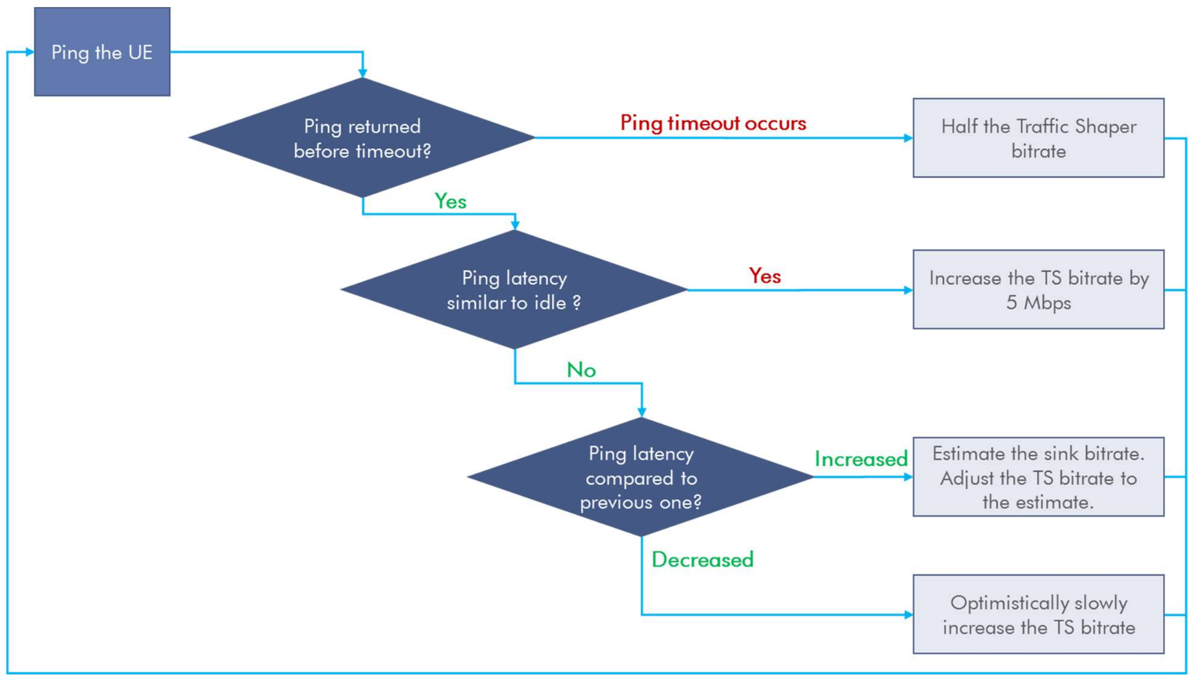

Ping the UE
Ping returned before timeout?
Ping timeout occurs
Half the Traffic Shaper bitrate
Yes
Ping latency similar to idle ?
Yes
Increase the TS bitrate by 5 Mbps
No
Ping latency compared to previous one?
Increased
Estimate the sink bitrate. Adjust the TS bitrate to the estimate.
Decreased
Optimistically slowly increase the TS bitrate

Figure 90 Traffic Shaper standalone application algorithm overview

There are 4 overall outcomes of a ping transmission, from top to bottom:

1) Half the traffic shaper bitrate:
   This is the worst scenario: the UXM buffer has more than 2s of data queued, LTE link is drowned in old packets.
   a. The traffic shaper detects that the UXM buffer is too full and quickly decreases the bitrate to allow the buffer to empty itself, before refilling it with fresh packets
   b. After the UXM buffer has recovered, the traffic shaper bitrate is increased to match the link capacity
2) Increase the TS bitrate by 5 Mbps
   This is a bad scenario: the UXM buffer is nearly empty, the pings are ”pass-through”. The buffer needs to be refilled more aggressively.
   a. The traffic shaper detects the UXM buffer being empty and quickly increases the bitrate to saturate the available link capacity (in ~6 seconds due to the 30 Mbps bitrate increase)
   b. Once the link capacity is achieved, the traffic shaper-imposed bitrate matches the capacity.
3) Adjust the TS bitrate to the estimate
   Stationary mode: the UXM buffer is increasing in size as noted by ping RTT increase, and the traffic shaping bitrate is adjusted to match estimated capacity.
   a. When the ping latency increases, a new sink bitrate estimate is calculated and the traffic shaper bitrate adapted.
4) Optimistically slowly increase the TS bitrate
   Optimistic increase of the traffic shaper bitrate in the case the ping latency decreases to avoid empty UXM buffer situations.

a. When the LTE link capacity undergoes small changes (light increase/decrease), the algorithm optimistically increases the traffic shaper bitrate to keep the buffer full.

8.4.2 Testbed deployment

Python configuration

This installation guide requires an existing functional deployment of the UMA impairment server, as it will only be updated with new features.

New command element: TSH (TSH:PAR for parameters, TSH:ENA to enable, TSH:CLN to clean up).

Linux commands being sent are exposed in TransportEmulator.py.

Key variable to edit is: “FORMAT_TRAFFIC_SHAPING_IDB_ENABLE”

It highlights which bytes in a packet the filtering is happening upon.

<u>Example filters</u>:

- Filtering all UDP packets: “match u8 0x11 0xff at **9**”
    - This matches a single byte to the value of 0x11 at position 9 (in the IP header, the protocol is at position 9, and 0x11 is 17, which corresponds to UDP)

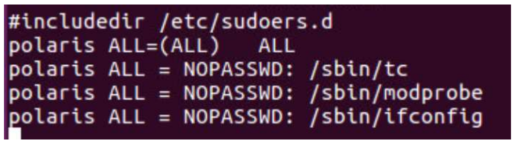
- Filtering all UDP within GTP packets: “match u8 0x11 0xff at 49”
    - Here, we are looking at the position 49, because of 20 bytes for the IP header, 8 bytes for UDP, 12 bytes for GTP header, and then 9th byte in the IP header:

IP packet				
IP header	IP payload			
	UDP header	UDP payload		
		GTP header	GTP payload (IP packet)	
			IP header	IP payload
20 bytes	8 bytes	12 bytes	Protocol at 9th byte	

- Filtering can be made in an arbitrary way, to any byte, as long as the filter in the variable highlighted above is modified

Shell configuration

Additional commands have to be added to the sudoers file, using sudo visudo:



```
#includedir /etc/sudoers.d
polaris ALL=(ALL)   ALL
polaris ALL = NOPASSWD: /sbin/tc
polaris ALL = NOPASSWD: /sbin/modprobe
polaris ALL = NOPASSWD: /sbin/ifconfig
```

Figure 91 Sudoers file after edits

- /sbin/tc should already be there from a previous Impairments server installation
- Add /sbin/modprobe and /sbin/ifconfig to the list of commands to run without sudo

This is because the new unix commands used by TrafficShaper API calls run modprobe and ifconfig, and the architecture of the Impairment Server runs these commands via shell opened from a remote host.

Examples of traffic shaping API calls

The following example commands (note that :DEF: represents the network interface configured in the Impairment server interfaces.cfg configuration file:

- TRA:DEF:TSH:CLN
  - Cleans up the connection
- TRA:DEF:TSH:ENA 1,192.168.3.11
  - Enables traffic filtering for the DUT at IP 192.168.3.11
- TRA:DEF:TSH:PAR 10mbit,500ms,10kb
  - Limits the filtered traffic towards the UE at 10mbit of peak bitrate, with 500ms max latency

If the commands execute successfully (no error in the SCPI reply, no error in the shell which runs the server), then the autonomous C# executable can be run.

Testing should be performed by ideally running a type of traffic affected by the filter, for example a downlink UDP stream, or a UDP stream within a GTP flow.

Typical test flow:

- Configure the UXM channel impairments (AWGN, DL scheduling) for a LTE link capacity of 30 mbps
- Connect the UE, configure an iperf sink at the UE
- Configure an iperf source in the network
- Stream large bitrate UDP from the source to the sink
- When enabling the traffic filtering and by sending traffic shaping commands, there should be visible impact on the DL UDP traffic seen at the UE side

Traffic Shaper application configuration

The traffic shaper C# executable comes with a configuration file "TRIANGLETrafficShaper.exe.config"
It contains the settings that the executable will use at runtime:

- **impairmentServerIP** & **impairmentServerPort**
    - On which IP and port will the executable reach the Python Impairment server
- **interfaceAlias**
    - What is the alias of the network interface that the Impairment server is trying to reach – this is configured in interfaces.cfg file of the Impairment server
- **ueIP**
    - What's the UE IP (as the traffic shaper filters packets towards a unique UE IP)
- **averagePing**
    - What is the average round-trip time between the Windows host which will run the C# executable and the UE. It is crucial that this value is correctly set as the traffic shaping uses it to determine whether the buffer is empty or filling up.
- pingTimeout (default: 2000ms)
    - How long to wait before declaring the ping lost and claiming that the buffer is full. If this value is too high, the UXM buffer will overfill with stale packets.
- minimumBitRate (default: 1mbps)
    - Minimum bitrate value that the Traffic Shaper will apply
- maximumBitRate (default: 60mbps)
    - maximum bitrate value that the Traffic Shaper will apply
- initialBitRate (default: 10mbps)
    - initial bitrate value that the Traffic shaper will apply at app start

Before running the executable, it is crucial that:
- It is on the same network (or at least can reach via SSH or ping) as the Python impairments server and the UE.
- The 5 configuration parameters in bold above are edited and set properly.

The traffic shaper executable will run autonomously when double clicked.
Built on VisualStudio 2017. The target framework is .net 4.6.2.

8.4.3 Testbed results & conclusion

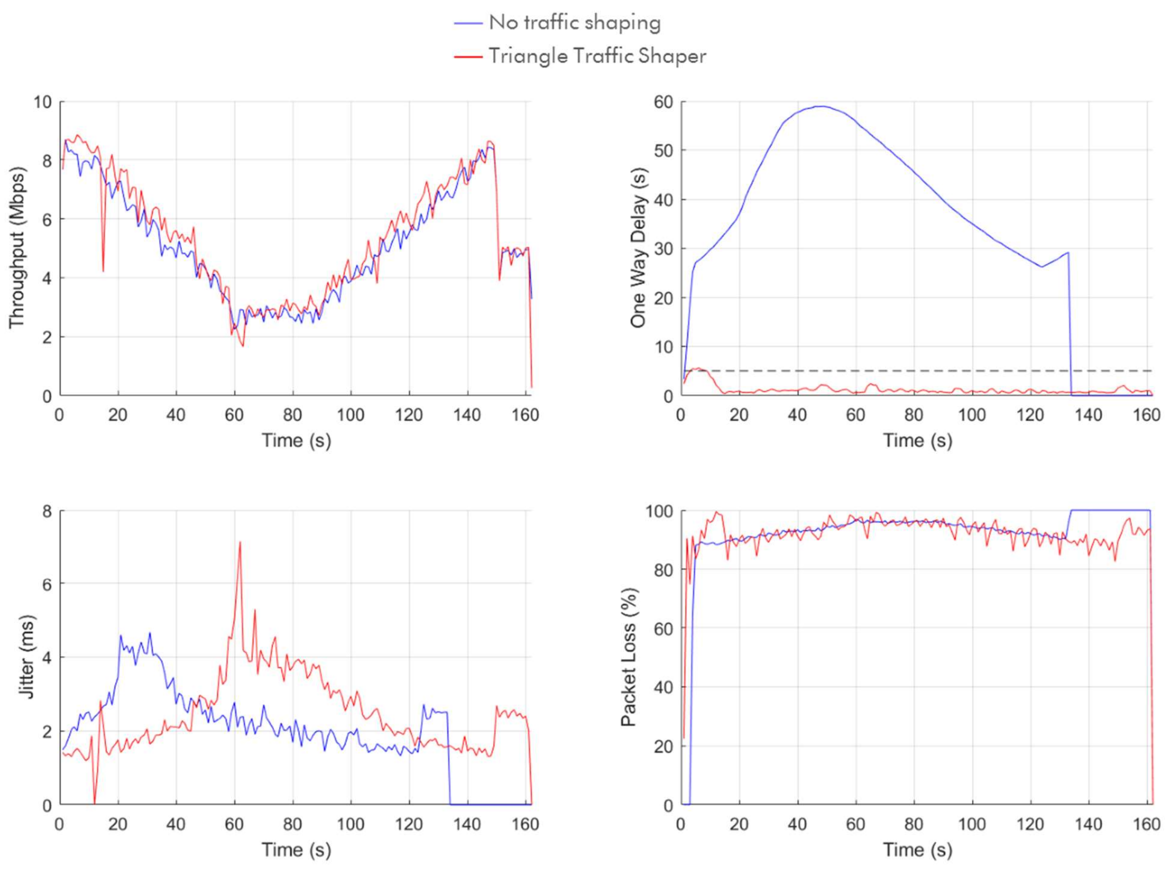


Figure 92 Traffic shaping testing in Urban Pedestrian network scenario

From the figure above it can be observed:

- There is identical throughput when activating the traffic shaper (no link capacity loss, UXM buffer never empty)
- Comparable jitter
- One-way delay significantly decreased (only "fresh" packets are reaching the UE)
- Comparable packet loss (as the link capacity is identical), however, packets are discarded "along the way" rather than only the last ones.

In conclusion, the proposed Traffic Shaper:

- Effectively reduces the UDP one-way delay while maintaining comparable throughput
  - This significantly reduces the age of the UDP packets reaching the UE
- Does not impact any other traffic
- Integrates in the existing TRIANGLE testbed architecture
  - New type of channel impairments via update of the UMA Python server
  - Can intercept all S1 packets leaving the core network via changes on Linux network interface
  - Can run on only Windows machine in the subnetwork
- Is configurable in the type of traffic to be shaped
  - Plain UDP with traffic passing through a Linux machine

- Or UDP within GTP within UDP for an external S1 interface testbed

9 User Equipment (UE) and Accessories

9.1 Supported UEs

The TRIANGLE testbed provides a set of mobile phones ready to be used. These devices have been connectorized as shown in this section. Devices sent for testing in the TRIANGLE testbed by users must be provided in pairs, one connectorized. Table 48 shows the current status of commercial devices connected to the TRIANGLE testbed.

Table 48 Current status of devices integrated into the testbed

Device	**Main Ant 1**	**Main Ant 2**	**Diversity Ant 1**	**Diversity Ant 2**	**Battery**
Samsung Galaxy S4	Yes	N/A	Yes	N/A	Yes
Samsung Galaxy S5 Neo	Yes	N/A	No	N/A	Yes
Samsung Galaxy S6	Yes	N/A	Yes	N/A	Yes
Samsung Galaxy S7	Yes	No	Yes	No	Yes
iPhone 7 Plus	Contact Ant	N/A	No	N/A	No
Samsung Galaxy S8	Yes	No	Yes	No	Yes
Samsung Galaxy S9	Yes	No	Yes	Yes	Yes
HTC One	Contact Ant	N/A	Contact Ant	N/A	No

9.2 DUT HUB

Similarly, there is also a need to connect the UEs via USB ports. This connection is used to send command to the UEs and to operate the applications and services. To avoid the challenges of connecting all the UEs directly, we use a Keysight DUT HUB. This is basically a USB hub which can be controlled via SPCI commands. This allows us to only connect the desired UE to the computer controlling the UE. The concept is very similar to the RF switch explained early.

The DUT features 4 high power USB ports. It is capable of delivering 1.5 A per port (Max. 5 A in total). It is able to power on or power off each USB port separately. The list of commands is provided in the table below:

Table 49 DUT HUB SCPI command reference

Command	**Description**
*ON <port>	Turns on power to a given port identified by <port>. E.g. "*ON 1" to turn on power for port 1.

*OFF<port>	Turns off power to a given port identified by <port>. E.g. "*OFF 1" to turn off power for port 1.
RATE<Hz>	Set the sampling rate to between 20 and 1000 Hz.
READ?	Returns the last acquired samples as a IEEE-488 block, and clears the sample buffer. Samples are encoded as signed 16-bit numbers in bundles of 4 – one value for each port. The unit is 0.1 mA. E.g. "READ?" might return #18<0x10><0x01><0x0><0x0><0x40><0x0><0x0><0x0> This indicates that 1 sample has been acquired for each port. The first port returned 0x10,0x01 which is 272 as a 16-bit number, corresponding to 27.2 mA. Port 2 and 4 are 0 mA, while port 3 measures 6.4 mA.
READ:ASC?	Returns the last acquired samples as comma-separated numbers in bundles of 4 samples, and clears the sample buffer. E.g. "READ:ASC?" could return: 272,0,64,0,217,0,67,0 Corresponding to two acquired samples. 27.2, 0.0, 6.4, 0.0 mA for the first sample. 21.7, 0.0, 6.7, 0.0 mA for the second sample.

9.3 Android Support

The TRIANGLE testbed supports the following features on Android devices:

- Device and App automation, for controlling the whole device interface or a single application.
- Access to the logging messages generated by the device during the execution of the experiments.
- GPS emulation by generating the satellite radio signals.
- Traffic capture in pcap format.

9.4 iOS Support

The TAP plugins developed for controlling devices based on iOS has been integrated into the master template which drives the execution of tests launched through the Portal. Thus, iOS devices can be selected in the Portal during the Creation of the testing campaign.

9.5 IoT devices performance characterization

The Keysight UXM base station emulator supports cellular IoT technologies, including NB-IoT and LTE-M. Any off-the-shelf cellular IoT device or a pre-commercial development kit can be connected to a highly configurable emulated cell, and multiple data performance as well as power consumption KPIs can be extracted.

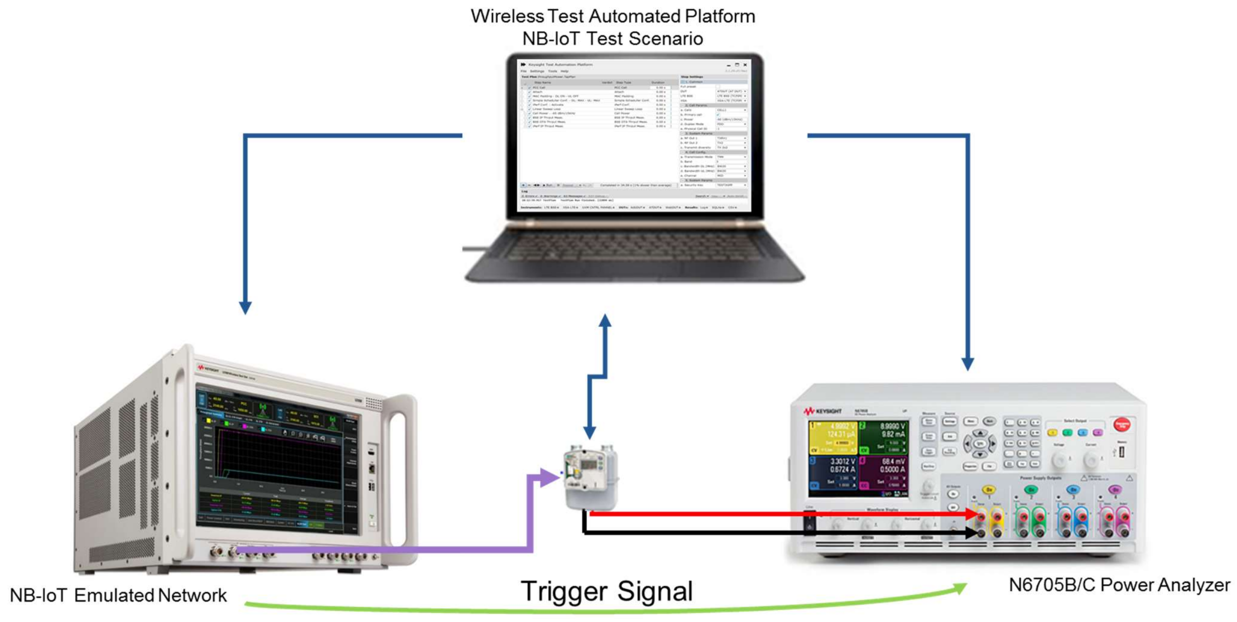
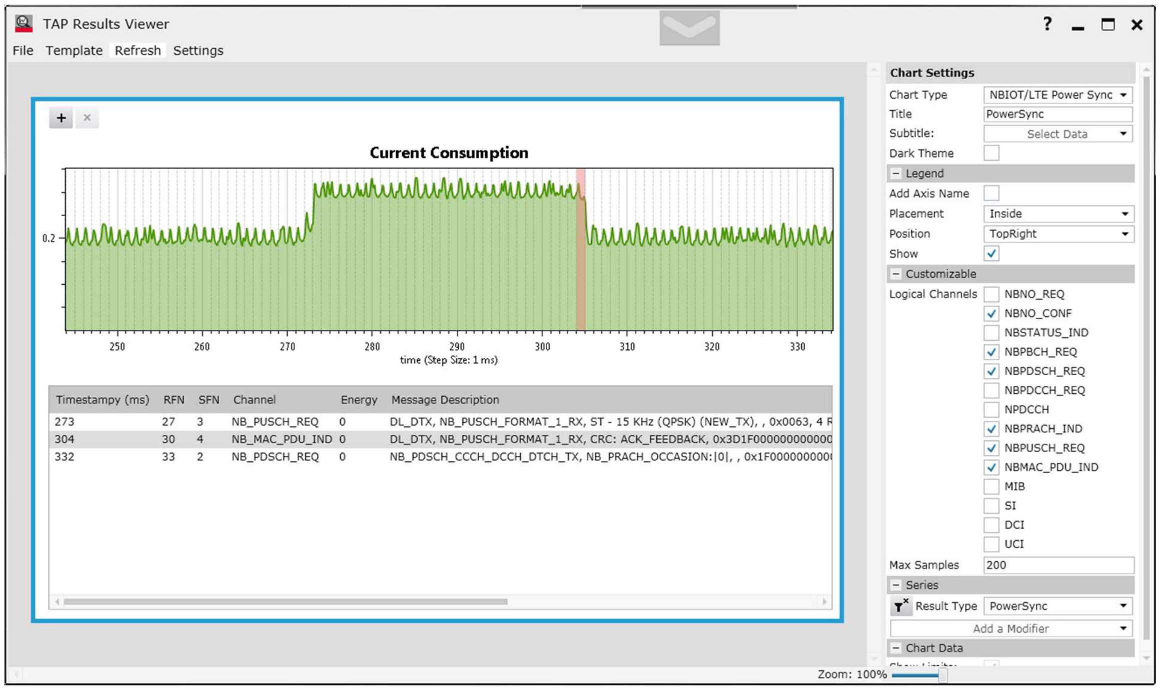


Figure 93 IoT Power & Performance characterization testbed

The IoT KPIs focused on during the TRIANGLE project are:

- Power consumption
    - Impact of radio aspects
        - TX power (P-max sweeps), UL/DL rate (across MCS & RU), data over C-plane versus U-plane
    - Impact of network parameters
        - Subspacing, multi-tone vs single tone, number of repetitions, coverage levels
    - Impact of protocol procedures
        - SYNC consumption, Attach consumption
    - Impact of power saving mechanisms
        - PSM, I-eDRX, C-DRX, cost of turning OFF the device
- Data performance
    - Downlink data throughput under static channel with AWGN, across multiple MCS indexes, different number of data, DCI or HARQ repetitions, or other IoT downlink scheduling parameters.

An accurate matching of power consumption to different UE states is achieved through synchronization of power measurements with layer 1, 2 and 3 messages exchanged between the cell and the UE, at a sub-1ms accuracy.



Timestampy (ms)	RFN	SFN	Channel	Energy	Message Description
273	27	3	NB_PUSCH_REQ	0	DL_DTX, NB_PUSCH_FORMAT_1_RX, ST - 15 KHz (QPSK) (NEW_TX), , 0x0063, 4 R
304	30	4	NB_MAC_PDU_IND	0	DL_DTX, NB_PUSCH_FORMAT_1_RX, CRC: ACK_FEEDBACK, 0x3D1F000000000000
332	33	2	NB_PDSCH_REQ	0	NB_PDSCH_CCCH_DCCH_DTCH_TX, NB_PRACH_OCCASION:|0|, , 0x1F00000000

Chart Settings
Chart Type: NBIOT/LTE Power Sync
Title: PowerSync
Subtitle: Select Data
Dark Theme: ☐
− Legend
Add Axis Name: ☐
Placement: Inside
Position: TopRight
Show: ✓
− Customizable
Logical Channels:
☐ NBNO_REQ
✓ NBNO_CONF
☐ NBSTATUS_IND
✓ NBPBCH_REQ
✓ NBPDSCH_REQ
☐ NBPDCCH_REQ
☐ NPDCCH
✓ NBPRACH_IND
✓ NBPUSCH_REQ
✓ NBMAC_PDU_IND
☐ MIB
☐ SI
☐ DCI
☐ UCI
Max Samples: 200
− Series
Result Type: PowerSync
Add a Modifier
− Chart Data

Figure 94 Synchronization of IoT Protocol & PHY messages with power consumption

The power consumption KPIs can be used in conjunction with an analytical model to predict an IoT device’s estimated battery lifetime, based on a realistic coverage level and traffic model.

10 Local Applications and servers

10.1 Servers virtualization (TNO extension Release 3)

This section describes the orchestrated cloud platform delivered by TNO as an extension to the TRIANGLE testbed. It consists of two main parts: The Orchestrator and the Virtual Infrastructure Manager.

The Management and Network Orchestration (MANO) stack allows for rapid deployment of Virtualized Network Functions (VNFs), their flexible configuration, lifecycle monitoring and decommissioning in order to automatically operate a potentially complex Network Service (NS). A model driven approach is taken to describe VNFs and NSs. MANO enhances interoperability with other components in the system such as Virtual Infrastructure Managers (VIMs) where the (virtualized) functions are deployed. To allow these deployments, VIM manages storage and networking resources, providing flexible and scalable infrastructure for the modern services such as Virtual Reality or ultra-high definition TV.

10.1.1 Cloud infrastructure description

TNO has realized and integrated a cloud environment based on OpenStack in the TRIANGLE testbed. This section describes how the physical and virtual infrastructure is set up and has been configured.

The implemented cloud environments are running on a single hypervisor described in Section 10.1.2. The hypervisor hosts multiple clouds described in Section 10.1.3 and performs routing and firewalling for their respective networks described in Section 10.1.4. The Juju models describing the cloud configuration are found in Section 10.1.5. An overview of access to the OpenStack Dashboard, command-line Client and REST APIs for using the cloud is given in Section 10.1.6, while Section 10.1.7 gives an overview of all important used IP addresses. An overall overview of the infrastructure is presented in Figure 95.

10.1.2 Hypervisor

Based on the instructions of the TRIANGLE project, the cloud environment has been installed on a single server placed in the testbed location at Malaga University. To provide isolation between the different types of nodes and physical machines usually found in a cloud environment, different functional nodes such as the cloud controllers, compute nodes, storage nodes, networking nodes and gateways, deployment managers and MANO orchestrators are configured to run as KVM instances on the hypervisor server managed by *libvirt*. Overall, there are 3 types of KVM instances:

- Manually-deployed instances, instances that are installed manually from an ISO. These nodes include Ubuntu MAAS and the DANE (note that DANE can and typically will be instantiated as a part of the specific experiment).
- MAAS-deployed instances, these instances have not been installed manually, but are installed through Ubuntu MAAS configuration and are later configured manually. These instances include Juju and the MANO orchestrators.
- Juju-configured instances, these instances have been installed and configured fully automatically through Juju models containing abstract configuration and functional linking description. Juju employs Ubuntu MAAS to automatically install nodes with an operating system prior to deploying configuration. The instances fully deployed through Juju consist of the Juju-Controller itself that in turn manages the cloud environments, and all Controller, Compute, Networking and Storage nodes found in each cloud.

The hypervisor has 4 SATA hard disks, configured in 2 software RAID-10 arrays for reliability purposes. The first array hosts the hypervisor software and configuration itself, while the second array hosts the storage of the instances. For performance reasons, the TRIANGLE consortium should consider upgrading this system to a two-array, four-hard disk hardware-based RAID system based on SAS instead of SATA.

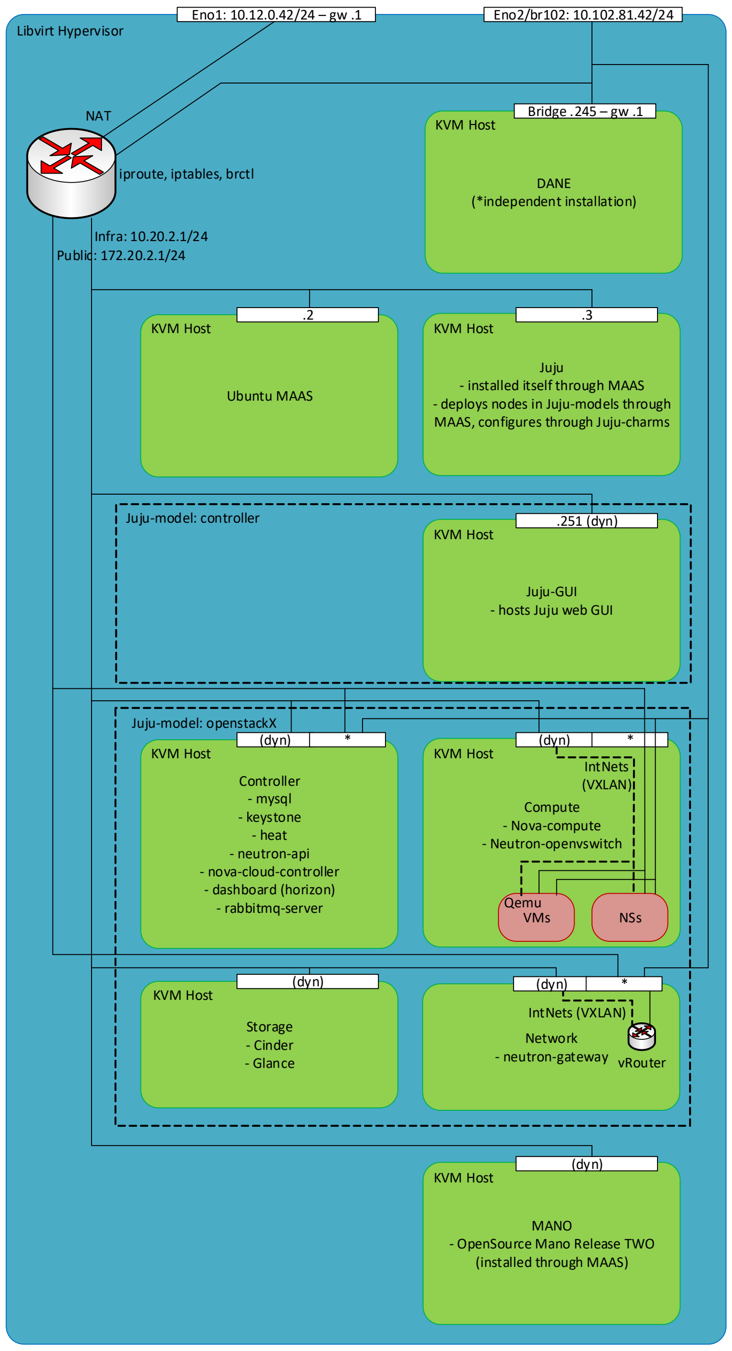
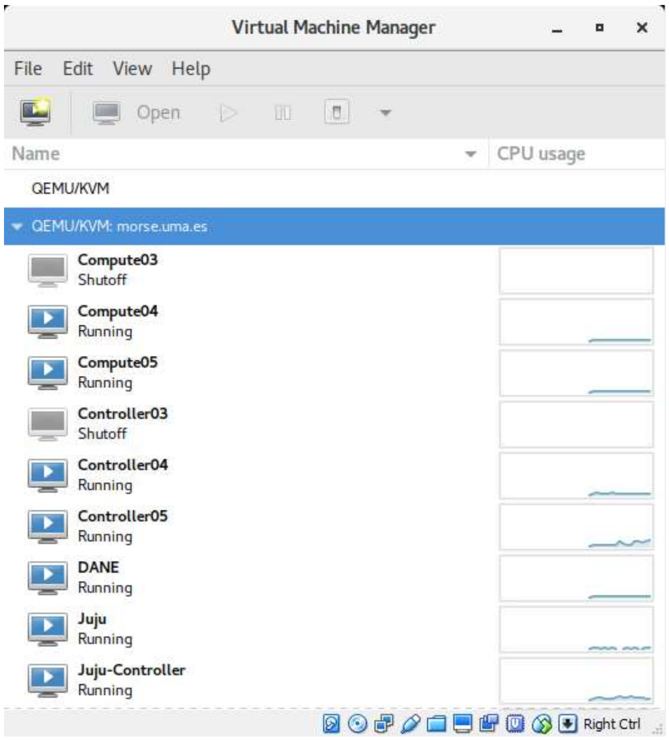
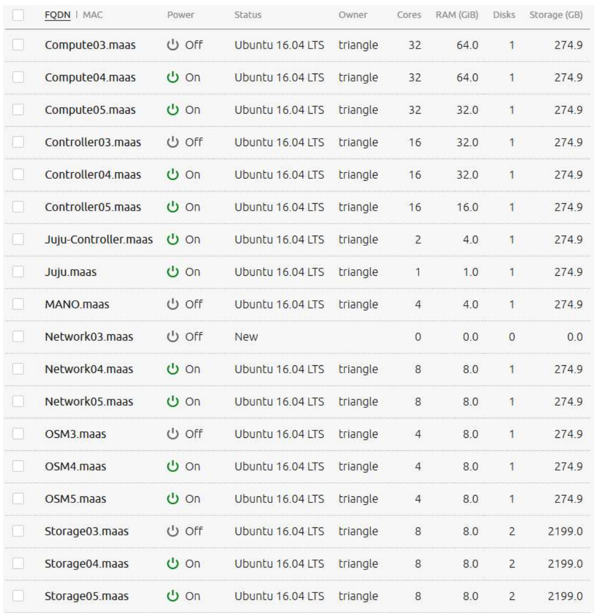


Figure 95 Architecture overview



QEMU/KVM
QEMU/KVM: morse.uma.es
Compute03 Shutoff
Compute04 Running
Compute05 Running
Controller03 Shutoff
Controller04 Running
Controller05 Running
DANE Running
Juju Running
Juju-Controller Running

Figure 96 Hypervisor configuration through virt-manager



FQDN | MAC	Power	Status	Owner	Cores	RAM (GiB)	Disks	Storage (GB)
Compute03.maas	Off	Ubuntu 16.04 LTS	triangle	32	64.0	1	274.9
Compute04.maas	On	Ubuntu 16.04 LTS	triangle	32	64.0	1	274.9
Compute05.maas	On	Ubuntu 16.04 LTS	triangle	32	32.0	1	274.9
Controller03.maas	Off	Ubuntu 16.04 LTS	triangle	16	32.0	1	274.9
Controller04.maas	On	Ubuntu 16.04 LTS	triangle	16	32.0	1	274.9
Controller05.maas	On	Ubuntu 16.04 LTS	triangle	16	16.0	1	274.9
Juju-Controller.maas	On	Ubuntu 16.04 LTS	triangle	2	4.0	1	274.9
Juju.maas	On	Ubuntu 16.04 LTS	triangle	1	1.0	1	274.9
MANO.maas	Off	Ubuntu 16.04 LTS	triangle	4	4.0	1	274.9
Network03.maas	Off	New		0	0.0	0	0.0
Network04.maas	On	Ubuntu 16.04 LTS	triangle	8	8.0	1	274.9
Network05.maas	On	Ubuntu 16.04 LTS	triangle	8	8.0	1	274.9
OSM3.maas	Off	Ubuntu 16.04 LTS	triangle	4	8.0	1	274.9
OSM4.maas	On	Ubuntu 16.04 LTS	triangle	4	8.0	1	274.9
OSM5.maas	On	Ubuntu 16.04 LTS	triangle	4	8.0	1	274.9
Storage03.maas	Off	Ubuntu 16.04 LTS	triangle	8	8.0	2	2199.0
Storage04.maas	On	Ubuntu 16.04 LTS	triangle	8	8.0	2	2199.0
Storage05.maas	On	Ubuntu 16.04 LTS	triangle	8	8.0	2	2199.0

Figure 97 Overview of nodes deployed through MAAS, cloud-related nodes are initiated and controlled through Juju.

10.1.3 Clouds

The cloud environment consists on 3 different clouds, all running on the previously described hypervisor. All clouds are running OpenStack Pike deployed through the Ubuntu MAAS and Juju services described in Section 10.1.5. Due to historic reasons, the clouds have the following names and functionality:

- Cloud5; the Stable cloud that can be used in production
- Cloud4; the Staging/Integration cloud in which we test and verify functionality before moving them to the Stable cloud.
- Cloud3; the Development cloud for experimental testing of functionality before moving it to Staging

10.1.4 Networks

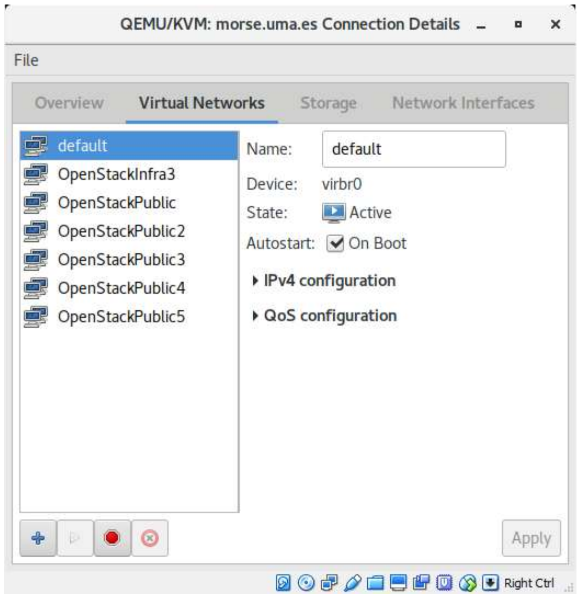
The following networks with respective subnets are considered from the cloud environments. Unless stated otherwise, the .1 IP address specify both the gateway and DNS server for that subnet and all these networks exist at the hypervisor through a *virsh* virtual network. DNS requests are forwarded to the UMA DNS server at 150.214.40.11 by the recursive DNS server *dnsmasq* running on the hypervisor.

The following networks are exposed from cloud environment to the testbed network.

- Access Network - 10.12.0.0/24:
  This is the access network used by the hypervisor to access and forward traffic towards the Internet, only the main Network Interface Card (NIC) of the hypervisor has an IP address in this range. Ideally, Network Address Translation (NAT) should be performed by the main router of the TRIANGLE network, however, due to limited functionality of the main router the hypervisor masquerades outgoing traffic on this NIC towards the Internet.
- TAP Network - 10.102.81.42:
  This is the network used for the devices part of and under test by the TAP testbed. From Cloud4 and Cloud5 instances can be fired up that are directly bridged into this network, for this respectively the ranges 10.102.81.100-149 and 10.102.81.150-199 have been reserved. Due to limitations in the main router of the TRIANGLE testbed, hosts besides cloud instances in this network need to manually configure network routes via 10.102.81.42 (the hypervisor IP address) to access the following 4 subnets.
- OpenStack Infra - 10.20.2.0/24:
  Through the Infrastructure network the VMs on which the cloud-infrastructure itself run are configured and maintained.
- OpenStack Public5 (Stable) - 172.16.4.0/24:
  This is the public network connected to the Stable cloud5, instances and floating IPs on this network are dynamically addressable by hostname through the subnet *<hostname>.cloud5.morse.uma.es*.
- OpenStack Public4 (Staging) - 172.16.3.0/24:
  This is the public network connected to the Staging cloud4, instances and floating IPs on this network are dynamically addressable by hostname through the subnet *<hostname>.cloud4.morse.uma.es*.
- OpenStack Public3 (Development) - 172.16.2.0/24:
  This is the public network connected to the Development cloud3.

The following internal networks exist for functional reasons within the OpenStack clouds; but are not exposed outside the cloud environments or hypervisor. They exist as private VXLAN networks within the cloud to interconnect instances with each other. Access to the exposed networks is realized using virtual routers in OpenStack performing NAT.

- OpenStack Internal5 (Stable) – Internal network on Stable Cloud5 interconnected to network OpenStack Public5.
- OpenStack Internal4 (Staging) – Internal network on Staging Cloud4 interconnected to network OpenStack Public4.
- OpenStack Internal3 (Development) – Internal network on Development Cloud3 interconnected to network OpenStack Public3.
- OpenStack TAP-Internal5 (Stable) - Internal network on Stable Cloud5 interconnected to network TAP Network.
- OpenStack TAP-Internal4 (Staging)- Internal network on Stable Cloud4 interconnected to network TAP Network.



Name: default
Device: virbr0
State: Active
Autostart: On Boot

Figure 98 Network overview of networks managed by lib-virt

10.1.5 Juju models describing the cloud infrastructure

The OpenStack clouds available at the TRIANGLE testbed are deployed and configured through Ubuntu Juju, a service configuration and modelling tool that automatically configures servers based on the abstract configuration expressed in a Juju model. Each functionality is expressed and configured through a Juju charm, which can be mapped to a physical machine, a virtual machine or an LXD container. Due to the ease of creating, reconstructing reconfiguring and cleaning up functions, we have a very fine-grained distribution where each function described by a Juju charm is mapped to its individual LXD container in a KVM instance or the root container of a KVM instance if the charm's functionality requires so. Figure 99 presents an overview of the Juju model, containing of the following functions mapped to the following nodes:

- Controller: A node executing the functions related to management, configuration and monitoring of the cloud, consists out of
  - Cinder-API: Offers the API and scheduling functionality for the block storage service and provides management of volumes.
  - Heat: OpenStack Orchestration service, a non-ETSI aligned alternative for MANO/NFV orchestration (note: a different solution, i.e., Open Source Mano, is actually used as the orchestrator).
  - Horizon: The OpenStack dashboard / web GUI.

  - Keystone: Identity, authentication and authorization service.
  - MySQL: A Percona database cluster running MariaDB instances
  - Neutron-API: Virtual network service, enabling network management, QoS, ACLs, etc.
  - Nova-cloud-controller: Cloud computing controller, scheduling, managing and monitoring computing resources through its subservices nova-scheduler, nova-api and nova-conductor.
  - RabbitMQ: Advanced Message Queuing Protocol server used by all services to interconnect.
- Compute:
  - Nova-compute (root container): Provides hypervisor service to run instances on.
- Network:
  - Neutron-gateway (root container): Provides central networking services such as virtual routers to interconnect external (bridged or routed) networks and internal (VXLAN) private networks through routing, firewalling and NAT.
- Storage:
  - Cinder-volume (root container): Actual storage of LVM volumes on nodes, managed by Cinder-API.
  - Glance: Image registration and discovery service
- Subordinate functions: Elements that upgrade functionality of the main elements described in the previous 4 categories.
  - Neutron-OpenvSwitch: Provides Neutron-API and Nova-compute with Open vSwitch functionality.
  - NTP: Provides time synchronization to nodes.

The Juju model configuring the clouds is custom-made for the TRIANGLE testbed and in particular contains the following additional configuration:

- L2 and L3 network connectivity between external and private internal (VXLAN) networks using Open vSwitch SDN switches throughout.
- Separate networking nodes to provide gateway and floating IP address functionality.
- DNS integration such that nodes in the routed public networks can be accessed through their hostnames <hostname>cloud[4-5].morse.uma.es.
- Additional non-ETSI aligned orchestration module through the OpenStack Heat package (note: a different solution, i.e., Open Source Mano, is actually used as the orchestrator).
- Has split storage control and storage volume functionality for future scalability through LVM.
- Has per-machine restriction and tag descriptions to automatically select the correct VMs for their respective functions.

The full YAML export of the Juju model and the cloud-config scripts are available and are used to automatically create all cloud configuration once a cloud is reinitiated.

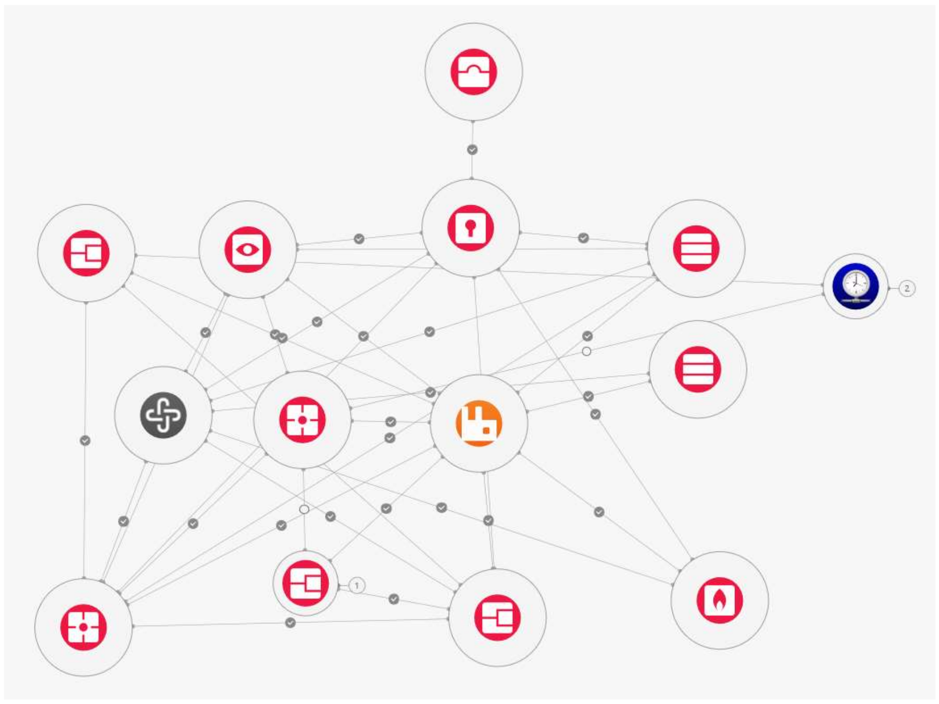


Figure 99 Juju model describing functional elements and their connections

10.1.6 Dashboard and API access

The cloud instances can be accessed and configured through both a Web GUI Dashboard (Horizon) and service-specific APIs. This section presents an overview of the available configuration services, section 10.1.7 gives an overall overview of the used IP addresses. Note that these IP addresses will change when elements of the cloud model are deleted and reinstalled through Juju. All addresses are publicly accessible from the networks described in section 10.1.4, given that local static routing is configured when residing in the Access or TAP Network. When connecting from outside the TRIANGLE Testbed network, an SSH Dynamic Tunnel (SOCKS Proxy) towards a node within the network can be used.

OpenStack Dashboard

The easiest way to configure and administer projects, instances, networks, authentication, storage, images, etc., is through the OpenStack Dashboard supplied by the Horizon package. It is accessible through either http://<OpenStack-Dashboard> or https://<OpenStack-Dashboard>, with the appropriate IP address selected from section 10.1.7. Annex 4 gives an introduction into the usage of OpenStack Dashboard.

CLI Access

Additionally, all configurations can be applied through the OpenStackClient (OSC) command-line client. OSC provides a shell interface with all configuration options available and is available under the *python-openstackclient* package in most package repositories. A full documentation of OSC can be found at [10]. In a nutshell, after installation the command *openstack* provides a shell that auto-completes and gives information on the fields necessary to complete a command. The following environment variables need to be set to connect to Stable Cloud5, which can be easily saved in a local file *admin-openrc5.sh* that can be sourced through `. admin-openrc5`:

Table 50 admin-openrc5.sh

```
export OS_AUTH_URL=http://10.20.2.45:5000/v3
export OS_PROJECT_DOMAIN_NAME="default"
export OS_PROJECT_NAME="admin"
export OS_USER_DOMAIN_NAME="default"
unset OS_TENANT_ID
unset OS_TENANT_NAME
export OS_USERNAME="admin"
export OS_PASSWORD="admin"
export OS_REGION_NAME="RegionOne"
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
unset OS_TOKEN
```

API Access

It is also possible to have programmatic access to the configuration of the OpenStack Cloud. In fact, both the OpenStack Dashboard and the OpenStackClient rely fully on API access to read and write OpenStack configuration. Additionally, API Access is for example used by the MANO orchestrators (OSM) connected to the OpenStack clouds to administer the appropriate configuration and is useful for custom environments to integrate with OpenStack. For example, we have used direct API access in our sample TAP scripts to configure QoS parameters in OpenStack.

Annex 4 provides a short introduction of this API. A full overview of all OpenStack API Documentation can be found at [11], we will give a short introduction to each API available in the TRIANGLE clouds and how to operate those.

Additionally, the cloud configuration can be orchestrated from the MANO orchestrators based on Open Source MANO (OSM) and the TAP client respectively described in sections 10.2 and 0.

10.1.7 Overall overview of IP addresses

The following services are accessible at the following IP addresses through the following connection methods

Host Name	IP addresses	Connections	Authentication
OpenStackCloud (Hypervisor)	10.12.0.42 10.102.81.42	ssh://morse.uma.es:11300 ssh://<IP-Address>	morse+tno (Key-auth)
MAAS	10.20.2.2	ssh://TRIANGLE@10.20.2.2 http://10.20.2.2/MAAS	TRIANGLE:TRIANGLE
Juju	10.20.2.3	ssh://ubuntu@10.20.2.3	(Key-auth)
Juju-Controller	10.20.2.215	https://10.20.2.251:17070/gui/u/admin	admin / Run `juju gui` at Ubuntu Juju for pw.
DANE	10.102.81.245	ssh://morse.uma.es:11341 ssh://10.102.81.245	tno (Key auth)

OSM5	10.20.2.44	https://10.20.2.44:8443 ssh://ubuntu@10.20.2.44	admin:admin (Key-auth)

For the Stable Cloud 5, the following services are deployed by Juju on 4 VMs labelled Compute05, Controller05, Network05 and Storage05. On these nodes, Juju creates an LXD container per service with its own IP address to allow fine-grained per-service configuration, deletion and recreation. Access to these web and API services is explained in section 10.1.6. Logging into the nodes through SSH is seldomly necessary as they are configured through the Juju model "OpenStack5" through the Juju Web GUI at *Juju-Controller*. If SSH access is necessary to any of these services, this is possible through the following command when logged into the Juju node: `juju ssh -m openstack5 <*unitname*>`, where the unit is build up from the servicename followed by its instantiation (generally "/0" when it is the first instantiation).

Service Name	Location	IP address
Nova-Compute	Root@Compute05	10.20.2.30
(Empty)	Root@Controller05	10.20.2.30
Cinder-API	LXD@Controller05	10.20.2.20
Heat	LXD@Controller05	10.20.2.41
Keystone	LXD@Controller05	10.20.2.45
MySQL	LXD@Controller05	10.20.2.40
Neutron-API	LXD@Controller05	10.20.2.21
Nova-Cloud-Controller	LXD@Controller05	10.20.2.47
OpenStack-Dashboard (Horizon)	LXD@Controller05	10.20.2.43
RabbitMQ-Server	LXD@Controller05	10.20.2.27
Neutron-Gateway	Root@Network05	10.20.2.14
Cinder-Volume	Root@Storage05	10.20.2.18
Glance	LXD@Storage05	10.20.2.34

The previous IP addresses may change when a cloud or elements of the clouds are deleted and reinitiated through Juju. The most recent IP addresses can be found through the Juju Web GUI at Juju-Controller or by running the command `juju status -m openstack5` on the Juju node itself. Due to their volatile nature, we have not included all service IP addresses of the Development Cloud 3 and Staging Cloud 4.

10.2 MANO – Management and Network Orchestration (TNO extension Release 3)

In this section we describe in more details the selected orchestrator platform being ETSI OSM (Open Source MANO [12]) in the context of the TRIANGLE testbed. To make the document self-contained, we provide some brief information about more generic aspects of MANO like architecture or deployment, however, the reader is referred to the OSM documentation for more extensive coverage of these topics.

10.2.1 Functions

The Management and Network Orchestration (MANO) stack allows for rapid deployment of Virtualized Network Functions (VNFs), their flexible configuration, lifecycle monitoring and decommissioning in order to automatically operate a potentially complex Network Service (NS). A model driven approach is taken to describe VNFs and NSs. MANO enhances interoperability with other components in the system such as Virtual Infrastructure Managers (VIMs) or Software-Defined Networking (SDN) controllers and is able to integrate with multiple instances and variants (solutions such as OpenStack, VMware ESXi, etc) of them.

10.2.2 Architecture

Open Source MANO (OSM (ETSI OSM, sd)) has an ambition to be a reference implementation of the ETSI standards. The general architecture is presented on Figure 100, extracted from [13]. Please refer to this document for the detailed architecture description.

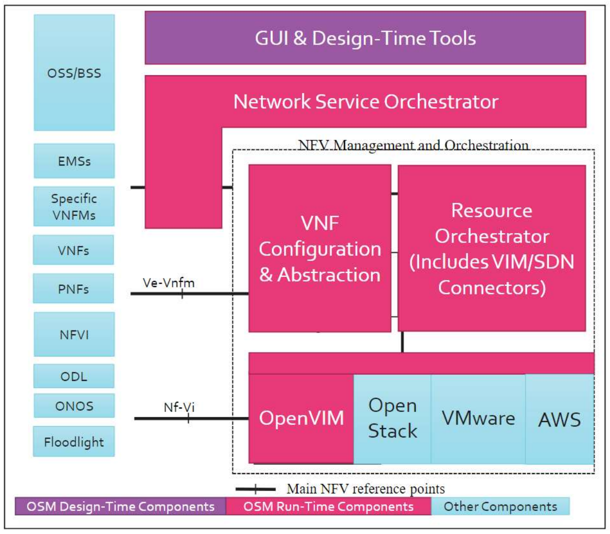

OSS/BSS
EMSs
Specific VNFMs
VNFs
PNFs
NFVI
ODL
ONOS
Floodlight
GUI & Design-Time Tools
Network Service Orchestrator
NFV Management and Orchestration
VNF Configuration & Abstraction
Resource Orchestrator (Includes VIM/SDN Connectors)
Ve-Vnfm
Nf-Vi
OpenVIM
Open Stack
VMware
AWS
Main NFV reference points
OSM Design-Time Components
OSM Run-Time Components
Other Components

Figure 100 OSM mapping to ETSI NFV MANO

10.2.3 Deployment

OSM deployment on the TRIANGLE testbed was performed using the binaries for Release TWO (a stable release during execution of the TRIANGLE Open Call 1 project), following the procedure from [Section 10.1.2] [14]. A dedicated Ubuntu 16.04 Virtual Machine (OSM5, 10.20.2.44) was deployed and configured to use (non-nested) LXD containers. As a result of the installation, three LXD containers are created in the OSM host: RO (Resource Orchestrator), VCA (VNF Configuration and Abstraction), and SO-ub (hosting Service Orchestrator and the User Interface), as shown in Figure 101 (source: [14].) and Table 51.

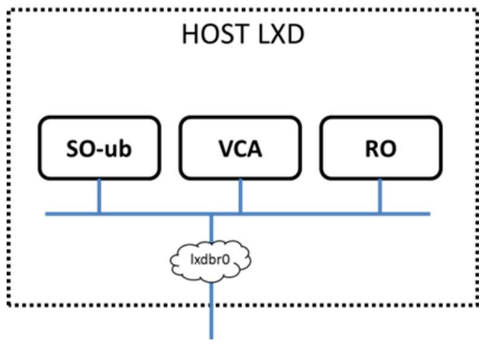


Figure 101 State of an OSM VM after OSM installation

Table 51 LXD containers after OSM installation

```
ubuntu@OSM5:~$ lxc list
+-------+---------+------------------------------+------+------------+-----------+
| NAME  |  STATE  |             IPV4             | IPV6 |    TYPE    | SNAPSHOTS |
+-------+---------+------------------------------+------+------------+-----------+
| RO    | RUNNING | 10.28.33.48 (eth0)           |      | PERSISTENT | 0         |
+-------+---------+------------------------------+------+------------+-----------+
| SO-ub | RUNNING | 10.28.33.163 (eth0)          |      | PERSISTENT | 0         |
+-------+---------+------------------------------+------+------------+-----------+
| VCA   | RUNNING | 10.44.127.1 (lxdbr0)         |      | PERSISTENT | 0         |
|       |         | 10.28.33.151 (eth0)          |      |            |           |
+-------+---------+------------------------------+------+------------+-----------+
```

Connectivity to OpenStack was then configured, following the procedure from [Section 10.2.2] [14], with the appropriate parameter for the Keystone address being http://10.20.2.45:5000/v2.0

10.2.4 Interfaces and integration with TAP

RO REST interface – Resource Orchestrator

The OSM Resource Orchestrator exposes the northbound REST interface which is documented in [15]. It allows, among others, to perform actions over tenants, data-centers, instances etc.

SO REST interface – Service Orchestrator

The OSM Service Orchestrator exposes the northbound REST interface which is documented in [16]. It allows, among others, to perform actions such as uploading service descriptors or instantiating a Network Service. At the time of writing this document the only available version refers to OSM Release ONE while the deployed version was Release TWO. While no thorough verification was performed, we have noticed some problems in running the examples (e.g., instantiating the network service using "nsd-ref" resource fails). As a partial remedy for these kind of problems, we decided to make a small modification in OSM Client (Section 0) which under the hood also uses REST. The client now prints a very verbose output which includes the REST URL, request headers, *json* payload, etc. which can in turn be used to compose the REST commands used for example by TAP script (Section 0).

CLI – Command Line Interface

The command line interface client is available for OSM MANO. The following sections describe the OSM client in more details and the modifications made to in the next section.

10.2.4.1.1 OSM client

The OSM community provided a python-based OSM Command Line Interface client. While for Release THREE it is installed by default, for Release TWO a manual installation is needed, followed by setting correct environmental variables, see both Table 52 and Table 51. The client was installed on the VM instance OSM5 (10.20.2.44) and the environmental variables were put for the convenience to *osmvars.sh* file (usage: *source osmvars.sh*)

Table 52 OSM Client installation, configuration and verification

```
sudo apt install libcurl4-gnutls-dev libgnutls-dev

sudo pip install git+https://osm.etsi.org/gerrit/osm/osmclient@v2.0.2

export OSM_HOSTNAME=10.28.33.163
export OSM_SO_PORT=8008
export OSM_RO_HOSTNAME=10.28.33.48
export OSM_RO_PORT=9090

osm vim-list
+----------------+--------------------------------------+
| vim name       | uuid                                 |
+----------------+--------------------------------------+
| openstack-site | 811971ae-c46a-11e7-b9ee-00163e1024a7 |
+----------------+--------------------------------------+
```

10.2.4.1.2 OSM client modifications

OSM client can be modified to display a very verbose output, helpful in inspecting REST calls, see Table 53. While useful, please note this is just a temporary fix and a more elegant solution (e.g., adding “verbose” switch) can be developed. Furthermore, if the descriptor was created in the GUI (see Section 0 and Section 17.2 in Annex 5), it is likely that it contains additional “meta-information”, which is necessary for the graphical layout of the service but has no infrastructural function. This information, if appearing in the *json* file submitted with the REST POST call to start a network service will cause an error and thus has to be removed. This is not a result of our modification but rather an inconsistency in the CLI client and GUI. Our TAP plug-in verifies the existence of the meta-information field, however does not remove it if present. See Table 54 for the details, with the meta-information part marked in red.

Table 53 OSM Client modifications

```
git clone -b 'v2.0.2' --single-branch https://osm.etsi.org/gerrit/osm/osmclient


#in the file
```

```
~/osmclient/osmclient/common/http.py

#in the function
def _get_curl_cmd(self, endpoint):

#the following line was added to allow for verbose output
    curl_cmd.setopt(pycurl.VERBOSE, True)

#to see the json content add around line 79 the print statement

            jsondata = json.dumps(postfields_dict)
            print jsondata
#to rebuild
~/osmclient$ sudo pip install . --upgrade

#to verify

$ osm vim-list
*   Trying 10.28.33.163...
* Connected to 10.28.33.163 (10.28.33.163) port 8008 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
[output omitted]
```

Table 54 Using OSM client with GUI generated descriptor may cause an error

```
ubuntu@OSM5:~/osmclient$ osm ns-create --ns_name rest_test13 --nsd_name nsd_3 --vim_account openstack-site
--admin_status ENABLED --ssh_keys piotr-zuraniewski-tno-nl
*   Trying 10.28.33.163...

[output omitted]

* Connection #0 to host 10.28.33.163 left intact
{"nsr": [{"short-name": "rest_test13", "ssh-authorized-key": [{"key-pair-ref": "piotr-zuraniewski-tno-
nl"}], "description": "default description", "om-datacenter": "811971ae-c46a-11e7-b9ee-00163e1024a7",
"nsd": {"id": "nsd_3", "constituent-vnfd": [{"member-vnf-index": 1, "start-by-default": "true", "vnfd-id-
ref":                                "ubuntu_2c_2G_1iface_vnfd"}],                              "meta":
"{\"containerPositionMap\":{\"1\":{\"top\":135,\"left\":435,\"right\":685,\"bottom\":190,\"width\":250,\
"height\":55},\"a8d499e9-ec0c-452b-bcb6-
6a0e8880fa67\":{\"top\":30,\"left\":135,\"right\":385,\"bottom\":85,\"width\":250,\"height\":55},\"vld-
1\":{\"top\":300,\"left\":447.5,\"right\":697.5,\"bottom\":338,\"width\":250,\"height\":38}}}",    "name":
"nsd_3", "vld": [{"mgmt-network": "false", "vnfd-connection-point-ref": [{"vnfd-connection-point-ref":
"eth0", "member-vnf-index-ref": 1, "vnfd-id-ref": "ubuntu_2c_2G_1iface_vnfd"}], "name": "vld-1", "id":
"vld-1"}]},    "admin-status":    "ENABLED",    "id":    "21e7c5ac-f60e-11e7-bedf-5254009bd772",    "name":
"rest_test13"}]}

[output omitted]

* Connection #0 to host 10.28.33.163 left intact
```

```
failed to create ns: rest_test13 nsd: nsd_3 result: {u'error': u'Resource target or resource node not found'}
```

GUI – Graphical User Interface

OSM also provides a graphical interface (called *launchpad*), in our case accessible under link https://10.20.2.44:8443/launchpad/

The GUI simplifies both descriptors design of descriptors as well as managing their catalogue, ssh keys deployment and network services instantiation, see Figure 102. From our experience, Launchpad GUI (service instantiate part) is however not always the best source of information in case of instantiation errors. Frequently, only “Failed” message (along with UUID of the instance) is given and for more insights MANO logs need to be inspected, see Section 10.2.6 for comments on troubleshooting.

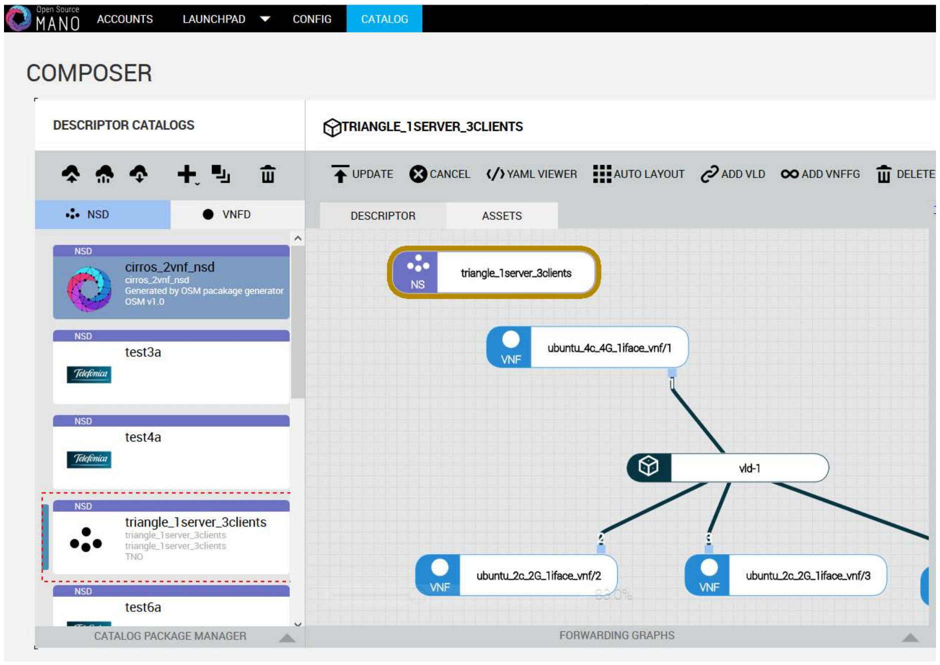


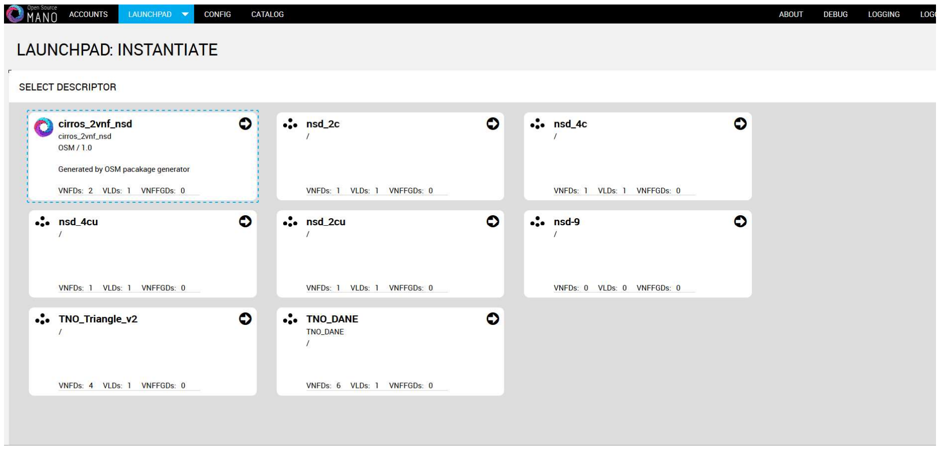
Figure 102 OSM GUI sample screenshot

TAP integration

We have integrated MANO with the TAP system. The detailed description of the steps along with the sample test plan is in Deliverable 3.4. Note, due to the bug/limitation of the SSH.NET library regarding implemented HostKey algorithms it is not possible to connect to openSSH servers in Ubuntu 16.04.4 and later (ssh_dispatch_run_fatal: Connection to 172.16.4.13: error in libcrypto). Modern servers offer ssh-rsa, rsa-sha2-512, rsa-sha2-256, ecdsa-sha2-nistp256, ssh-ed25519 while SSH.NET offers only ssh-rsa and (depreciated) ssh-dss. We have tested TAP plug-in with Ubuntu 16.04.3 (earlier version should also be supported).

10.2.5 Instantiation

Network service instantiation is performed using the Launchpad: Instantiate page (see Figure 103)



cirros_2vnf_nsd
cirros_2vnf_nsd
OSM / 1.0
Generated by OSM pacakage generator
VNFDs: 2 VLDs: 1 VNFFGDs: 0

nsd_2c
/
VNFDs: 1 VLDs: 1 VNFFGDs: 0

nsd_4c
/
VNFDs: 1 VLDs: 1 VNFFGDs: 0

nsd_4cu
/
VNFDs: 1 VLDs: 1 VNFFGDs: 0

nsd_2cu
/
VNFDs: 1 VLDs: 1 VNFFGDs: 0

nsd-9
/
VNFDs: 0 VLDs: 0 VNFFGDs: 0

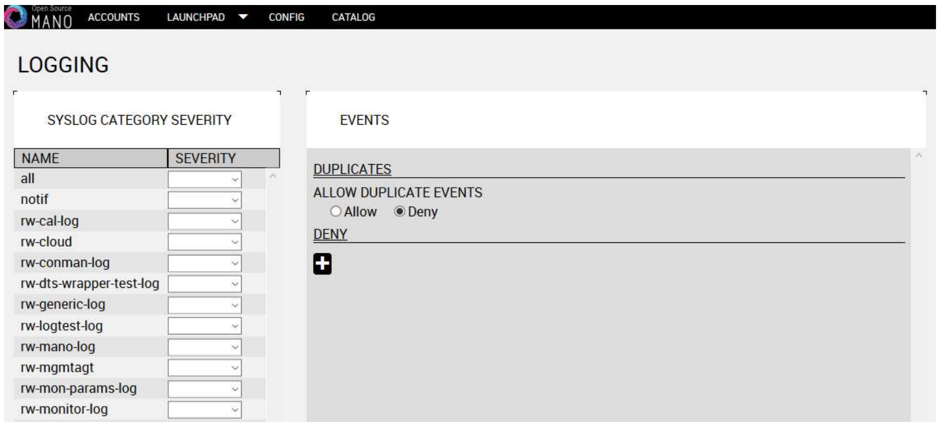
TNO_Triangle_v2
/
VNFDs: 4 VLDs: 1 VNFFGDs: 0

TNO_DANE
TNO_DANE
/
VNFDs: 6 VLDs: 1 VNFFGDs: 0

Figure 103 Instantiation

10.2.6 Troubleshooting

Most of the useful logs from MANO are stored on the LXD container running the given service (such as Resource Orchestrator). A good point to start with debugging is to log in to the OSM VMs (*ssh ubuntu@10.20.2.44*) and pull the Resource Orchestrator log (`lxc file pull RO/var/log/osm/openmano.log .`). The detailed description of the typical operations can be found in [16]. From the GUI, *Logging* and *Debug* sections are available, see Figure 104 and Figure 105



NAME	SEVERITY
all	
notif	
rw-cal-log	
rw-cloud	
rw-conman-log	
rw-dts-wrapper-test-log	
rw-generic-log	
rw-logtest-log	
rw-mano-log	
rw-mgmtagt	
rw-mon-params-log	
rw-monitor-log	

DUPLICATES
ALLOW DUPLICATE EVENTS
Allow [✓ Deny]
DENY

Figure 104 Logging and debug via OSM GUI

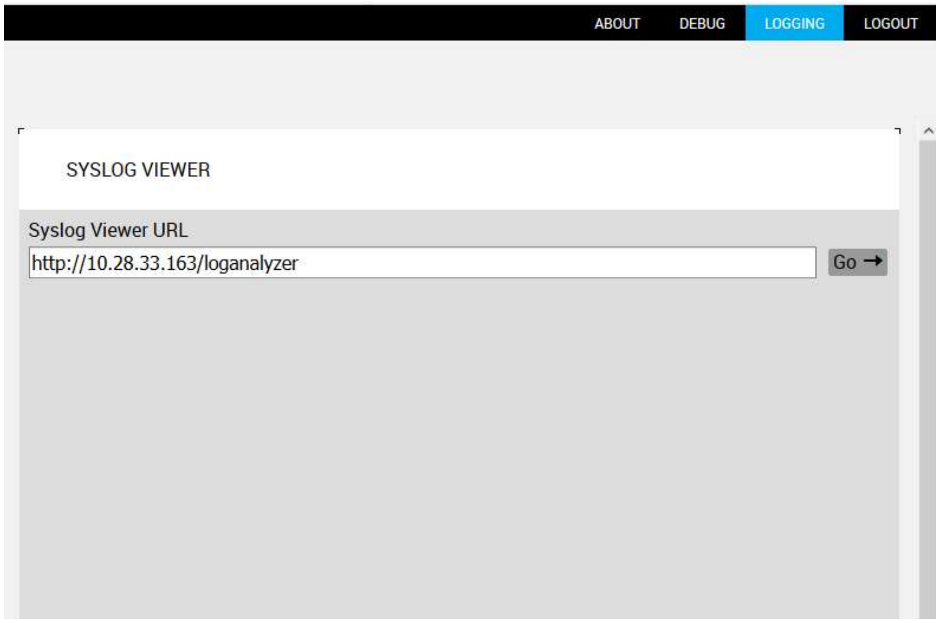


Figure 105 Logging and debug via OSM GUI

Annex 5 provides a sample workflow which can be used by the experimenter using MANO integrated with TAP.

10.3 MEC (Mobile Edge Computing) (CNIT Genova Extension Release 4)

This section describes the extension of the TRIANGLE testbed to support and to host Mobile Edge Computing (MEC) services. The extension of the TRIANGLE testbed towards MEC technologies allows experimenters to upload the “mobile edge” counterpart of the applications running on User Equipment (UE) to the TRIANGLE facilities, and consequently testing and validating the entire ecosystem in a highly flexible and controlled environment. The integration regards both the configuration tools available for the experimenters and the extension of the TRIANGLE testbed towards MEC.

The concept behind Mobile Edge Computing (MEC) consists in the deployment of computing and storage resources in the premises of the Telco Operators instead of within remote datacenters as envisaged by the Cloud Computing paradigm. This simple shift in the environment allows not only to lower latency thanks to the reduced distance with the users, but also to provide more degrees of freedom thanks the knowledge of user location and the network data available within the Telco premises.

As a result, Telco Operators can support MEC services characterized by real-time responsiveness and by a high degree of personalization of networking, billing and features that become feasible thanks to the information related to the knowledge of user location and the network data available within the Telco premises. Moreover, new business opportunities can be created by exploiting the more rapid deployment of applications and services to enable vertical industries to work on top of pre-existing infrastructure domains.

This extension has the goal of integrating the MEC paradigm in the TRIANGLE testbed. In this implementation, MEC services are composed of chains of Virtual Machines (VMs) potentially deployed in different Points of Presence (PoPs). The communication and information exchanged among VMs of the same service chain are provided by overlays called Back-end Networks (BN), while a Personal Network (PN) is associated to each User Equipment (UE) and is used to interconnect the UE to the associated service chains.

The deployment of such services in TRIANGLE requires the testbed to be extended according to the high-level representation in Figure 106. Extensions regard both the user interface and the testbed architecture itself.

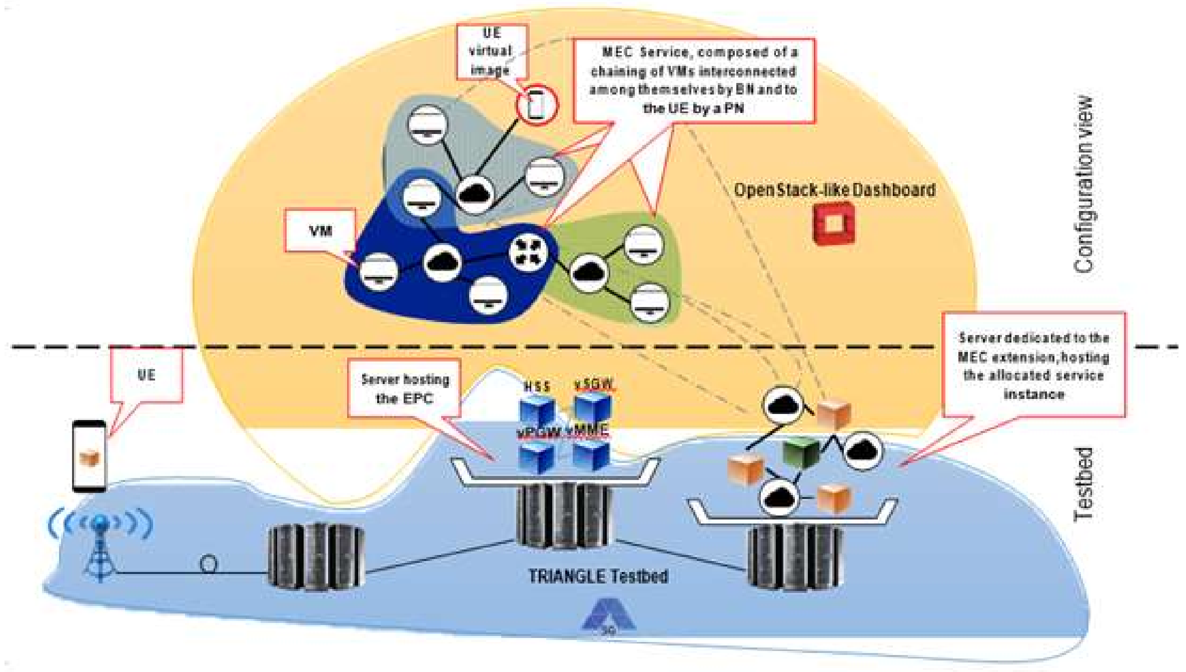

UE virtual image
MEC Service, composed of a chaining of VMs interconnected among themselves by BN and to the UE by a PN
OpenStack-like Dashboard
Configuration view
VM
UE
Server hosting the EPC
HSS
vSGW
vPGW
vMME
Server dedicated to the MEC extension, hosting the allocated service instance
Testbed
TRIANGLE Testbed

Figure 106 Deployment of MEC service instances in the TRIANGLE testbed.

In more detail, from the experimenters' point of view, there is a need to, on the one hand, upload the VMs composing the services and defining the service chain structure and requirements and, on the other hand, to design the test steps and execution. In order to support the tests, the testbed needs to provide additional capabilities integrated with the Radio Access Network (RAN) for traffic identification/isolation and management. The following sections provide technical details on the integration of the experimenters' interface and on the testbed, respectively.

The Annex 11 contains the user guide for MEC experimenters.

10.3.1 Test Creation and Deployment

Experimenters will deploy their tests in two phases:

- A Test Configuration phase, in which experimenters define their MEC services: upload their virtual machines, compose the service chain, and specify latency requirements and policies to be used during tests.
- A Test Orchestration phase, to define the test setup (e.g., number of PoPs, network latency among them and towards base-stations, etc.) and perform lifecycle operations, such as instantiate/de-instantiate, migrate, etc.

The service chain definition will be made available by means of a dashboard similar to the one provided by OpenStack [48]. Then, the service chain is instantiated and test steps can be defined and run in the TRIANGLE testbed by means of a Keysight Test Automation Platform (TAP) plugin. The testbed extension will enable the proper run of such tests and the management of the whole MEC service lifecycle. These two phases are described as follows:

Configuration Phase: DevStack

For the realization of the interface allowing experimenters to create their MEC service chains, DevStack [50], a GitHub-based deployment of OpenStack that can run in a VM, has been used. The user interface is provided by means of the Horizon dashboard.

Once connected to the dashboard, the experimenters will be able to upload their VMs, connect them to the UE PN and among themselves through BNs, and, if needed, to assign additional constraints to each VM. For example, in Figure 107, we can see the assignment of a "proximity class," which represents the allowed maximum distance (in terms of latency delay) from the UE.

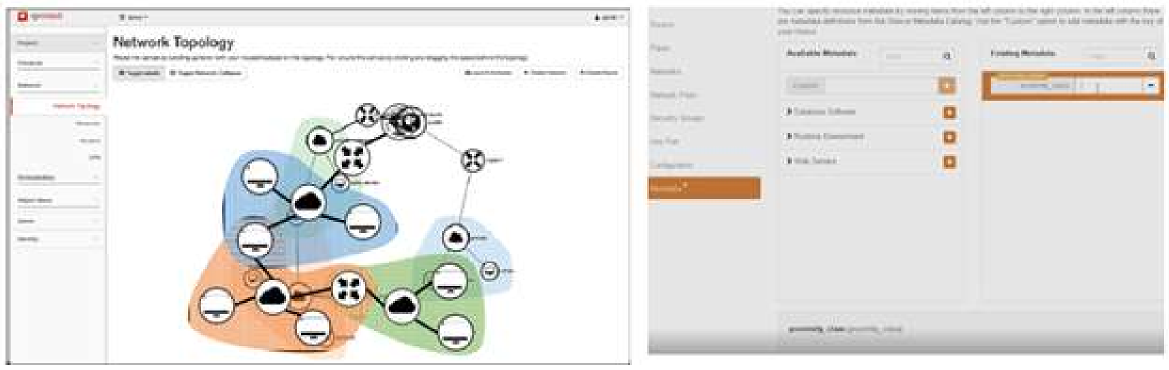


Figure 107 Scrrenshots of the OpenStack Horizon dashboard

This configuration phase is treated as a step in the TAP plugin and is managed by means of specific scripts that allow to refresh the user information after each test to prepare the system for another experimenter, as described in the next section.

Orchestration Phase: TAP

A Test Automation Platform (TAP) plugin for configuring and running the tests has been designed. In more detail, the plugin allows the experimenters to define the testbed configuration, e.g., the initial server and the service chain placement, and additional performance constraints, such as, for example, network latency between service and UE or towards base-stations. Then, the plugin is also used to perform the lifecycle operations on the service chains, such as instantiate/de-instantiate, migrate, etc., by means of TAP test steps. The plugin exposes to the experimenters some basic operations that are used to access the corresponding OpenStack APIs. In the current version, the TAP plugin developed for our extension allows to start, stop, suspend and resume individual VMs, along with providing the authentication management. Additional operations related to migration can be added in the case of more than one server made available in the TRIANGLE testbed for the MEC extension. As final operation, a script is provided to cleanup the system and make it available for the following experiment.

10.3.2 MEC Integration in the TRIANGLE Testbed Release 4

As highlighted by the ETSI MEC working group, the primary objective of the MEC technology is to dramatically reduce latency times throughout high service continuity levels [51]. To achieve this objective, IP packets need to be forwarded to the MEC applications meant to handle the traffic, which are typically hosted locally on the MEC platform or on a remote server. However, as the current 3GPP 4G architectural specification does not allow exposing its reference points externally, additional functionalities are required to foster the interaction between the edge and the mobile network.

In this respect, additional capabilities have been included to monitor the presence and the position of a specific UE, and intercept its traffic flowing from the Evolved Packet Core (EPC), and more specifically from the Packet Data Network Gateway (P-GW) to realize the MEC attach points at the SGi interface. This type of deployment is considered suitable for some of the 5G use cases, such as Mission Critical Push to Talk (MCPTT), and M2M communications [52].

The integrated functionalities are implemented in a single VM as shown in Figure 108. Bearers are defined per UE, including VLAN tags, and are managed by means of a REST interface. The Interception is in charge of intercepting messages from the SGi interface and parse the information needed to univocally recognize the UE, to identify the eNodeB where the UE is attached and handover events, as well as to understand the configuration of its bearers. If the intercepted packets do not belong to the service of interest, they are released without performing any further operations on them. Then, a Virtual Gateway is used to simply forward the packets to the corresponding service.

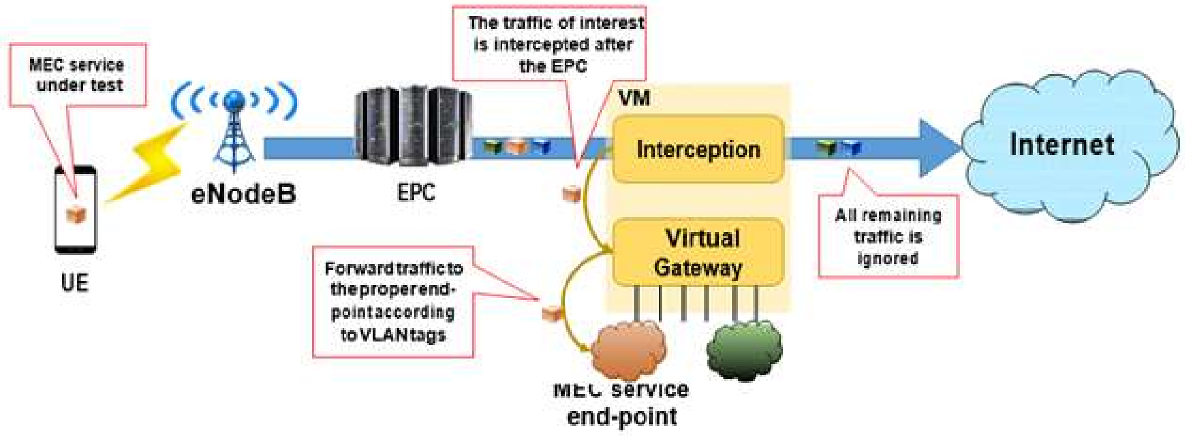

MEC service under test
UE
eNodeB
EPC
The traffic of interest is intercepted after the EPC
VM
Interception
Virtual Gateway
Internet
All remaining traffic is ignored
Forward traffic to the proper end-point according to VLAN tags
MEC service end-point

Figure 108 Integration of MEC functionalities in the TRIANGLE testbed

10.4 DANE (DASH-Aware Network Element) (TNO extension Release 3)

TNO’s extension adds a DANE (DASH-Aware Network Element) in the TRIANGLE testbed. The DANE is a recently standardised network element [41][42].whose aim is to provide network assistance to video streaming clients supporting the MPEG-DASH (or the 3GP-DASH) protocol [42][43]. Specifically, the DANE supports the SAND protocol [41][42], which enables it to provide quality-related assisting information to the streaming clients, asynchronously using Websockets. Providing this information to clients reduces the chance of freeze events and of frequent switches between low and high video streaming bitrates.

A DNS entry is configured for the DANE (“http://dane”), which runs on the “local servers” of the TRIANGLE testbed and implements the “Consistent QoS / QoE” profile defined in [42]. In particular, the DANE waits for Websocket connections from streaming clients on port 9000, and is enabled to receive a “SharedResourceAllocation” message from connected streaming clients, specifying a list of bitrate values corresponding to different bitrates in which a video stream can be provided by the streaming server. Based on information about the bandwidth available to the UE in the RAN, the DANE makes a decision with respect to which video stream bitrate the streaming client can support, and communicates this information to the client via a “SharedResourceAssignment” message. As the bandwidth available to the UE in the RAN changes, the DANE recalculates the corresponding video streaming bitrate that the client can support, and communicates it to the client again via a “Shared ResourceAssignment” message. Figure 109 pictures the DANE within the context of the TRIANGLE testbed, including a streaming server offering DASH content and a VR streaming application consuming it.

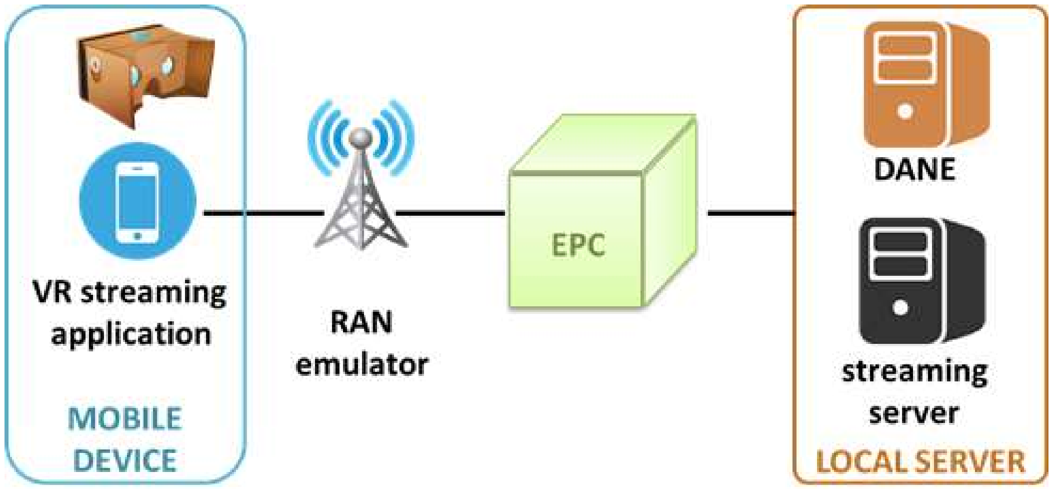
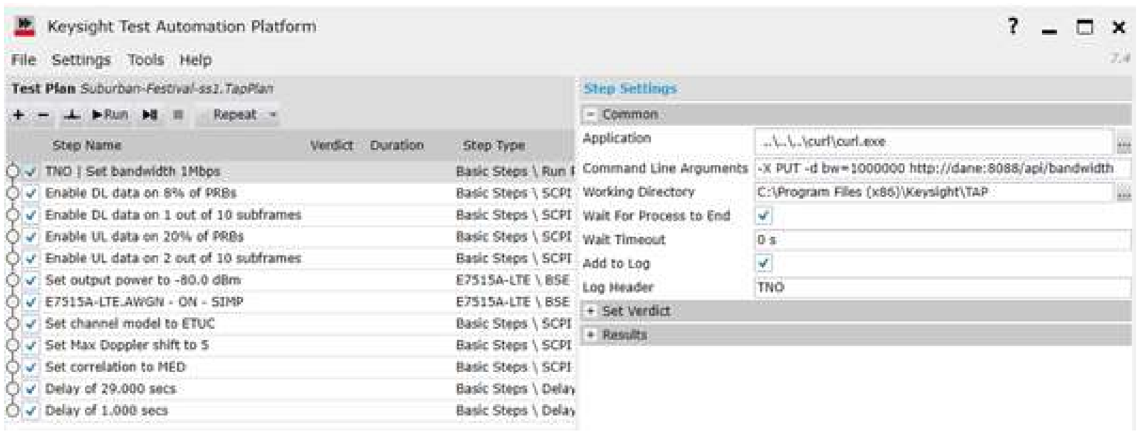


Figure 109 SAND architecture within the TRIANGLE testbed, including DANE, DASH streaming server and DASH VR app

As can be understood, the DANE needs to obtain an estimation of the bandwidth available to a UE in the RAN at any given moment. This mechanism is not currently described in any standard and it is left up to the eNodeB and DANE vendors’ discretion. In the TRIANGLE testbed, this mechanism is enabled as follows. The DANE exposes a REST API that can be used to set the bandwidth value. The REST API is invoked within a “test scenario” executed via the Core Sequencing Engine of TAP Figure 110. In fact, for each test scenario, the average bandwidth that the UE is expected to achieve has been previously computed via measurements with iperf.


Keysight Test Automation Platform

File Settings Tools Help

Test Plan *Suburban-Festival-ss1.TapPlan*

Step Name	Verdict	Duration	Step Type
✓ TNO | Set bandwidth 1Mbps			Basic Steps \ Run
✓ Enable DL data on 8% of PRBs			Basic Steps \ SCPI
✓ Enable DL data on 1 out of 10 subframes			Basic Steps \ SCPI
✓ Enable UL data on 20% of PRBs			Basic Steps \ SCPI
✓ Enable UL data on 2 out of 10 subframes			Basic Steps \ SCPI
✓ Set output power to -80.0 dBm			E7515A-LTE \ BSE
✓ E7515A-LTE.AWGN - ON - SIMP			E7515A-LTE \ BSE
✓ Set channel model to ETUC			Basic Steps \ SCPI
✓ Set Max Doppler shift to 5			Basic Steps \ SCPI
✓ Set correlation to MED			Basic Steps \ SCPI
✓ Delay of 29.000 secs			Basic Steps \ Delay
✓ Delay of 1.000 secs			Basic Steps \ Delay

Step Settings

– Common

Application	..\..\..\curl\curl.exe
Command Line Arguments	-X PUT -d bw=1000000 http://dane:8088/api/bandwidth
Working Directory	C:\Program Files (x86)\Keysight\TAP
Wait For Process to End	✓
Wait Timeout	0 s
Add to Log	✓
Log Header	TNO

+ Set Verdict

+ Results

Figure 110 Setting the DANE's bandwidth within a TAP's test scenario

10.4.1 Installation

The DANE software library can be installed by issuing the following command:

```
python setup.py install
```

10.4.2 Running

After installation, the DANE can be run by invoking the script installed:

```
dane
```

10.4.3 SAND support

We have implemented the following messages from the MPEG-SAND protocol:

Table 55 Messages from MPEG-SAND protocol

Device	**Message**	**Description**
Dane	DaneCapabilities	Informs the video client on the messages supported by this DANE.
	SharedResourceAssignment	Informs the video client on the operating point (i.e. DASH representation) allocated to it by the DANE.
Client	ClientCapabilities	Informs the DANE on the messages supported by this video client.
	SharedResourceAllocation	Informs the DANE that this client wishes to receive resource assignment. Furthermore, it informs the DANE of the operating points (i.e. DASH representations) it supports.

10.4.4 Protocol

The following diagram illustrates the operation of the SAND protocol, and the messages used to exchange bandwidth information.

```
+-----------+                                            +-----+
|DASH client|                                            |DANE |
+-----+-----+                                            +--+--+
      |                                                     |
      |                 WebSocket connect                   |
      +---------------------------------------------------> |
      |                                                     |
      |                 ClientCapabilities                  |
      +---------------------------------------------------> |
      |                                                     |
      |                 DaneCapabilities                    |
      | <---------------------------------------------------+
      |                                                     |
      |                 SharedResourceAllocation            |
      +---------------------------------------------------> |
      |                                                     |
      |                 SharedResourceAssignment            |
      | <---------------------------------------------------+
      |                                                     |
      +                                                     +
```

10.4.5 DANE REST API

The DANE REST API runs on port 8088.

Setting the available bandwidth

```
curl -X PUT -d bw=my_bandwidth http://dane:8088/api/bandwidth
```

Sets the available bandwidth to my_bandwidth.

Example:

```
curl -X PUT -d bw=12345 http://dane:8088/api/bandwidth
```

Getting the available bandwidth

```
curl -X GET http://dane:8088/api/bandwidth
```

Returns the available bandwidth as known by the DANE.

Resetting the DANE

```
curl -X POST http://dane:8088/api/reset
```

Resets the DANE state, i.e. disconnect all clients and reset the available bandwidth.

10.4.6 Dependencies

The DANE software depends on the following PyPI packages:

- pytz, for formatting datetime as requested in the SAND protocol.
- autobahn, a WebSocket library in Python.

10.4.7 Metrics Database

10.4.8 Introduction

Webserver with API for the storage and retrieval of logs created by a dash.js client. This server demonstrates how dash.js client metrics can be logged and retrieved. Note that the storage is in-memory, and will be lost on a restart/crash.

10.4.9 Running

Before running, install project dependencies by running:

```
npm install
```

The program can be run by the following command:

```
npm start
```

The exposed port can be configured by setting the PORT environment variable before running the program.

10.4.10 Docker

The docker image can be built by invoking:

```
docker build -t metricsdb .
```

10.4.11 REST API

Logging metrics data Metrics data can be logged by making a POST request in the following way:

```
curl -X POST http://dane:8081/{uuid}/MetricsData
```

where {uuid} should be an id which uniquely identifies a client corresponding to the metrics.

Logging quality data Quality data can be logged by making a POST request as follows:

```
curl -X POST http://dane:8081/{uuid}/QualityData
```

where {uuid} should be an id which uniquely identifies a client corresponding to the metrics.

Retrieving data Data can be retrieved by making a GET request to the same endpoint as used for logging data. The client uuids which are currently in the database can be obtained by making a GET request as follows:

```
curl -X GET http://dane:8081/ids
```

Dumping data The current state of the log can be dumped by making a POST request in the following way:

```
curl -X POST http://dane:8081/dump/{name}
```

where {name} should be a prefix of an indexed set of logs.

10.4.12 Metrics Visualisation

This service will display from a server hosting dash.js metrics at these endpoints:

- <url>/QualityData
- <url>/MetricsData

Data will be polled every second, and displayed on a graph.

10.4.13 Fake client swarm

10.4.14 Introduction

A simple service to launch and manage multiple SAND clients.
These clients use a weight of 5 by default. The default port is 10260.

10.4.15 Commands

In order to kill all clients, a POST request should be made as follows:

```
curl -X POST http://dane:10260/reset
```

Spawning new clients can be done by issuing a PUT request in the following way:

```
curl -X PUT http://dane/spawnClients/{count}
```

where {count} should be replaced by the desired number of new clients.

10.4.16 Obtaining logs

The logging server provides an overview of all log files at its root endpoint. The URL to view the logging directory is as follows: http://dane:10000/

When operating, the DANE logs its communication with the clients, and provides details on its bandwidth assignment algorithm. The log file can be accessed by navigating to the log service URL. From the TAP machine, this will be located at:

```
http://dane:10000/dane.log
```

Logged metrics can be dumped when the Metrics database service is running. Assuming the id 'test' was used to dump a metrics database for the first time, this file can be obtained at:

```
http://dane:10000/test-0.log
```

10.4.17 Default port assignments

The ports on the DANE VM are assigned as follows:

Service	Sub-service	Container port	Host port
DANE	SAND WebSocket	9000	90000
	REST	8088	8088
Fake SAND client	REST	10260	10260
Web client	HTTP	8080	8080
Metrics database	REST	3000	8081
Metrics viewer	HTTP	80	8082
Log web service	HTTP	10000	10000

10.5 Video quality-of-experience assessment extension (Streamowl extension Rel. 4)

The extension provides an Android application and the appropriate interfaces to assess the user-perceived quality of video streaming services in mobile devices. This is achieved by both analyzing the displayed video (in terms of the pixels of the captured frames on screen) or by analyzing the network traffic.

The concept of the extension is depicted in Figure 111: the mobile device requests the video from a video streaming service (e.g. YouTube, Netflix, BBC, etc.). The video is played back either via an embedded video player, which provides the required libraries (e.g. using the YouTube player API) for monitoring the video performance, or the native video player of the application/service is employed and the KQIs of the video streaming performance are extracted by processing the video packets at the network level using the StreamOwl quality monitoring probe and/or by capturing in the video screen the displayed video.

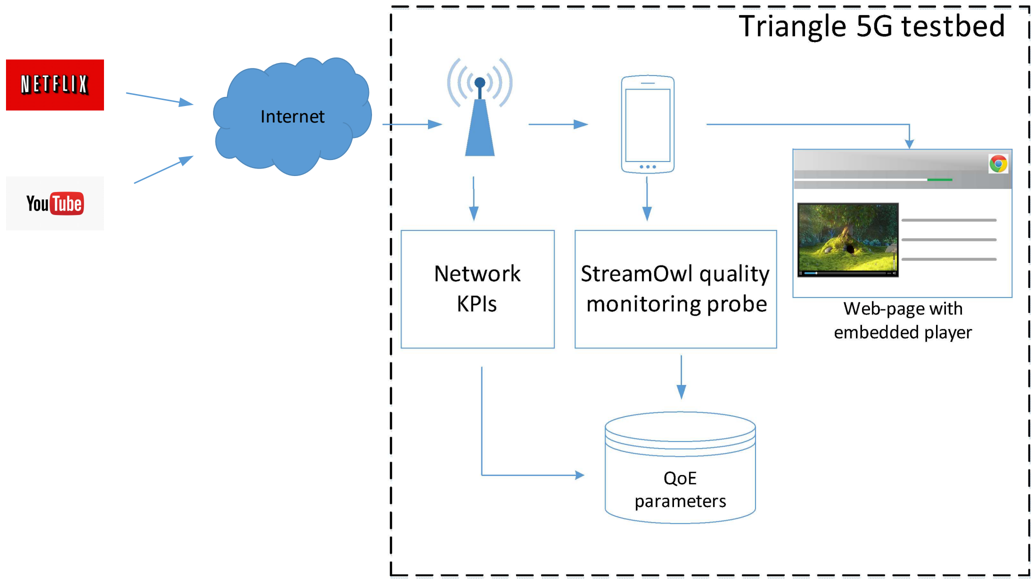


Figure 111: Concept of the extension for adaptive video streaming performance evaluation.

The Android App can be controlled in non-interactive fashion to perform automated tests using adb shell to initiate actions in the application. The following actions are supported through broadcast receiver message (which are also provided as individual steps in the provided TAP plugin):

- Start capturing of network video traffic: this function starts the packet capturing process to record all incoming and outgoing network traffic in a pcap file. A rooted device is required for capturing from the network interface. At the same time, the video screen is recorded, using the native display resolution of the mobile device, using a high bitrate (~20mbps). The adb command is:

  ```
  adb shell am broadcast -a
  com.example.streamowl.onlinevideotester.USER_START_RECORD
  ```

- Open the video player to play the requested video: in the app, we assume that the popular OmxPlayer is used to play the video, but any compatible video player can be used. The video player is employed to fetch the video from the requested source and display the video:

  ```
  adb shell am start -n com.mxtech.videoplayer.ad/.ActivityScreen -d
  http://playertest.longtailvideo.com/adaptive/oceans_aes/oceans_aes.m3u8
  ```

- Stop the video playback and stop capturing the network video traffic. In this step, the video player is stopped, the video recording is stopped, and two files are stored in the mobile device: a) the video recording which contains the video that was played back during the experiment, and the pcap file which contains the network packets of incoming and outgoing traffic. These two files will be used at the next step for the assessment of video quality.

adb shell am broadcast -a com.example.streamowl.onlinevideotester.USER_STOP_RECORD

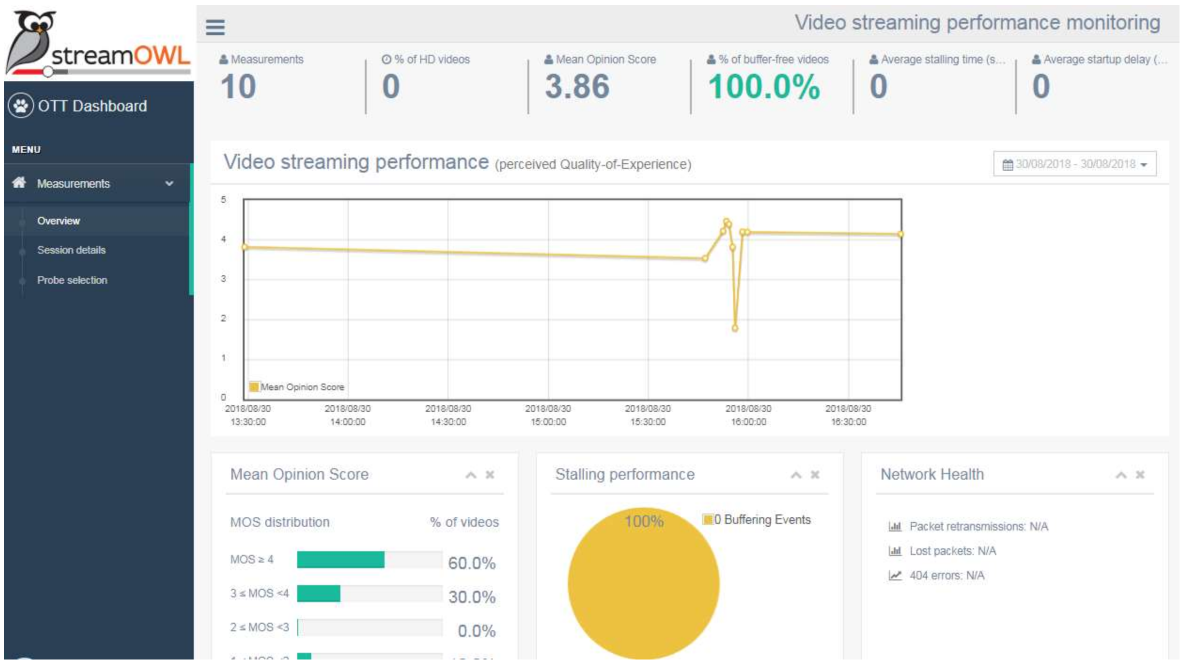
- The two files (avi video file and pcap file with network traffic) that are generated in the previous step are uploaded to a central remote ftp server (which can be configured via the settings of the app) for remote processing and estimation of the video quality.

  adb shell am broadcast -a com.example.streamowl.onlinevideotester.USER_FILE_UPLOAD

When new files are uploaded to the central database, the video quality estimation software is automatically called to estimate the video quality, using the following KPIs:

- Startup delay (or time-to-first-picture)
- Number of video stallings
- Total duration of video stallings
- Mean Opinion Score (MOS)

MOS is an holistic metric that reflects the overall user experience and his/her satisfaction from the video quality. The results are presented in a dashboard which provides intuitive and descriptive information about the quality of the streaming service, as depicted in the Figures below. Thus, the user of the platform can configure (either in the mobile device or the provided TAP plugin) the URL of the video stream that needs to be streamed, and each step can be followed independently of the others.



Video streaming performance monitoring

Measurements	% of HD videos	Mean Opinion Score	% of buffer-free videos	Average stalling time (s...	Average startup delay (...
10	0	3.86	100.0%	0	0

Video streaming performance (perceived Quality-of-Experience)

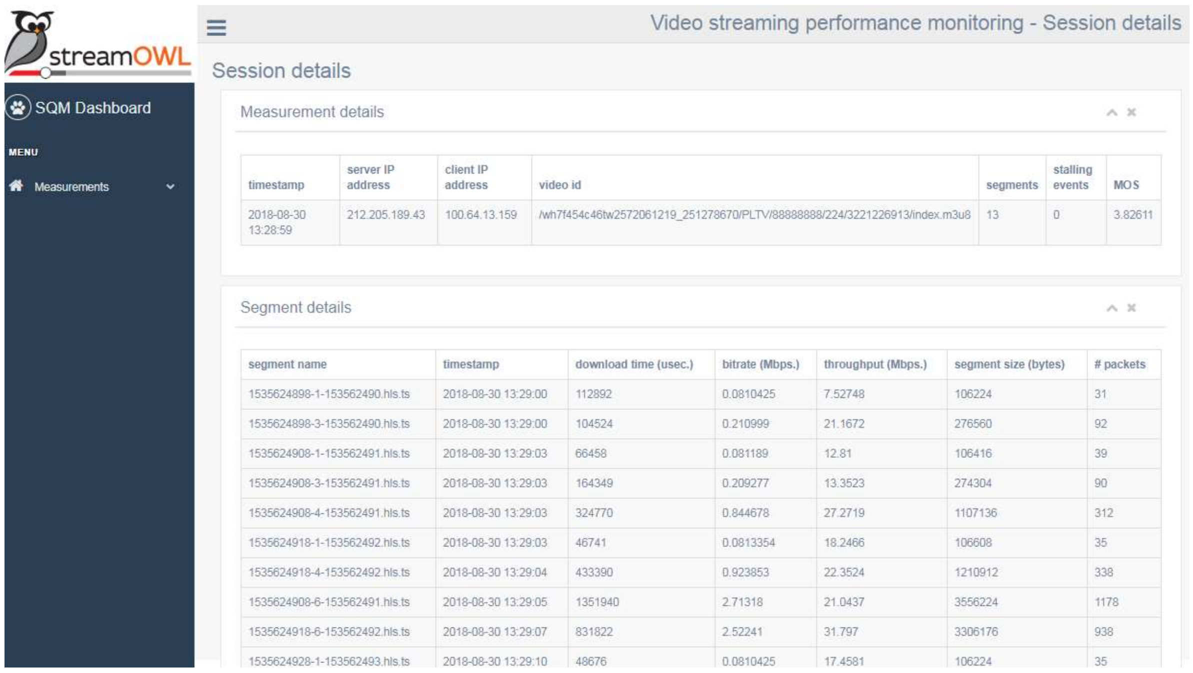
Mean Opinion Score

MOS distribution	% of videos
MOS ≥ 4	60.0%
3 ≤ MOS <4	30.0%
2 ≤ MOS <3	0.0%

Network Health

Packet retransmissions: N/A
Lost packets: N/A
404 errors: N/A

Figure 112: Dashboard for the video quality performance monitoring.



Session details

Measurement details

timestamp	server IP address	client IP address	video id	segments	stalling events	MOS
2018-08-30 13:28:59	212.205.189.43	100.64.13.159	/wh7f454c46tw2572061219_251278670/PLTV/88888888/224/3221226913/index.m3u8	13	0	3.82611

Segment details

segment name	timestamp	download time (usec.)	bitrate (Mbps.)	throughput (Mbps.)	segment size (bytes)	# packets
1535624898-1-153562490.hls.ts	2018-08-30 13:29:00	112892	0.0810425	7.52748	106224	31
1535624898-3-153562490.hls.ts	2018-08-30 13:29:00	104524	0.210999	21.1672	276560	92
1535624908-1-153562491.hls.ts	2018-08-30 13:29:03	66458	0.081189	12.81	106416	39
1535624908-3-153562491.hls.ts	2018-08-30 13:29:03	164349	0.209277	13.3523	274304	90
1535624908-4-153562491.hls.ts	2018-08-30 13:29:03	324770	0.844678	27.2719	1107136	312
1535624918-1-153562492.hls.ts	2018-08-30 13:29:03	46741	0.0813354	18.2466	106608	35
1535624918-4-153562492.hls.ts	2018-08-30 13:29:04	433390	0.923853	22.3524	1210912	338
1535624908-6-153562491.hls.ts	2018-08-30 13:29:05	1351940	2.71318	21.0437	3556224	1178
1535624918-6-153562492.hls.ts	2018-08-30 13:29:07	831822	2.52241	31.797	3306176	938
1535624928-1-153562493.hls.ts	2018-08-30 13:29:10	48676	0.0810425	17.4581	106224	35

Figure 113: Detailed session information of the quality monitoring dashboard.

11 Additional features for testing

11.1 GPS Emulation

Support for GPS Emulation is an incipient need. Making possible to run non-static scenarios would allow to test more realistic situations that the devices will have to face: continuous changes of location, speed, etc. In addition, there are many applications that require dynamic scenarios for testing correctly such as obviously those related to navigation and location systems, sport and fitness tracking apps, m-health applications, etc.

11.1.1 USRP

The main component of the GPS signal emulation system is the Universal Software Radio Peripheral (USRP), which is a particular software-defined radio (SDR) platform.

The USRP is a family of boards for radio software implementation, designed and sold by Ettus Research, a company that belongs to National Instruments. It was specially designed with the main purpose to provide an affordable family of hardware for the implementation of SDR systems. As it was designed to ease the developing of low-cost SDR applications, there are plenty of open-source tools that can be used to control the USRP such as the GNU Radio platform or free resources (libraries or schematics of the USRP boards) available in the official Ettus website.

Among other advantages, this hardware allows the design of RF applications from DC to 6GHz, including the possibility of developing multiple antenna (MIMO) systems. It also incorporates AD/DA converters, an interface for signals in RF and a FPGA which is responsible for the processing and conversion of the signal to different frequencies. After the signal has been processed and the data has been sampled by the FPGA, the information is sent to the computer via USB.

Specifically, the USRP used in TRIANGLE is the USRP B210. This platform belongs to the family of USRP Bus Series, and they are characterized by having high-speed USB 3.0 connection for streaming data to the host computer. Besides this, it also includes the AD9361 RFIC direct-conversion transceiver, which provides up to 56 MHz of real-time bandwidth, and an open and reprogrammable Spartan6 FPGA. With all these characteristics, it allows the configuration of 2 transmitters and 2 receivers (half or full duplex), to implement a coherent 2x2 MIMO system, and a modifiable ADC/DAC sampling rate of 12 bits. Figure 114 shows how to connect the RF output of USRP to the UE GPS antenna.

In addition, it includes the possibility of adding daughter boards that would expand its functionality and make it configurable for most of the signal spectrum.

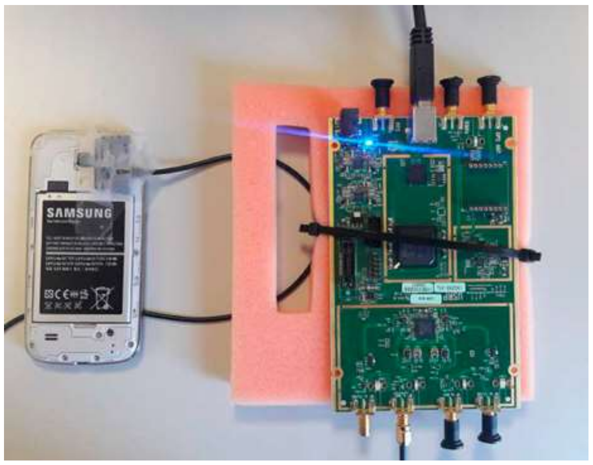


Figure 114 Connecting USRP output to the UE GPS antenna

GNU Radio

GNU Radio platform is a free and open-source software development toolkit that provides signal processing blocks to implement software-defined radio systems. Among other advantages, what makes GNU Radio special is that includes a graphical and friendly environment in which applications can be implemented by using these predefined blocks (filters, synchronization elements, equalizers, modulators and demodulators, encoders, decoders, etc.). Also, this platform provides mechanisms to connect and manage the communication and transmission of data between different processing blocks.

Another advantage of GNU Radio is that the implementation of the processing blocks themselves or more specific applications, could be programmed using Python by means of the use of GNU Radio's libraries, which are implemented in C ++.

Besides, using GNU Radio the USRP can be configured using the UHD driver, through which different parameters can be configured such as the choice of the antenna, transmission frequency, gain and also decimation and interpolation factors.

GNU Radio runs on Linux, Mac and Windows platforms and since it is an open-source tool, there are a huge number of applications already implemented and freely available on the Internet.

UHD (USRP Hardware Driver)

The UHD is the driver created by Ettus Research for application development on all USRP products. It also provides the mechanisms that make possible the interoperability between different USRP families. This driver, together with GNU Radio offers a simple interface to use it for the control of the USRP.

This driver is also based on an open-source software and it is available on Linux, Windows and MAC OS. The aim of UHD is to provide an API (Application Programming Interface) as well as the driver needed to control the USRP. UHD offers cross-platform support for multiple industry standard development environments and frameworks, including RFNoC, GNU Radio, LabVIEW and Matlab/Simulink, but it also offers a stand-alone mode (no operating system is required to run) through the API, programming directly on the UHD. For the stand-alone mode, it is important to know that both the driver and the firmware of the UHD are programmed in C/C++ whereas Verilog is the one used for the control of the FPGA.

Through UHD is how in this system USRP's parameters such as gain, transmission frequency, sample rate and the number of bits of the I/Q modulation can be modified.

11.1.2 GPS emulation with USRP

Software-Defined GPS Signal Simulator

Software-Defined GPS Signal Simulator (GPS-SDR-SIM) is an open-source programme available in the GitHub platform. Under MIT license, this software generates GPS baseband signal data streams, which can be converted to RF using software-defined radio platforms, such as bladeRF, HackRF, and USRP. Despite the word "simulator" in its name, what this tool actually does is not a simulation of a GPS signal but an emulation of it, it *really* generates a GPS signal which can be received by a GPS receiver.

In the first place, to generate a signal the user has to specify the GPS satellite constellation through a daily GPS broadcast ephemeris file (brdc). These files are available on the Internet (ftp://cddis.gsfc.nasa.gov/gnss/data/daily/) and updated every day.

Using these files, the program generates the calculations of the pseudo-distances and Doppler frequencies for the GPS satellites in view. These data are then used to generate digitized I/Q samples for the GPS signal in the L1 C/A band (the civil band), which are then converted into RF signals using SDR platforms that can provide a quadrature modulation in this band, such as the USRP in this case.

A great variety of parameters can be configured using this program in addition to the ephemeris file, which is a mandatory parameter. It also enables to specify the scenario date, the duration of the emulation, the possibility of signal emulation in static or dynamic mode, to set a specific sampling frequency of the USRP or the number of bits of the I/Q modulation, and even the option to skip the ionospheric delay that the code includes for spatial scenarios.

The following command in a Linux operating system, would use this program to emulate a signal in a static scenario with the parameters that are shown (the ephemeris specific file, coordinates, duration, sample rate and number of bits for modulation).

```
$ ./gps-sdr-sim -e efemerides1703.17n -l 50,30,15 -d 120 -s 2500000 -b 16
```

GPS-SDR-SIM-UHD

The repository includes a python script to transmit the samples to the USRP in a very simple way. This program uses GNU Radio to control the parameters of the USRP and allows the control of the board through the execution of a command line in which different options can be customised.

The program takes as an argument the simulation file created before and sends it to the USRP with the configuration required.

```
$ ./gps-sdr-sim-uhd.py -t gpssim.bin -f 1575420000 -x 0 -s 2500000 -b 16
```

In the command line above, the program takes the simulation file that had been created previously and sends it to the USRP with the GPS civil band transmission frequency, 0 dB gain which should be good enough to receive the signal, sample frequency of 2.5 MHz and 16 bits for the I/Q modulation. The last two parameters must be exactly those for the USRP B210, otherwise the GPS receptor would not receive the signal correctly.

KML to CSV

The format used to save customized routes created by Google Earth or Google Maps is a KML (Keyhole Markup Language) file. For this reason, a program that enables the use of this type of files on the GPS emulator has been implemented.

Introducing just the name of a specific route saved in a KML file, the program extracts the coordinates of the route (leaving aside the rest of the data this file contains) and makes possible the conversion into the appropriate format for the emulator to generate the signal simulation file.

This program also allows the configuration of the speed of the designed route for its emulation. What the program actually does, is the interpolation of the saved points of the route. The number of new points interpolated depends on the speed, (the slower it is the more number of new points are needed).

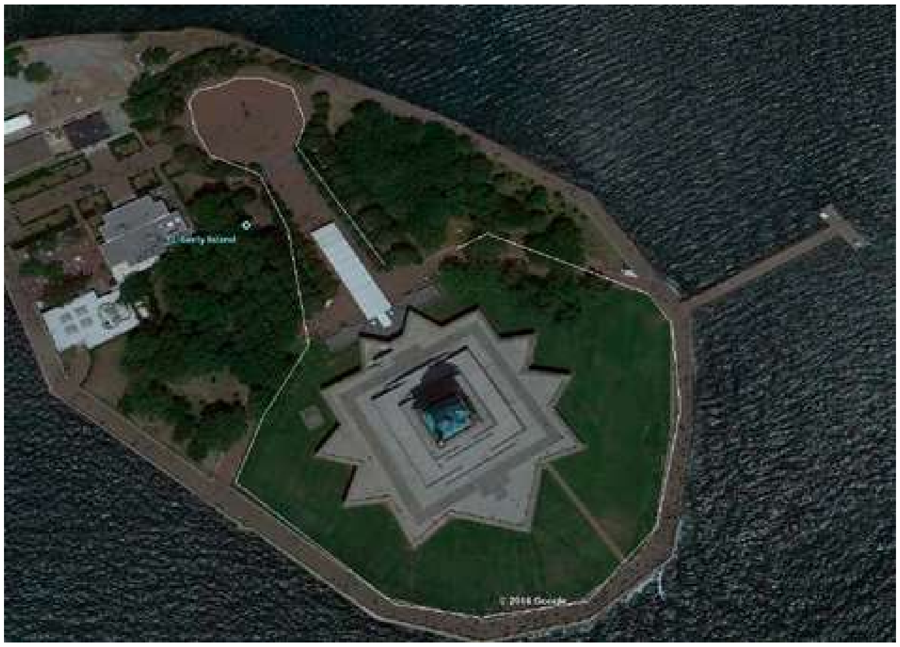
To calculate the interpolated points a spherical-Earth model has been used, ignoring ellipsoidal effects. This gives errors typically up to 0.3% which has been considered accurate enough for this purpose.

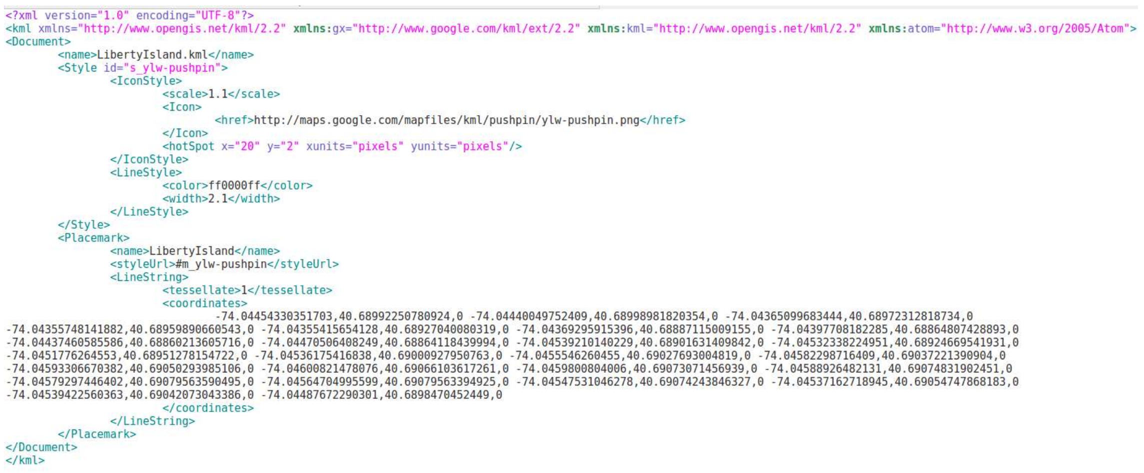
Besides this, it also enables to configure a time in which a fixed position will be emulated in order to settle the GPS signal in the receiver. The program will create the necessary number of points to achieve the time required, considering that for the emulator the sampling rate to read the user motion file (the csv file) has to be 10 Hz.

The next command would create a user motion file from a route called “seattleSTAR.kml”, with a speed of 10 m/s, and a settlement period of 8 seconds. The last parameter names the user motion file with a specific name.

```
$ ./kml2csv.py -k seattleSTAR.kml -s 8 -v 10 -o umFile
```

Figure 115 shows how to create a route and the resulting XML file.





```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">
<Document>
	<name>LibertyIsland.kml</name>
	<Style id="s_ylw-pushpin">
		<IconStyle>
			<scale>1.1</scale>
			<Icon>
				<href>http://maps.google.com/mapfiles/kml/pushpin/ylw-pushpin.png</href>
			</Icon>
			<hotSpot x="20" y="2" xunits="pixels" yunits="pixels"/>
		</IconStyle>
		<LineStyle>
			<color>ff0000ff</color>
			<width>2.1</width>
		</LineStyle>
	</Style>
	<Placemark>
		<name>LibertyIsland</name>
		<styleUrl>#m_ylw-pushpin</styleUrl>
		<LineString>
			<tessellate>1</tessellate>
			<coordinates>
				-74.04454330351703,40.68992250780924,0 -74.04440049752409,40.68998981820354,0 -74.04365099683444,40.68972312818734,0
-74.04355748141882,40.68959890660543,0 -74.04355415654128,40.68927040080319,0 -74.04369295915396,40.68887115009155,0 -74.04397708182285,40.68864807428893,0
-74.04437460585586,40.68860213605716,0 -74.04470506408249,40.68864118439994,0 -74.04539210140229,40.68901631409842,0 -74.04532338224951,40.68924669541931,0
-74.0451776264553,40.68951278154722,0 -74.04536175416838,40.69000927950763,0 -74.0455546260455,40.69027693004819,0 -74.04582298716409,40.69037221390904,0
-74.04593306670382,40.69050293985106,0 -74.04600821478076,40.69066103617261,0 -74.0459800804006,40.69073071456939,0 -74.04588926482131,40.69074831902451,0
-74.04579297446402,40.69079563590495,0 -74.04564704995599,40.69079563394925,0 -74.04547531046278,40.69074243846327,0 -74.04537162718945,40.69054747868183,0
-74.04539422560363,40.69042073043386,0 -74.04487672290301,40.6898470452449,0
			</coordinates>
		</LineString>
	</Placemark>
</Document>
</kml>
```

Figure 115 Google Maps route and XML file

11.2 Virtual Reality Applications Testing

Virtual Reality (VR) is a 5G use case identified in Work Package 2. We have developed in TRIANGLE a testing solution for VR applications on Android devices in order to close that gap between the testbed capabilities and the test specifications. VR is together with Augment Reality a showcase of 5G applications. To test that our VR testing framework works properly and provides meaningful information to app developers, we engaged with a VR application developer and thoroughly tested their application. This application was not mature enough to become an official TRIANGLE experimenter, yet it provided valuable insights to the TRIANGLE team to fine tune the VR testing solution. More details on VR testing will be shared in the upcoming Deliverable D5.6.

11.2.1 Requirements

The goal of VR applications is to emulate a natural and fluid interaction between the user and a virtual world, which will demand network resources. How far from natural and fluidity will determine the quality of experience perceived by VR users

When the VR is implemented on a mobile phone, the accelerometer and gyroscope are the components that provides the app a sense of movement, enabling users moving to discover the surrounding world by moving their heads.

Nowadays, and this may change in the near future, VR experience is fixed, meaning that users cannot just get up and walk around in order to discover the virtual world. Movement is implemented by the VR apps by tapping on the phone screen or the corresponding button on the HMD (Head-Mounted Display) host.

The KPIs of interest for this type of application will surely depend on the business logic to which the app belongs. However, relevant common KPI were identified in the D2.2 Appendix 4 [D2.2]. Table 56 is an extract from the Apps User Experience Test Specification (AUE).

Table 56 VR Application User Experience Key Performance Indicators

KPI	Definition

Time to load the virtual world	Time elapsed from selecting a scenario (world, experience, etc.) to loading the 3D visual context
Immersion Cut-off	Probability that successfully started immersion is ended by a cause other than the intentional termination by the user

In summary, a repetitive and automated test suite for measuring VR application has the following three requirements:

- Stimuli:
  - Rotating the device in the three-spatial axis for discovering the virtual world.
  - Emulating taps on the device screen for moving ahead.
- Responses:
  - Capturing the state of the application and visualizing certain content from the virtual world. This is required for measuring the KPI but also for automatically browsing the virtual environment to follow a given test script.

11.2.2 Architecture

Based on the requirements exposed in the previous section using a three servo motors mechanic platform is necessary for the stimuli. Additionally, an IR (Image Recognition) based solution is also necessary to capture the response from the VR app host (Android phone). Figure 116 shows the high-level architecture of the VR testing solution implemented in TRIANGLE.

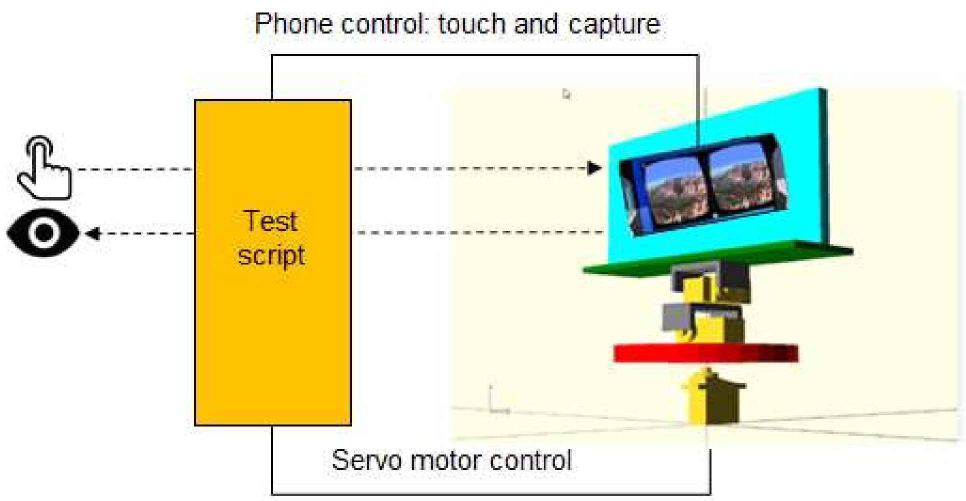


Figure 116 VR test module architecture

An important aspect to consider in this module is the accuracy of the servo motors because one of the common features of the VR apps is that they use a kind of visual aims so that users can select a menu option for browsing throughout the app.

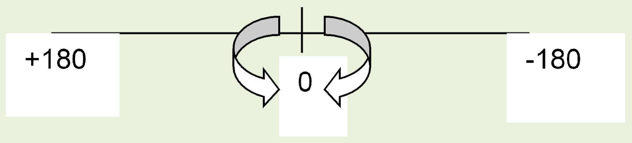
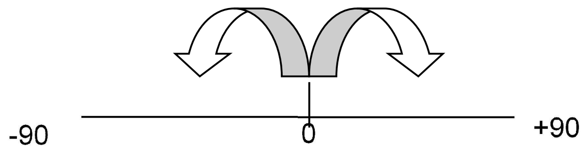
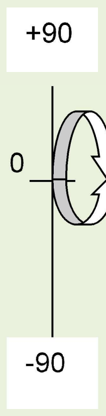
11.2.3 Robotic Arm

A robotic arm has been designed to support a device phone on top. This enables that the test script can force the phone to move in the three-spatial axis thus discovering the surrounding virtual world.

Three servo motors constitute the robotic arm for the three-spatial axis. The one place at the bottom rotates in the axis called “yaw”, the one in the body in the axis “roll” and the one at the bottom in the axis “pitch”.

Table 58 shows the position range and reference system used by the servos:

Table 57 Robotic arm position range and reference

Axis	Range
Yaw	
Roll	
Pitch	

A controller board drives the servo motors. This controller is connected to the test script host (the TRIANGLE test bed) via serial communication over USB. The commands dictionary basically just includes commands for setting and getting the position of the servo motors.

11.2.4 Controller Commands

Two commands are needed for the implementation of this module.

<u>Command 1</u>: Set position of the servo:

```
#<ch> P <pw> S <spd> ...# <ch> P <pw> S <spd> T <time><cr>
```

<ch>: Servo channel number, 0 - 23

<pw>: pulse width(us), 500 - 2500; the destination position

<spd>: single-channel speed (us/s)(Optional)

<cr>: carriage return, the symbol of the end, ASCII code 13 (Required)

<esc>: Cancel the current command, ASCI code 27

<u>Command 2</u>: Get position of the servo:

QP<ch><cr>

<ch>: Servo channel number, 0 - 23

This command returns the current pulse width of the servo in microseconds.

Auxiliary Commands

We have implemented some "utile" functions in order to facilitate further implementation of test scripts.

Position conversion degrees- µs

We have implemented a function to convert the magnitude of the position from pulse width (µs) into degrees. This way is much more intuitive for programming purposes.

The function to convert from µs to degrees is:

Angle (degrees) = (Central position (µs) – Angle (µs)) / Time to spin 180* in µs

The function to convert from degree to us is:

Angle (µs) = Central position (µs) – (Time to spin 180º (µs) * Angle (µs)) / 180

Table 58 shows the reference values needed in the conversion functions for the servos used in the TRIANGLE testbed:

Table 58 Robotic arm reference values

Axis	Central Position (µs)	Time to spin 180º (µs)
Yaw	1450	1700
Roll	1550	1650
Pitch	1450	1750

These values are obtained by calibration. More specifically, sending PUTTY commands to set the reference position and visually checking. This process must be repeated in case the servos are replaced.

Wait until reach position

As both native commands (get/set) are non-blocking, we have implemented a blocking function which waits until the servo has reached a set position.

Stop servos

There is no native command to stop the servos. For this reason, we have implemented a function to stop the servos. This will help for writing test scripts, for instance, when there is need to move the phone all around until it finds certain object in the virtual world. Basically, this function reads the current positions of the servo and right after sends the native command for setting that position.

11.2.5 Image Recognition

Image Recognition is used in this module for two main purposes:

1. Finding objects in the virtual world, which is the foremost importance for measuring the KPI of VR apps.
2. Browsing throughout the menus of the app where web driver-based technologies (Android UI Automation) does not work, i.e., GPU powered user interface.

We have used openCV (open Computer Vision) library as Image Recognition engine in TRIANGLE. This library implements functions for comparing images, more specifically object and its containers, providing a matching score (from 0 to 1). This totally meets the requirement of finding patterns (object) in the virtual world (container) and enables the implementation of the KPI Time to load virtual world / scene.

Figure 117 shows how openCV refers to the axis depending on the image orientation.

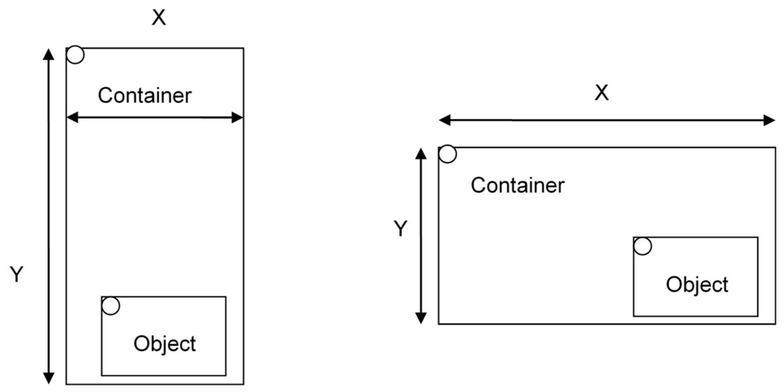


Figure 117 OpenCV system reference

From the comparison result the library provides the coordinates of the maximum matching point (circle inside the object in Figure 117). The openCV-based image matching process implemented in TRIANGLE is as described next.

The container image comes from a buffer provided by "minicap" library which runs on the phone and contains the phone screen (i.e., screen sharing) in near real time basis (see section 0). The buffer contains some header bytes, which provides metadata about the captured image (version, size, orientation, etc.). Based that information pointing the first byte of the image (coded in JPEG) is possible. openCV operations do not work on a specific image coding format. Rather, it uses a matrix format called Mat. Then, decoding the image buffer into this matrix is the first step. Then, both container and object images are grey scaled for finding the brightest point, which corresponds to the maximum matching point (using matchTemplate function).

openCV implements several alternative algorithms for the matching function: CV_TM_SQDIFF, CV_TM_SQDIFF_NORMED, CV_TM_CCORR, CV_TM_CCORR_NORMED, CV_TM_CCOEFF, and CV_TM_CCOEFF_NORMED.

In TRIANGLE we decided to use CV_TM_CCOEFF_NORMED because after experimentation turned to be the one with higher successful matching rate.

11.2.6 Interaction with the phone

In order to interact with the phone there are two alternatives: ADB (Android Device Bridge) and the high performance library “minitouch”.

ADB

ADB is a general-purpose command line tool for debugging Android phone via USB interface. We have used this tool to program from the test script tapping and swiping on the screen phone.

The response time of this tool since the test script sends the command until the tap really happens on the phone varies from 0.5 to 1 s approximately. This performance is sufficient for some test scripts operations such as browsing on the app menu or tapping on the screen to move ahead in the virtual world. However, there may be some operations to automate which require higher performance, for instance tapping at moving targets in VR shooter apps. Then, “minitouch” library (see section 0) is optionally required for use cases.

Additionally, ADB could be used for screen sharing by using its screen shooting command. However, the repose time of this function is very high, up to 3 s. Screen sharing is a mandatory performance requirement because the capture frame rate depends on it. Therefore, ADB has been discarded for this purpose and “minicap” library is mandatorily required for the implementation of the module.

Minitouch

Minitouch library provides a direct socket interface to Android phones for performing multi-touching and swiping operations. The response time is very high and sufficient for tapping moving targets. Experiments proves that this library performs response times around 50 ms and even lower.

Phone data usage

In order to pull out the data used by the phone while executing a VR app we use the information from the following file from the phone file system:

```
"cat /proc/net/xt_qtaguid/stats | grep -E \'nw_iface.* app_uid\'\"
```

Where the variables are:

nw_iface: Network interface

app_uid: this UID of the VR app under test.-

This file contains the following fields:

idx iface acct_tag_hex uid_tag_int cnt_set rx_bytes rx_packets tx_bytes

The data usage is counted in these two fields: rx_bytes and tx_bytes.

Minicap

The screen capturing rate (the number of screen captures per second) is the highest requirement of the software of the module. Minicap library provides a direct interface socket with the phone to get screen captures in a high rate. The library documentation claims up to 40 frames per second. This is sufficient to cover all foreseen testing scenarios in VR apps.

Minicap uses a virtual screen resolution. The screen captures are encoded in JPEG and scaled to that virtual resolution. This way the application (i.e., the TRIANGLE test script) does not need to care about the actual resolution of the phone. This is important because there are other coordinates reference systems in the module such as the openCV, the robotic arm and minitouch. All components of the module using the same coordinate’s reference system appeared a must in the implementation of this module.

11.2.7 Validation

We have used the reference VR App called “Google Cardboard” for the validation of this module. Figure 118 shows the validation scenario.

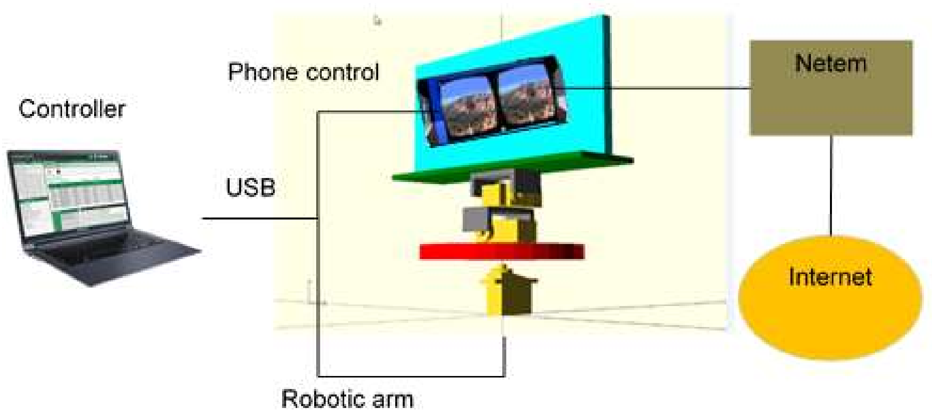


Figure 118 VR test solution validation

The test scripts run on the Controller host which uses all the components of the module: openCV, minicap, minitouch and the robotic arm. The Controller is connected to the platform via two USB cables, one for the phone (minicap, minitouch) and another for the robotic arm. The phone is connected to Internet with WLAN. In the network side, Netem is the software we have used to set the network conditions, in particular for bandwidth throttling. Netem runs on a host with two Gigabit Ethernet network interfaces and introduces the impartments on that link.

Figure 119 shows a photo taken in the lab during the validation of the module:

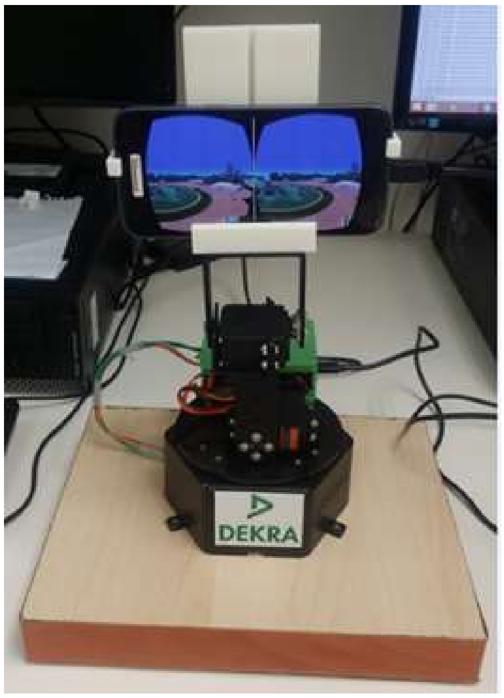


Figure 119 VR validation

The tests have been organized in two main groups.

The first group of the tests aims at determining the optimal configuration of the IR component. There is one a parameter in the openCV library configuration which determines how restrictive is the matching operation. The goal is to determine the matching threshold parameter to achieve the best balance between false matching (i.e., the object is not present but the IR matching operation returns true) and failed matching (i.e., the object is present but the IR matching operation returns false). In this procedure we have taken into account that, whenever the matching operation does not work properly, failed matching (which eventually would mean “KPI not reported”) is more desirable than false matching (which would mean “a false KPI value”). The later would introduce noise in further data analytics.

Table 59 compiles the results of the tests. The KPI in the table is the target KPI of the validation, i.e., time to load the virtual world.

Based on these results we have determined that the optimal matching threshold is 0.80.

Table 59 Validation results for determining matching threshold

Matching threshold	Average KPI (s)	Deviation KPI (s)	Failed matching Operations
0.75	8.27	1.46	10
0.76	9.17	3.40	8
0.77	9.00	1.87	3
0.78	9.38	1.85	6

0.79	10.65	4.18	9
0.80	9.36	2.68	4
0.81	9.46	1.30	7
0.82	10.95	2.44	6
0.83	10.50	1.63	10
0.84	12.39	4.05	10
0.85	11.75	3.40	6

Once determined the optimal configuration of the IR component we have implemented the test AUE/VR/001 specified in D2.2 to validate the VR test solution.

Table 59 compiles the results of the tests for different network bandwidths.

Table 60 AUE/VR/001 Validation results

Bandwidth (Mbit/s)	Average KPI (s)	Deviation KPI (s)	Failed matching operations
Unlimited	9.36	2.68	4
1	61.35	8.87	3
2	29.97	3.25	6
3	22.06	4.06	9
4	14.85	4.11	5
5	10.86	2.79	2
6	9.57	1.60	7

The data usage by the phone to load the virtual world has been 8 MB across all the network bandwidth configurations.

As observed in the results the IR library sometimes fails in the matching operation. Then this type of testing will require redundant test repetitions in order to get the necessary amount of KPI values.

11.3 Model-based testing extension

This section presents the extension of TRIANGLE testbed with model-based testing. Model-based testing is a testing technique that uses a model of the system under test to extract the test cases. In TRIANGLE, the model describes the interaction of the user and the mobile app, and the objective is to automatically produce a pool of app user flows that can be used in the test campaigns to activate (in many different ways) the app functionality required to evaluate a KPI.

The following subsections introduce the foundations of the model-based testing technique as well as the modeling and specification languages. Then, a methodology to (semi-) automatically extract the app model is presented. Finally, the model-based testing campaign is presented, which is the current implementation of the approach, accessible via the TRIANGLE portal.

11.3.1 Model-based testing foundations

Model-based testing techniques [37] use a model of the system under test for automatically generating test cases with an adequate coverage. In the TRIANGLE project, a model-based approach has been adopted to automatically construct app user flows (user interactions) that allow the evaluation of the app features using a set of KPIs.

In previous work [28][29][30], the tool MVE was constructed for automating the analysis of extra-functional properties on Android mobile apps, based on model-based testing and runtime verification techniques. The former was used to generate a large set of test cases from an app model provided by the app developer/tester, and then the latter analysed the executions of each test case for certain properties of interest. Model checking technique [32] has supported test cases generation and runtime verification, in particular SPIN model checker [31]. Model checking is a formal technique that exhaustively explores all the possible behaviours of a model to verify that a property is satisfied or not. On the one hand, the exhaustive exploration is used to produce the test cases, since they correspond with model behaviours. On the other, the analysis of properties has been used to detect if the application traces (associated to the execution of a test cases in the real device) satisfies or not the extra-functional property.

The main drawback of this approach is that a reasonably complete model of an app may generate thousands, if not millions, of user interactions, which are unfeasible to execute on a real mobile device. Furthermore, if a developer wants to test a specific feature of the app, we should be able to produce test cases on which the property can be analysed. Currently, to guarantee this, the model must be manually modified in order to include only the desired behaviours. The compositional nature of the app models is not enough to make this task easy, since the user could miss significant behaviours that contributed to the feature being tested while modifying the model.

In the TRIANGLE project, the previous approach has been extended in two different ways. First, the construction of the app model and the specific requirements are explicitly separated to produce meaningful test cases. In consequence, the reduction of the set of test cases happens during the generation process rather than in the modelling phase. Second, the generation phase uses user-defined requirements to guide the search of significant test cases. In this way, the app user flows constructed are guaranteed to satisfy the requirements, and the number of app user flows is reduced.

App model

The app under test is modelled using nested state machines [29]. By providing explicit models, a developer is able to define the realistic uses of the app, instead of generating random inputs to test it. An app state machine was composed of one or more view state machines, which corresponded to the different screens in the app. Each view state machine contained several nested state machines modelling different uses of the screen. The edges of a state machine represented the user actions, such as tapping a button or entering text that should be executed when traversing the edge.

Figure 120 shows the graphical representation of the model of the Universal Music Player sample app from the Android SDK, whose GUI can be seen in Figure 121. The app contains a list of songs classified by genre. The user can select the genre and the first song to play. Then, the app reproduces the list of songs in a loop starting from the selected one. The app plays music until the user exits or clicks the pause button. The app model is divided into two activities: one for selecting a song from genre playlists, called MainView, and other with a full screen player with the playback controls, called FullScreenView. The model in Figure 120 shows the two activities (MainView and

FullScreenView) and the possible user events (Play/Pause, back, etc.) that can happen during the app execution.

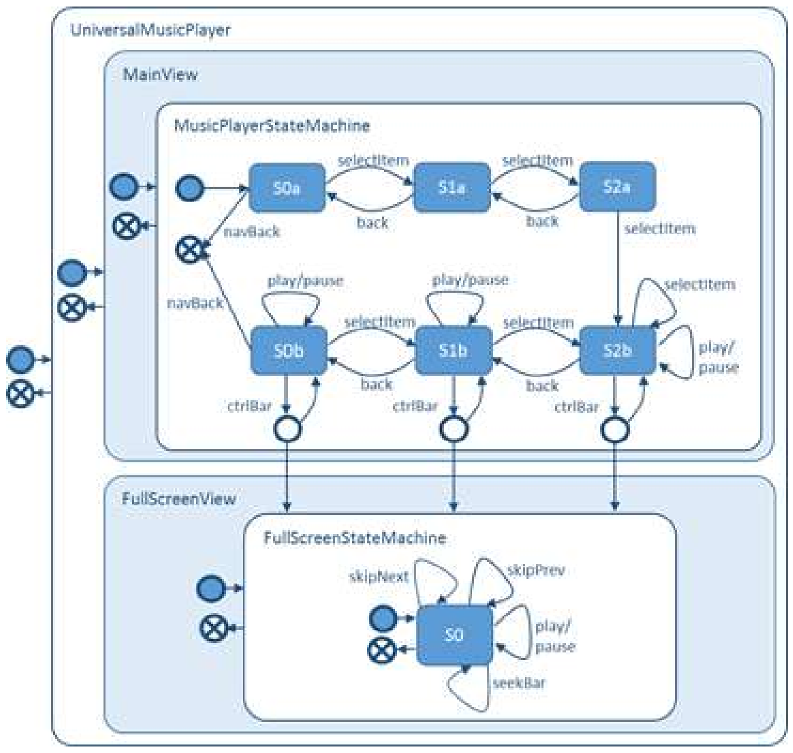


Figure 120 Universal Music Player model

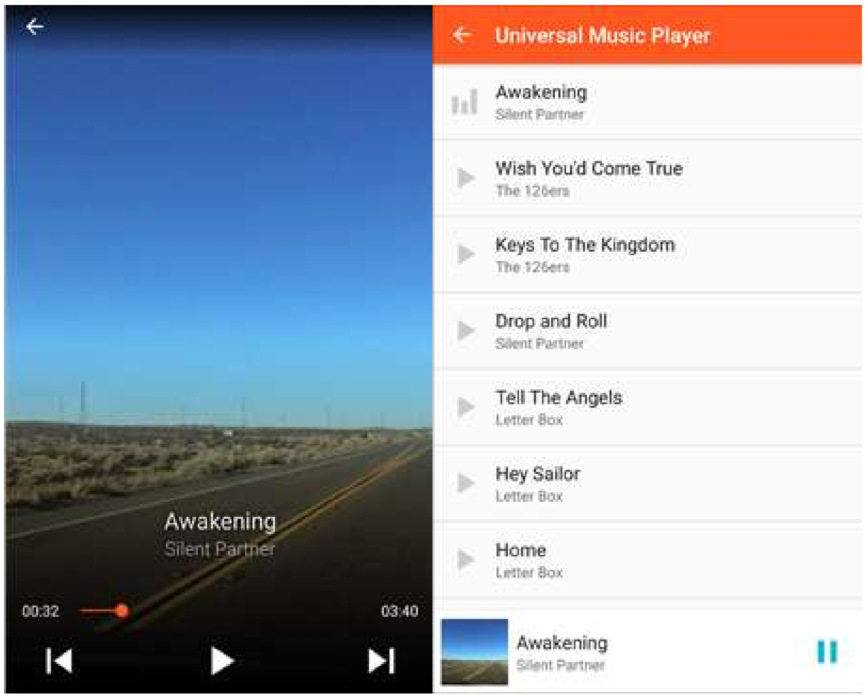


Figure 121 Universal Music Player GUI

An app user flow is defined as a sequence of user events that goes from an initial state to a final state of the app state machine. Thus, by exploring the model exhaustively, we were able to generate all possible app user flows.

To generate the app user flows, the implemented approach used a XML representation of the app model, instead of the graphical representation (see Figure 120). The format of XML model file will be presented in Section 11.3.2 (Model parse).

The XML file containing the app model is translated into a PROMELA specification [31]. We then used the SPIN [31] model checker to explore this specification exhaustively. When a valid end state was reached, we recorded the generated app user flow in a result file. These app user flows was then converted into Java programs that performed the flows on an actual Android device, using the UiAutomator API.
Figure 122 shows part of the PROMELA specification generated automatically from the app model in Figure 120. The state machines are translated in a single do loop, where each branch corresponds to a transition. For instance, the one in line 7 corresponds to the transition between states *S2a* and *S2b*.

```
DeviceType devices [ DEVICES ];
# define curBackstack devices [ device ]. Backstack
# define curState curBackstack . states [ curBackstack . index ]

proctype device_4107a7166c03af9b (int device ) {
  do
    :: curBackstack . index > -1 && curState == St_MainView_MusicPlayerSM_S2 ->
       // Event : selectItem
       transition (device , VIEW_MainView , 6);
        curState = St_MainView_MusicPlayerSM_S2b
      :: curBackstack . index > -1 && curState == St_MainView_MusicPlayerSM_S2b ->
        // Event : clickBack
        transition (device , VIEW_MainView , 7);
        curState = St_MusicPlayer_MainView_MusicPlayerSM_S1b
      :: curBackstack . index > -1 && curState == St_FullScreenView_FullScreenSM_init ->
        pushToBackstack (device , St_FullScreenView_FullScreenPlayerSM_init );
        transition (device , VIEW_FullScreenView , 0);
        curState = St_FullScreenPlayerView_FullScreenSM_S0
// ...
  od
}
```

Figure 122 Extract of PROMELA specification for test case generation

App User Flow Requirements

The TRIANGLE project defines the KPIs of interest that will be used to evaluate the features of mobile apps, and therefore, provides the requirements for the app user flows. These requirements are specified as a set of mandatory states and/or transitions of the app model that have to be reached, along with their execution order. For instance, in audio streaming applications, the main KPIs are the bit rate (related to audio quality), the buffering time (time spent waiting until music play starts or resumes), play length (amount of data streamed) and buffering ratio (waiting time over listening time). In addition, in mobile phones, energy consumption is also relevant, especially during playback. All these KPIs require that the app starts playing music, thus an essential requirement of the app user flows is starting music playback.

Since we use the exhaustive exploration of Spin, it is natural to describe the requirements as never claims [31]. The never claim is a special Spin process that executes synchronously with the system model, and checks whether a property holds. If it reaches the end state (its closing curly brace), Spin states that the property is violated and produces a counter-example, which in our case is interpreted as an app user flow that satisfies the requirements. Our methodology consists of translating all requirements into a never claim to make Spin generates the app user flows that satisfy them.

Moreover, the never claim can also be used to prune the state space explored, and thus reduce the time and resources required, since Spin backtracks and explores a different execution path when the never claim is blocked.

We apply our approach to obtain the app user flows of the Universal Music Player app. In this case study, we focus on the following requirements:

- The app eventually starts playing a song, which corresponds to reach state *S2b* of the app model.

- After that it eventually has to exit. This means that the app has to pass through states *S1b*, *S0b* and the end state of the state machine.
- The full screen activity is never launched.
- The app user flow cannot execute a transition more than once.

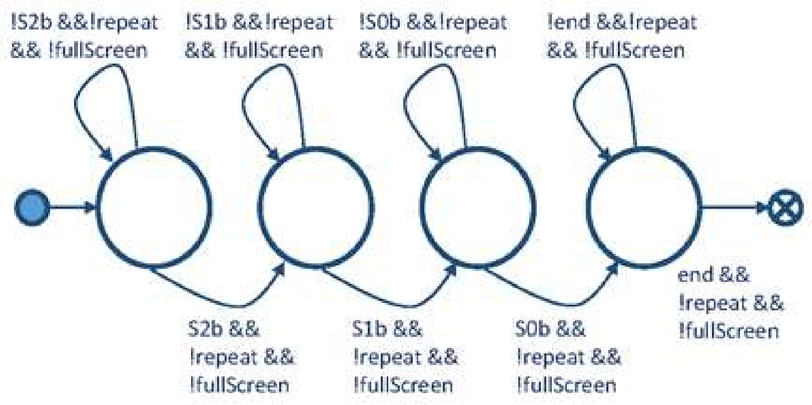

!S2b &&!repeat && !fullScreen
!S1b &&!repeat && !fullScreen
!S0b &&!repeat && !fullScreen
!end &&!repeat && !fullScreen
S2b && !repeat && !fullScreen
S1b && !repeat && !fullScreen
S0b && !repeat && !fullScreen
end && !repeat && !fullScreen

Figure 123 Pruning never claim as automaton

Figure 123 represents the never claim of the case study as an automaton. The label *!fullScreen* expresses that the full screen activity has been not visited, i.e. Spin never takes the branch in line 15 in Figure 122. Labels *S2b*, *!S2b* and so on specify that the corresponding state of the app model has been (respectively not) reached. This is checked when the branches are evaluated, e.g. in line 11. Finally, the label *!repeat* states that there are no repeated transitions. We have to define this requirement because in our PROMELA specification, Spin's global state contains the app user flow explored so far. Repeating a transition of the app model adds new actions to the app user flow and produces new states in Spin, thus the matching algorithm does not detect the repeated transition in the app model. Although it can see as a drawback, this behaviour allows us to describe other kinds of requirements that explicitly fire an event several times, which is very useful to discover behavioural errors of the app.

Note that each state of the automaton has two different transitions, one that links two states, and another that loops in the same state. The linking transitions are guarded with the requirements and tracks that they are satisfied in the correct order. When the automaton end state is reached, the corresponding app user flow is returned. A looping transition lets the execution of the app model advance while its guard condition is satisfied. When none of the transitions are enabled, Spin stops exploring the current path, pruning the search state space, as commented above. Therefore, the guard of a looping transition has to be disabled when the linking transition is enabled (to correctly track the requirements) and when the current app user flow is not interesting (to prune the search). For instance, in Figure 123, the looping transitions exclude the paths that have repeated transitions or activate the full screen activity.

Evaluation

We have carried out some experiments using Spin 6.4.6, with two different never claims: *pr.* and *no-pr.*, as well as without one. The *pr.* never claim prunes the search as we have just explained (see Figure 123). The *no-pr.* never claim differs from *pr.* in looping transitions, that are guarded by else instead of more restrictive conditions, such as the ones in Figure 120 Universal Music Player model. Table 61 shows the results. The first three rows use a maximum app user flow length (maximum number of app model transitions) of 10, and the bottom three rows use 20. The Flows column shows two values. The first one represents the number of app user flows that satisfy the requirements. The second one represents all the app user flows explored. Observe that for a maximum trace size of 10, the *no-pr.* never claim explores 85 traces, and only 18 are app user flows that satisfy the requirements. In contrast, the pr. never claim explores 20 traces and finds the same number of valid app user flows. This means that the use of the pr. never claim drastically reduces the time elapsed in the analysis. The difference between using the *pr.* and *no-pr.* never claims becomes more evident when the maximum length of app user flow increases.

Table 61 App user flow generation - Experiments

Max. len.	Never	Flows	Time	Memory	States
10	pr.	18/20	< 1s	9.5MB	1,059
10	no-pr.	18/85	55s	11.1MB	20,787
10	-	-/85	< 1s	10.9MB	20,787
20	pr.	20/22	< 1s	9.6MB	1,645
20	no-pr.	-/31,159	7.7h[a]	1.29 GB	13,226,035
20	-	-/18,303,632	50.7s	8.1 GB[b]	74,968,614

[a] Unfinished after 7.7 hours
[b] Unfinished after reaching memory limit of 8GB

For example, when using a maximum length of 20 and the *no-pr.* never claim, after more than seven hours and 31,159 different app user flows explored (most of them not satisfying the requirements), the analysis had still not finished. Therefore, our approach, which uses the *pr.* never claim to prune the search state space, greatly improves the performance of test case generation process. If we do not use a never claim at all, the generation of all possible app user flows is much faster, as seen in third and sixth rows. However, the developer does not know which ones satisfy the requirements. For instance, in the sixth row, the user ends up with millions of app user flows, which is not very useful in practice. It is clear that pruning the state space using requirements is still the best option.

11.3.2 Automatic App model extraction

In deliverable D3.2 "Progress report on the testing framework Release 2" we presented a model-based testing approach that given a model of the application under test, produces a set of app user flows that can be used during the test case execution. This approach has been enhanced by defining a methodology and implementing a tool to assist the app developer to generate the application model. Although the approach is independent of the device operating system, some tasks are explained targeting Android devices.

Figure 124 shows an overview of the approach, which is based on three main elements: (i) the app controller, (ii) the exploration algorithm, and (iii) the model parser. Given the application binaries, the app controller installs and controls the execution of the target app. The app controller can perform user events, and capture the hierarchy of visual elements. The exploration algorithm decides the order in which events are performed and determines whether the app is in a visited or new state after performing such events. Finally, the model parser transforms the states and transitions obtained from the exploration into the app model. The following sections explain each element in more detail.

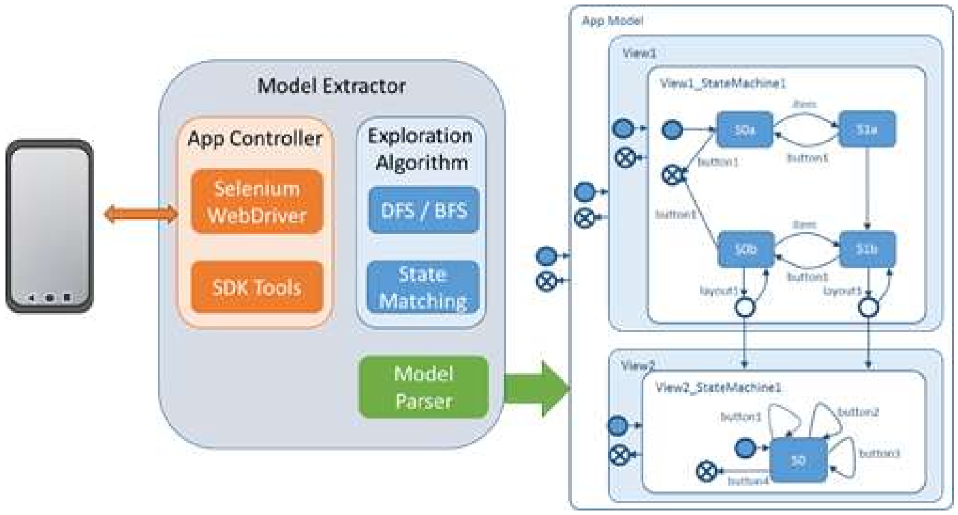


Figure 124 Model extraction overview

App controller

The app controller interacts with the device where the application is running. The app controller is responsible for the following tasks: 1) Install, launch and close the application, 2) obtain the hierarchy of visual elements of the active view, 3) determine the list of visual elements that accept user events, 4) perform user events on specific visual elements (e.g. click, long click or scroll), and 5) fire system events (e.g. open/close keyboard or play/pause media).

This element is closely related to the device's operating system, and thus its implementation may change depending on that. Currently, we have implemented an app controller for Android devices, which is based on two testing frameworks:

Quamotion WebDriver is a test automation framework that automates iOS and Android apps on real devices. WebDriver is an open protocol for test automation originally designed for web applications based on exchange of JSON messages.

Android UIAutomator [38] is a UI testing framework included in the Android SDK. In our approach, the main task of UIAutomator is to extract the Document Object Model (DOM) of the app; i.e., to obtain the hierarchy of visual elements. For each visible element, the DOM includes a list of attributes: the resource identifier, the class, and the user events accepted. This information is especially suitable to determine if the UI has changed after performing a user event, and to obtain the list of visual elements and events that must be explored.

Mobile applications have complex UIs with multiple activities, fragments, overlays, views, etc., which accept different types of user events. The exhaustive exploration of all visual elements that react to user events can lead, in the worst case, to large models with a worse performance in the generation phase of the app user flow. To manage the size of the application model, the app controller includes the following configurable options:

- Types of events considered in the exploration. For instance, if scrolling a layout produces the same effect as clicking on a tab menu, we can ignore the scrollable views and perform only click events.
- Number of list items explored. In some apps, the effect of performing an event on a list item is the same. For instance, in a music player with a list of playable songs, clicking on each song will start playback. Thus, exploring just some of the items is enough to extract the model.
- Ignored elements. Some events in specific elements could have non-desired effects during the model extraction or the test execution. For instance, elements that restore the account password or open the configuration settings. In this way, they are systematically excluded from exploration.
- Predefined text for specific EditText fields. This is relevant in those apps whose behaviour depends on the information provided in a form, for instance apps that require logging. Observe that most of the configurable options are related to task 3, obtaining the list of visual elements that will be explored. In this way, the exploration algorithm examines a reduced number of states. In contrast, the configuration of input text for EditText fields, is required to correctly explore all desirable app behaviours.

Exploration algorithm

The exploration algorithm defines a strategy to execute user events and traverse the different app states. A state represents the app UI after performing a user event on a specific element, that is; a state is a 3-tuple (*xPath, event, dom*), where *xPath* identifies the element in the source DOM, *event* is the event performed on the element, and *dom* is the DOM after the event. Transitions between states symbolise the execution of the event on the element stored in the target state.

We have an exploration algorithm with a depth-first search strategy. The main data structures are the set of *visited* states, the *path* (list) of states that leads to the current one, and the stack of *unvisited* states. While there are unvisited states, the algorithm extracts one and requests that the app controller performs the user event on the visual element, and stores the resulting DOM. Then, the algorithm checks whether a state in visited has an equivalent DOM. If so, the state is considered visited, and the app model is updated with a new transition from the parent state to the visited state and backtracks. Otherwise, the state is new, and it is included in visited. In addition, the app model is updated with the new state and the transition from its parent. If the node has successors, that is visual elements that can handle user events, they are included in the stack of unvisited states. Otherwise, the algorithm backtracks to a previous state.

Matching criterion:

The matching criterion defines when two DOMs can be considered equivalent. If we consider that two DOMs are equivalent when they are completely equal, the number of different states explored can be very large. Thus, the objective of the matching criterion is to reduce the number of different states, abstracting away specific content of the DOMs. The criterion is defined at different levels as follows:

- Two DOMs are equivalent if they are in the same activity, have the same hierarchy (hierarchy relation and number of nodes) and their nodes are equivalent.
- Two nodes are equivalent if the following attributes are equal: resource-id, class, package, checkable, clickable, enabled, scrollable, long-clickable.

- If the node is editable, the text attribute must be also equal. A node is editable if its class is android.widget.EditText or inherits from this class. This rule is required to generate models of forms.

Backtracking

The exploration algorithm must backtrack when the explored state has been previously visited or when it has no successors. The backtracking process consists in finding a state in *path* (from last to first) that has at least one unexplored successor state, i.e.; there is a successor state in *unvisited*. The backtracking process leaves this state in the last position of *path*. After that, the algorithm requests that the app controller closes the app, and performs the list of user events (on their corresponding elements) included in *path*. When the backtracking process ends, the app will be ready to accept the user event of the next unexplored state.

Model parse

The model parser produces the app model that will be used to generate the app user flows. The model parser acts when new states and/or transitions are added. The app model is described with the modelling language presented in [39], which was also introduced in deliverable D3.2. The language is based on nested state machines. The higher-level state machine represents the app. It can contain one or several state machines associated with the different activities of the app. At the lower level, the state machines represent the interaction of the user with the app. Transitions represent the user events performed in specific visual elements (buttons, layouts, etc.), and the source and target states symbolise the UI before and after the user event. There is a special type of state, called the connection state, used to transit between state machines. Connection states have two outgoing unlabelled transitions: one that points to a state of the same state machine (returning state), and another to the target state machine. When a connection state is reached, the app executes the target state machine, and when the target state machine reaches the final state, the app comes back to the returning state of the source state machine. Figure 125 shows the model of the Universal Music Player app.

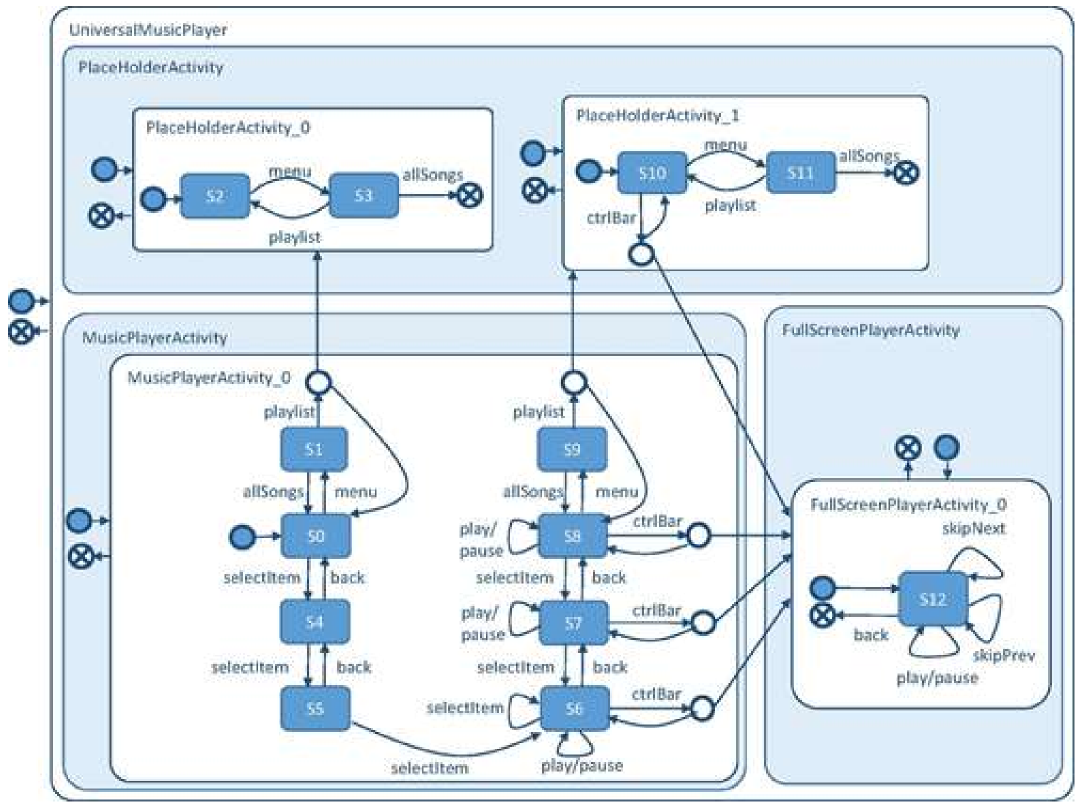


Figure 125 Universal Music Player model

In Figure 125, the app model describes the desired/undesired behaviours that the developer wishes to test. However, the app model depicts the possible interactions of the user. Although the modelling language is very expressive, the current version of the model parser has the following restrictions:

- Transitions between state machines of different views are allowed, but not transitions between state machines of the same view.
- Transitions are not labelled with temporal constraints.
- Transitions to other apps are not included in the model.
- Initial states do not represent the state of the UI. Thus, transitions from initial states are automatically fired, without any user event.

The resulting model is stored as a XML file that must be compliant with the XML schema shown in Figure 126.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:element name="StaticConfiguration">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Applications">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Application">
```

```
<xs:complexType>
 <xs:sequence>
  <xs:element name="Events">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="Event" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="name" type="xs:string"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="Views">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="View">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="StateMachines">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="StateMachine" maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="States">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="State" maxOccurs="unbounded">
                  <xs:complexType>
                   <xs:attribute name="name" type="xs:string"/>
                  </xs:complexType>
                 </xs:element>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
              <xs:element name="Transitions">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="Transition" maxOccurs="unbounded">
                  <xs:complexType>
                   <xs:attribute name="id" type="xs:int"/>
                   <xs:attribute name="next" type="xs:string"/>
                   <xs:attribute name="eventgroup" type="xs:string"/>
                   <xs:attribute name="type" type="xs:string"/>
                  </xs:complexType>
                 </xs:element>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
             <xs:attribute name="name" type="xs:string"/>
             <xs:attribute name="externalAccessibility" type="xs:string"/>
```

```
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                              <xs:attribute name="name" type="xs:string"/>
                              <xs:attribute name="externalAccessibility" type="xs:string"/>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="codeName" type="xs:string"/>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:attribute name="package" type="xs:string"/>
                  <xs:attribute name="externalAccessibility" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Devices">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Device">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="AssignedApplication">
                      <xs:complexType>
                        <xs:attribute name="xmlns:xsi" type="xs:string"/>
                        <xs:attribute name="xsi:type" type="xs:string"/>
                        <xs:attribute name="codeName" type="xs:string"/>
                        <xs:attribute name="viewPlace" type="xs:int"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="port" type="xs:int"/>
                  <xs:attribute name="path" type="xs:string"/>
                  <xs:attribute name="serialNumber" type="xs:string"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="xmlns" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 126 App model XML schema

Evaluation

To evaluate the model extraction process, we have chosen five different apps from Google Play Store and Android SDK: iDo Calculator, Kolab Notes, Topeka, Universal Music Player, and WordPress. The model extraction is a configurable process, which allows us to select the maximum depth explored (number of consecutive user events), number of list items explored, and list of elements excluded. In addition, if the app requires a form to be filled in, we can provide specific text input. Table 62 shows the different configurations for the evaluated apps, and Table 63, some results.

Table 62 Model extraction - Configuration

App	max depth	list items	events	el. excluded
iDo Calc.	5	1	click	0
UAMP	10	1	click	0
Kolab Notes	8	2	click	2
Topeka	15	1	click	0
Word Press	8	1	click	7

iDo Calculator[4] is a calculator with simple and scientific modes. Text input is performed by clicking independent buttons for each digit and arithmetic operation. The resulting app model considers the click on each calculator button as a different user event. Although all these events leave the app in the same state, the number of transitions is too high to produce the app user flows later. In this situation, we recommend manually modifying the model to abstract the buttons, for example by defining two types of buttons, numeric and arithmetic, and reducing the number of transitions.

Universal Music Player (UAMP)[5] is a sample app included in the Android SDK. It presents a list of songs, classified by Genre that can be reproduced. From the point of view of the app model, playing any of the songs has the same effect. Thus, we have configured the model extraction to explore just the first item of each list. It is worth mentioning a limitation of the UIAutomator dump process; it cannot obtain the DOM when the UI is changing dynamically, for instance if a video/song is playing. Thus, the application controller has to pause media playing before obtaining the DOM.

Kolab Notes[6] is an app for taking notes that can be local to, or shared by different devices using an account. The model only considers the local mode. In addition, we have excluded two elements from the exploration, because these elements show a colour picker that is currently difficult to handle. With respect to the text input, we fill in the note title and content with the same text in all executions. A note can be deleted, edited, etc., and these options are shown in a list, thus, we have set the number of list items explored to 2.

[4] Available at https://play.google.com/store/apps/details?id=com.ibox.calculators

[5] Available at https://github.com/googlesamples/android-UniversalMusicPlayer

[6] Available at https://play.google.com/store/apps/details?id=org.kore.kolabnotes.android

Topeka[7] is a Google sample for playing quizzes. Quiz questions are random, some of them are answered by choosing from a possible four answers, and others by writing text. Thus, it is difficult to systematically get the right answer. In addition, this issue can produce slightly different models and complicates the automatic execution of app user flows. Therefore, the app developer must provide us with a list of questions that are asked in the same order.

WordPress[8] is an app for visualising and managing WordPress sites. The app has an initial login form, thus we configure the user name and password with specific input text. In addition, the app provides links to recover user name and password, create a new account or read the Terms of Service. We have excluded these links from the exploration. Another peculiarity of this app, is that it suggests sites to visit. The list of sites changes dynamically, and furthermore, the sites can include different clickable elements. This situation is similar to the dynamic questions in Topeka. However, in order to produce a first version of the model, we have limited the number of sites explored by limiting the maximum depth.

Table 63 Model extraction - Results

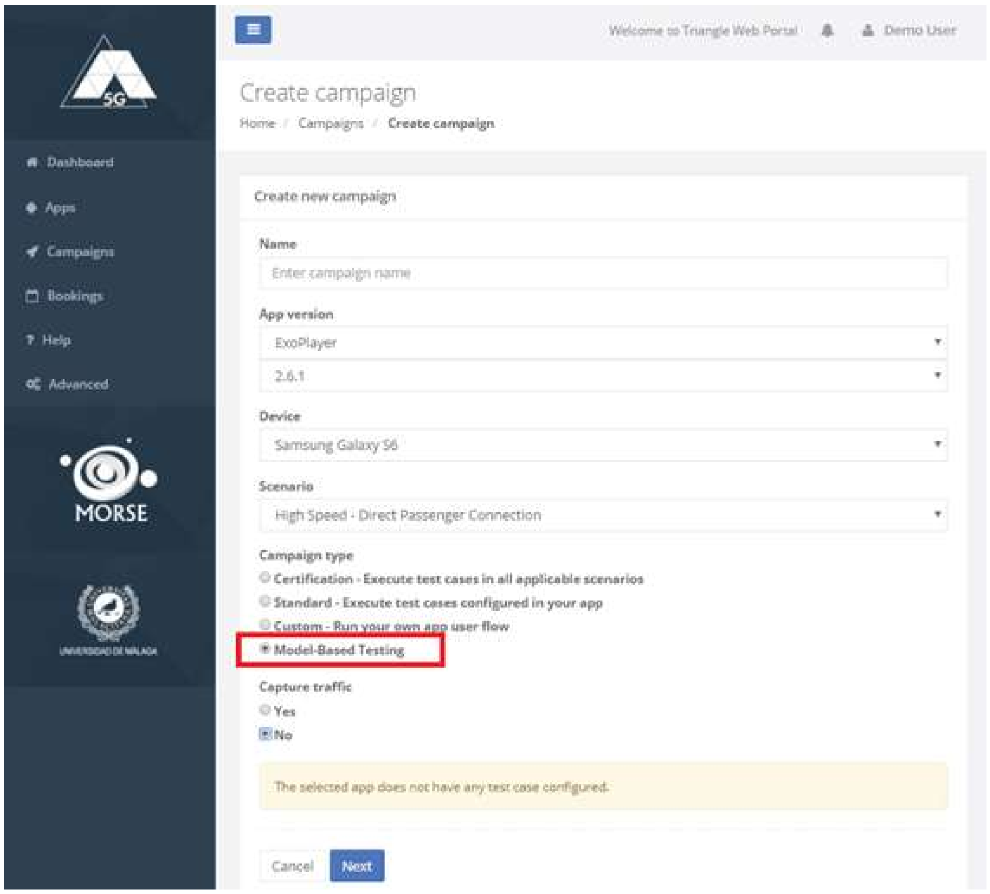
App	Activities	State machines	States	Transitions	Time (min)	App launches
iDo Calc.	2	2	9	93	49	23
UAMP	3	4	13	41	14	22
Kolab Notes	2	2	16	69	32	45
Topeka	3	3	16	51	33	38
Word Press	14	16	31	77	54	39

11.3.3 The model-based testing campaign

This section presents the integration of the model-based testing approach in the TRIANGLE framework. This new functionality can be used by means of a new type of campaign available in the TRIANGLE portal, called Model based Campaign (see Figure 127). To include this new campaign in the portal, it has been integrated in the testbed (and in the testbed workflow) the engine to automatically generate and select the app user flows to be executed during the campaign.

[7] Available at https://github.com/googlesamples/android-topeka

[8] Available at https://play.google.com/store/apps/details?id=org.wordpress.android



Create campaign

Home / Campaigns / Create campaign

Create new campaign

Name: Enter campaign name

App version: ExoPlayer, 2.6.1

Device: Samsung Galaxy S6

Scenario: High Speed - Direct Passenger Connection

Campaign type:
- Certification - Execute test cases in all applicable scenarios
- Standard - Execute test cases configured in your app
- Custom - Run your own app user flow
- [✓ Model-Based Testing]

Capture traffic: Yes [✓ No]

The selected app does not have any test case configured.

Figure 127 TRIANGLE portal - model based testing campaign

The complete framework, the integration of the model-based testing approach in TRIANGLE, and the preliminary evaluation of different apps (Universal Music Player and ExoPlayer) using the model-based testing approach are presented in [34], [35] and [36].

To run a model based testing campaign some previous steps have to be performed. First, the app developer has to upload to the portal the binaries of an instrumented version of the app. Thesesteps are common for all campaign types in the TRIANGLE testbed.

In addition, the app developer has to produce a model of the app that captures the different states of the app (GUI) based on the interactions with the app user. As presented in Section 11.3.1, the modelling language is based on nested state machines, which in practice is specified in an xml file. The model can be manually specified or automatically extracted using the approach described in Section 11.3.2. It is worth noting that the automatic approach has some limitations because it can only capture a subset of user interactions and GUI components.

Finally, the app developer has to define the requirement that the app user flows generated must satisfy. In Section 11.3.1, the requirements were defined with automata using the specification language of the underlying tool (the SPIN model checker) [33]. However, to simplify the notation and to abstract the app developer of the underlying tools, the TRIANGLE portal accepts the requirements in xml format. Figure 128 shows an example of requirement described in xml and the following table shows the complete XML schema. The requirements can restrict the number of times a user event is fired, the app screens/view visited, etc. The requirement is composed by the invariants and the sequence of constraints. The invariants have to be satisfied in the complete app user flow. For instance, in the example the invariant (Figure 128 line 3) defines the maximum number of play_pause events in the complete app user flow; that is, restricts how many times the app user can click in the play/pause button; and imposes a delay of 50 seconds between the execution of the play_pause event and the next event fired. In contrast, the constraints do not have to be satisfied in the complete trace. They must be satisfied in the order they are defined. In the example, the sequence of constraints defines that first, the event Dash_video_1 must be fire one time. Then, it is checked that the play_pause event has been fired two times, and finally, the end state of the SampleChooser activity must be visited, the app must enter and leave this activity. This requirement can generate multiple app user flows suitable to test the *play and pause features* of a content distribution app.

```
 1  <appUserFlowRequirement xmlns="appuserflowrequirement" name
        ="DemoPlayer_never_1">
 2    <invariants>
 3      <event name="play_pause" max="2" time ="50"/>
 4    </invariants>
 5    <sequence>
 6      <constraint type="simple">
 7        <event name="Dash_video_1" min="1" max="1"/>
 8      </constraint>
 9      <constraint type="simple">
10        <event name="play_pause" min="2" max="2"/>
11      </constraint>
12      <constraint type="simple">
13        <state name="end" view="SampleChooserActivity"
14          statemachine="SampleChooserActivity_0" visit="true"
                /> 
15      </constraint>
16    </sequence>
17  </appUserFlowRequirement>
```

Figure 128 Example of requirement in xml format

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           targetNamespace="http://www.morse.uma.es/appuserflowrequirement"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="appUserFlowRequirement">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="description" type="xs:string" />
   <xs:element name="application">
    <xs:complexType>
     <xs:attribute name="codeName" type="xs:string" use="required" />
    </xs:complexType>
   </xs:element>
   <xs:element name="invariants">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="loop">
       <xs:complexType>
```

```
        <xs:attribute name="min" type="xs:unsignedByte" use="required" />
        <xs:attribute name="max" type="xs:unsignedByte" use="required" />
       </xs:complexType>
      </xs:element>
      <xs:element name="event">
       <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required" />
        <xs:attribute name="max" type="xs:unsignedByte" use="required" />
       </xs:complexType>
      </xs:element>
      <xs:element name="view">
       <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required" />
        <xs:attribute name="visit" type="xs:boolean" use="required" />
       </xs:complexType>
      </xs:element>
      <xs:element name="state">
       <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required" />
        <xs:attribute name="view" type="xs:string" use="required" />
        <xs:attribute name="statemachine" type="xs:string" use="required" />
        <xs:attribute name="visit" type="xs:boolean" use="required" />
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="sequence">
    <xs:complexType>
     <xs:sequence>
      <xs:element maxOccurs="unbounded" name="constraint">
       <xs:complexType>
        <xs:sequence>
         <xs:element minOccurs="0" maxOccurs="unbounded" name="state">
          <xs:complexType>
           <xs:attribute name="name" type="xs:string" use="required" />
           <xs:attribute name="view" type="xs:string" use="required" />
           <xs:attribute name="statemachine" type="xs:string" use="required" />
           <xs:attribute name="visit" type="xs:boolean" use="required" />
          </xs:complexType>
         </xs:element>
         <xs:element minOccurs="0" name="event">
          <xs:complexType>
           <xs:attribute name="name" type="xs:string" use="required" />
           <xs:attribute name="max" type="xs:unsignedByte" use="required" />
           <xs:attribute name="time" type="xs:unsignedByte" use="optional" />
           <xs:attribute name="text" type="xs:string" use="optional" />
          </xs:complexType>
         </xs:element>
        </xs:sequence>
        <xs:attribute name="type" type="xs:string" use="required" />
       </xs:complexType>
      </xs:element>
```

```
          </xs:sequence>
        </xs:complexType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="name" type="xs:string" use="required" />
   </xs:complexType>
 </xs:element>
</xs:schema>
```

12 References

[1] Cisco Visual Networking Index: Forecast and Trends, 2017-2022

[2] Vagrant by HashiCorp: https://www.vagrantup.com/

[3] Bootstrap front-end framework: http://getbootstrap.com/

[4] The dummynet project: http://info.iet.unipi.it/~luigi/dummynet

[5] Android Debug Bridge: https://developer.android.com/studio/command-line/adb.html

[6] Logcat Command-line Tool: https://developer.android.com/studio/command-line/logcat.html

[7] OML Measurement Library: https://oml.mytestbed.net/projects/oml/wiki/

[8] D2.1 Initial report on the testing scenarios, requirements and use cases, Appendix 2

[9] http://www.keysight.com/en/pd-2372474-pn-E7515A/uxm-wireless-test-set?cc=US&lc=eng

[10] OpenStackClient," [Online]. Available: https://docs.openstack.org/python-openstackclient/latest/.

[11] "OpenStack API Documentation," [Online]. Available: https://developer.openstack.org/api-guide/quick-start/.

[12] "ETSI OSM," [Online]. Available: https://osm.etsi.org/.

[13] C. B. (ed.), "OSM White Paper - Release TWO Technical Overview," April 2017. [Online]. Available: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTWO-FINAL.pdf.

[14] "OSM Release TWO," [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Release_TWO.

[15] "RO Northbound Interface," [Online]. Available: https://osm.etsi.org/wikipub/index.php/RO_Northbound_Interface.

[16] "SO REST API (OSM RELEASE ONE)," [Online]. Available: https://osm.etsi.org/wikipub/images/2/24/Osm-r1-so-rest-api-guide.pdf.

[17] "Logs and troubleshooting (Release TWO)," [Online]. Available: https://osm.etsi.org/wikipub/index.php/Logs_and_troubleshooting_(Release_TWO).

[18] M. Broy, B. Jonsson, J.P. Katoen, M. Leucker, A. Pretschner. Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science). Springer-Verlag, Berlin, Heidelberg (2005)

[19] ITU-T P.501: Test signals for use in telephonometry

[20] ITU-T, "G.1030 Estimating end-to-end performance in IP networks for data applications"

[21] ITU-T, "G.1031 QoE factors in web-browsing"

[22] J. Nielsen, "Response Times: The Three Important Limits", from "Usability Engineering", 1993

[23] ISO/IEC 23009-1:2014 Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats

[24] Asterisk custom communications: http://www.asterisk.org/

[25] CSipSimple VoIP client: https://code.google.com/archive/p/csipsimple/

[26] ExoPlayer: https://google.github.io/ExoPlayer/

[27] GPAC Multimedia Open Source Project: https://gpac.wp.mines-telecom.fr/

[28] A. R. Espada, M. M. Gallardo, A. Salmerón, and P. Merino. Runtime verification of expected energy consumption in smartphones. In Proc. of the 22nd Int. Symposium on Model Checking Software, pages 132–149. Springer International Publishing, Aug. 2015.

[29] A. R. Espada, M. M. Gallardo, A. Salmerón, and P. Merino. Using model checking to generate test cases for android applications. In Proc. 10th Workshop on Model Based Testing, volume 180 of EPTCS, pages 7–21. Open Publishing Association, 2015.

[30] A. R. Espada, M. M. Gallardo, A. Salmerón, and P. Merino. Performance Analysis of Spotify® for Android with Model Based Testing. Mobile Information Systems, 2017:14, 2017.

[31] G. Holzmann. The SPIN Model Checker : Primer and Reference Manual. Addison-Wesley Professional, Sept. 2003.

[32] E. Clarke, O. Grumberg, and D. Peled. Model checking. Mit Press, 1999. Android: Testing UI for Multiple Apps. https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html

[33] L. Panizo, A. Salmerón, M. M. Gallardo, and P. Merino. Guided Test Case Generation for Mobile Apps in the TRIANGLE Project: Work in Progress. In Proc. of the 24th International SPIN Symposium on Model Checking of Software, pages 192-195. ACM, 2017.

[34] L. Panizo, A. Díaz, B. García, B.: An extension of TRIANGLE testbed with model-based testing. In: M.M. Gallardo, P. Merino (eds.) Model Checking Software, pp. 190-195. Springer International Publishing (2018)

[35] A.R. Espada, M.M. Gallardo, A. Salmerón, L. Panizo and P. Merino. A formal approach to automatically analyse extra-functional properties in mobile applications. Submitted to Journal Software Testing Verification and Reliability.

[36] L. Panizo, A. Díaz, B. García. A formal approach to automatically analyse extra-functional properties in mobile applications. Submitted to International Journal on Software Tools for Technology Transfer.

[37] M. Broy, B. Jonsson, J.P. Katoen, M. Leucker, A. Pretschner. Model-Based Testing of Reactive Systems: Advanced Lectures (Lecture Notes in Computer Science). Springer-Verlag, Berlin, Heidelberg (2005)

[38] Android: Testing UI for Multiple Apps. https://developer.android.com/training/testing/ui-testing/uiautomator-testing.html

[39] "Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)", ISO/IEC 23009-5:2017

[40] "python-docx" Library. Available: https://python-docx.readthedocs.io

[41] "Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)", ISO/IEC 23009-5:2017

[42] "Server and network assisted DASH for 3GPP Multimedia Services", ETSI 3GPP SA-170732, June 2017

[43] "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats", ISO/IEC 23009-1:2014/Amd. 1:2015/Cor.1:2015

[44] "Authentication and API request workflow," [Online]. Available: https://developer.openstack.org/api-guide/quick-start/api-quick-start.html#authentication-and-api-request-workflow.

[45] "OpenStack images," [Online]. Available: https://docs.openstack.org/image-guide/obtain-images.html.

[46] "Reference VNF and NS Descriptors (Release TWO)," [Online]. Available: https://osm.etsi.org/wikipub/index.php/Reference_VNF_and_NS_Descriptors_(Release_TWO.

[47] http://www.keysight.com/en/pd-1842303-pn-N6705B/dc-power-analyzer-modular-600-w-4-slots?pm=PL&nid=-35714.937221&cc=DK&lc=dan

[48] A. Abdelrazik, G. Bunce, K. Cacciatore, K. Hui, S. Mahankali, F. Van Rooyen, "Adding Speed and Agility to Virtualized Infrastructure with OpenStack," White Paper, Apr. 2015.

[49] R. Bruschi, G. Genovese, A. Iera, P. Lago, G. Lamanna, C. Lombardo, S. Mangialardi, "OpenStack Extension for Fog-Powered Personal Services Deployment", First International Workshop on Softwarized Infrastructures for 5G and Fog Computing (Soft5 2017), Genoa, Italy, September 2017.

[50] DevStack, https://wiki.openstack.org/wiki/DevStack.

[51] ETSI Group Report MEC 018, "End to End Mobility Aspects", version 1.1.1, October 2017.

[52] "MEC Deployments in 4G and Evolution Towards 5G", ETSI White Paper, February 2018

13 Annex 1: Portal Database

The Portal uses an SQL database as backend. This annex includes the current definition of the database tables, using SQL dialect supported by PostgreSQL.

```
CREATE TABLE users (
  id                  integer PRIMARY KEY,
  username            text UNIQUE NOT NULL,
  email               text UNIQUE NOT NULL,
  profiles            integer -- Flags: app developer, device maker, MNO, researcher
);

CREATE TABLE apps (
  id                  integer PRIMARY KEY,
  code                text UNIQUE NOT NULL,
  user_id             integer NOT NULL FOREIGN KEY REFERENCES users (id),
  name                text NOT NULL,
  os                  integer -- Enum: Android, iOS, Other
);

CREATE TABLE app_versions (
  id                  integer PRIMARY KEY,
  app_id              integer NOT NULL FOREIGN KEY REFERENCES apps (id),
  version             text NOT NULL,
  version_code        text,
  file                bytea,
  CONSTRAINT unique_version UNIQUE (app_id, version)
);

CREATE TABLE app_user_flows (
  id                  integer PRIMARY KEY,
  app_id              integer NOT NULL FOREIGN KEY REFERENCES apps (id),
  kpi_id              integer NOT NULL FOREIGN KEY REFERENCES kpis (id),
  name                text NOT NULL,
  script              bytea NOT NULL
);

CREATE TABLE kpis (
  id                  integer PRIMARY KEY,
  name                text NOT NULL,
  description         text,
  kpi_type            integer -- Enum: app, device...
);

CREATE TABLE kpi_marks (
  id                  integer PRIMARY KEY,
  kpi_id              integer NOT NULL FOREIGN KEY REFERENCES kpis (id),
  name                text NOT NULL,
  description         text,
  order               integer,
  allowed_types       integer, -- Flags: user action, UI element, UI element property,
internal
  internal_help       text
);
```

```
CREATE TABLE kpi_mark_supports (
  id                  integer PRIMARY KEY,
  app_user_flow_id    integer NOT NULL FOREIGN KEY REFERENCES app_user_flows (id),
  kpi_mark_id         integer NOT NULL FOREIGN KEY REFERENCES kpi_marks (id),
  support_type        integer NOT NULL, -- Enum: user action, UI element, UI element
property, internal
  user_action         integer,
  ui_element          text,
  ui_element_property    text
);

CREATE TABLE scenarios (
  id                  integer PRIMARY KEY,
  name                string UNIQUE NOT NULL
);

CREATE TABLE test_configurations (
  id                  integer PRIMARY KEY,
  scenario_id         integer NOT NULL FOREIGN KEY REFERENCES scenarios (id),
  name                string UNIQUE NOT NULL
)

CREATE TABLE devices (
  id                  integer PRIMARY KEY,
  string              name UNIQUE NOT NULL,
  owner_id            integer FOREIGN KEY REFERENCES users (id),
  testbed             boolean NOT NULL,
  os                  integer -- Enum: Android, iOS, Other
);

CREATE TABLE campaigns (
  id                  integer PRIMARY KEY,
  owner_id            integer NOT NULL FOREIGN KEY REFERENCES users (id),
  name                text NOT NULL,
  certification       boolean NOT NULL,
  campaign_type       integer, -- Enum: app, device, mno, researcher
  repeat              integer NOT NULL,
  state               integer -- Enum: created, scheduled, running, finished
);

CREATE TABLE campaign_app_versions (
  id                  integer PRIMARY KEY,
  campaign_id         integer NOT NULL FOREIGN KEY REFERENCES campaigns(id),
  app_version_id      integer NOT NULL FOREIGN KEY REFERENCES app_versions(id),
  CONSTRAINT unique_campaign_app_version UNIQUE (campaign_id, app_version_id)
)

CREATE TABLE campaign_kpis (
  id                  integer PRIMARY KEY,
  campaign_id         integer NOT NULL FOREIGN KEY REFERENCES campaigns (id),
  kpi_id              integer NOT NULL FOREIGN KEY REFERENCES kpis (id),
  app_user_flow_id    integer FOREIGN KEY REFERENCES app_user_flows (id),
  CONSTRAINT unique_support UNIQUE (campaign_id, kpi_id)
);
```

```
CREATE TABLE campaign_scenarios (
  id                  integer PRIMARY KEY,
  campaign_id         integer NOT NULL FOREIGN KEY REFERENCES campaigns (id),
  scenario_id         integer NOT NULL FOREIGN KEY REFERENCES scenarios (id),
  CONSTRAINT unique_scenario UNIQUE (campaign_id, scenario_id)
);

CREATE TABLE campagin_devices (
  id                  integer PRIMARY KEY,
  campaign_id         integer NOT NULL FOREIGN KEY REFERENCES campaigns (id),
  device_id           integer NOT NULL FOREIGN KEY REFERENCES devices (id),
  CONSTRAINT unique_device UNIQUE (campaign_id, device_id)
);

CREATE TABLE experiments (
  id                  integer PRIMARY KEY,
  campaign_id         integer NOT NULL FOREIGN KEY REFERENCES campaigns (id),
  order               integer,
  app_version_id      integer FOREIGN KEY REFERENCES app_versions (id),
  kpi_id              integer FOREIGN KEY REFERENCES kpis (id),
  app_user_flow_id    integer FOREIGN KEY REFERENCES app_user_flows (id),
  scenario_id         integer FOREIGN KEY REFERENCES scenarios (id),
  device_id           integer FOREIGN KEY REFERENCES devices (id),
  oml_database        text
);
```

14 Annex 2: Portal API REST

This annex describes the API REST provided by the Portal to access the test campaign information available at the Portal.

14.1 Devices

List all devices

```
GET /v1/devices
```

Response

```
Status: 200 OK
---
[
  {
    "id": 1,
    "name": "Samsung Galaxy S4",
    "testbed": false,
    "os": "android",
    "device_id": "AD001",
    "url": "https://[host]/v1/devices/1",
    "user_url": "https://[host]/v1/users/1"
  }
]
```

Get a single device

```
GET /v1/devices/:id
```

Response

```
Status: 200 OK
---
{
  "id": 1,
  "name": "Samsung Galaxy S4",
  "testbed": false,
  "os": "android",
  "device_id": "AD001",
  "url": "https://[host]/v1/devices/1",
  "user_url": "https://[host]/v1/users/1"
}
```

14.2 Users

List all users

```
GET /v1/users
```

Response

```
Status: 200 OK
---
[{
  "id": 1,
  "email": "john.doe@triangle-project.eu",
  "name": "John Doe",
  "testbed": true,
  "campaign_notification": true,
  "url": "https://[host]/v1/users/1"
}]
```

Get a single user

```
GET /v1/users/:id
```

Response

```
Status: 200 OK
---
{
  "id": 1,
  "email": "john.doe@triangle-project.eu",
  "name": "John Doe",
  "testbed": true,
  "campaign_notification": true,
  "url": "https://[host]/v1/users/1"
}
```

14.3 Apps

List all apps

```
GET /v1/apps
```

Response

```
Status: 200 OK
---
[
  {
    "id": 1,
    "code": "com.triangle.portal_demo",
    "name": "Triangle Web Portal Demo",
    "os": "android",
    "url": "https://[host]/v1/apps/1",
    "user_url": "https://[host]/v1/users/1",
    "app_versions": [
      {
        "id": 1,
        "version": "5",
        "version_name": "1.0.4",
        "url": "https://[host]/v1/versions/1"
      }
    ]
  }
]
```

Get a single app

```
GET /v1/apps/:id
```

Response

```
Status: 200 OK
---
{
  "id": 1,
  "code": "com.triangle.portal_demo",
  "name": "Triangle Web Portal Demo",
  "os": "android",
  "url": "https://[host]/v1/apps/1",
  "user_url": "https://[host]/v1/users/1",
  "app_versions": [
    {
      "id": 1,
      "version": "5",
      "version_name": "1.0.4",
      "url": "https://[host]/v1/versions/1"
    }
  ]
}
```

14.4 Features

List all features

```
GET /v1/features
```

Response

```
Status: 200 OK
---
[
  {
    "id": 2,
    "name": "Media file playback",
    "description": "Media file playback",
    "feature_type": "application",
    "use_case": {
      "id": 5,
      "name": "Content Distribution Straming Services",
      "description": "Content Distribution Straming Services"
    },
    "url": "https://[host]/v1/features/2"
  },
  {
    "id": 3,
    "name": "Download media content for offline playing",
    "description": "Download media content for offline playing",
    "feature_type": "application",
    "use_case": {
      "id": 5,
      "name": "Content Distribution Straming Services",
      "description": "Content Distribution Straming Services"
    },
    "url": "https://[host]/v1/features/5"
  }
]
```

Get a single feature

```
GET /v1/features/:id
```

Response

```
Status: 200 OK
---
{
  "id": 2,
  "name": "Media file playback",
  "description": "Media file playback",
  "feature_type": "application",
  "use_case": {
    "id": 5,
    "name": "Content Distribution Straming Services",
    "description": "Content Distribution Straming Services"
  },
  "url": "https://[host]/v1/features/2"
}
```

14.5 Test cases

List all test cases

```
GET /v1/test_cases
```

Response

```
Status: 200 OK
---
[
  {
    "id": 1,
    "name": "Non-Interactive Playback",
    "description": "Measure the UX KPIs while playing a media file.",
    "features": [
      {
        "id": 2,
        "name": "Media file playback",
        "description": "Media file playback",
        "url": "https://[host]/v1/features/2"
    ],
    "url": "https://[host]/v1/test_cases/1"
  },
  {
    "id": 5,
    "name": "Broadcast live video",
    "description": "Measure the capability of broadcasting live content.",
    "features": [
      {
        "id": 5,
        "name": "Broadcast live video",
        "description": "Broadcast live video",
        "url": "https://[host]/v1/features/5"
    ],
    "url": "https://[host]/v1/test_cases/5"
  }
]
```

Get a single test case

```
GET /v1/test_cases/:id
```

Response

```
Status: 200 OK
---
{
  "id": 1,
  "name": "Non-Interactive Playback",
  "description": "Measure the UX KPIs while playing a media file.",
  "features": [
    {
      "id": 2,
      "name": "Media file playback",
      "description": "Media file playback",
      "url": "https://[host]/v1/features/2"
  ],
  "url": "https://[host]/v1/test_cases/1"
}
```

14.6 Scenarios

List all scenarios

```
GET /v1/scenarios
```

Response

```
Status: 200 OK
---
[
  {
    "id": 1,
    "name": "Urban - Pedestrian",
    "url": "https://[host]/v1/scenarios/1",
  }
]
```

Get a single scenario

```
GET /v1/scenarios/:id
```

Response

```
Status: 200 OK
---
{
  "id": 1,
  "name": "Urban - Pedestrian",
  "url": "https://[host]/v1/scenarios/1",
}
```

14.7 Campaigns

List all the campaigns in the application

15 Annex 3: Measurements points (Instrumentation library)

15.1 Common Services

Login

- App Initialization Start - Login Required
  `eu.TRIANGLE_project.appinstr.co.Login.appInitializationStartLoginRequired()`
  Generated message:
  `Co\tLogin\tApp Initialization Start - Login Required`

- App Initialization Start - Login Not Required
  `eu.TRIANGLE_project.appinstr.co.Login.appInitializationStartLoginNotRequired()`
  Generated message:
  `Co\tLogin\tApp Initialization Start - Login Not Required`

- App Started
  `eu.TRIANGLE_project.appinstr.co.Login.appStarted()`
  Generated message:
  `Co\tLogin\tApp Started`

Menu Navigation

- Start Menu Navigation
  `eu.TRIANGLE_project.appinstr.co.MenuNavigation.startMenuNavigation()`
  Generated message:
  `Co\tNavigation\tStart Menu Navigation`

- Menu Navigation - App Ready
  `eu.TRIANGLE_project.appinstr.co.MenuNavigation.menuNavigationAppReady(<success>)`
  Generated message:
  `Co\tNavigation\tMenu Navigation - App Ready\t<boolean success>`

15.2 Content Distribution Streaming Services

Media File Playback

- Media File Playback - Start
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackStart()`
  Generated message:
  `Cs\tPlayback\tMedia File Playback – Start`

- Media File Playback - End
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackEnd()`
  Generated message:
  `Cs\tPlayback\tMedia File Playback – End`

- Media File Playback - First Picture
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackFirstPicture()`
  Generated message:

`Cs\tPlayback\tMedia File Playback - First Picture`

- Media File Playback - Video Resolution
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackVideoResolution(<resolution_x>, <resolution_y>)`
  Generated message:
  `Cs\tPlayback\tMedia File Playback - Video Resolution\t<int resolution_x>\t<int resolution_y>`

- Media File Playback - Content Stall Start
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackContentStallStart()`
  Generated message:
  `Cs\tPlayback\tMedia File Playback - Content Stall Start`

- Media File Playback - Content Stall End
  `eu.TRIANGLE_project.appinstr.cs.MediaFilePlayback.mediaFilePlaybackContentStallEnd()`
  Generated message:
  `Cs\tPlayback\tMedia File Playback - Content Stall End`

Play and Pause

- Media File Playback - Pause
  `eu.TRIANGLE_project.appinstr.cs.PlayAndPause.mediaFilePlaybackPause(<success>)`
  Generated message:
  `Cs\tPlayPause\tMedia File Playback - Pause\t<boolean success>`

- Media File Playback - Resume
  `eu.TRIANGLE_project.appinstr.cs.PlayAndPause.mediaFilePlaybackResume(<success>)`
  Generated message:
  `Cs\tPlayPause\tMedia File Playback - Resume\t<boolean success>`

Stop and Replay

- Media File Playback - Stop
  `eu.TRIANGLE_project.appinstr.cs.StopAndReplay.mediaFilePlaybackStop(<success>)`
  Generated message:
  `Cs\tStopReplay\tMedia File Playback - Stop\t<boolean success>`

Search and Seek

- Media File Playback - Search
  `eu.TRIANGLE_project.appinstr.cs.SearchAndSeek.mediaFilePlaybackSearch(<success>)`
  Generated message:
  `Cs\tSearchSeek\tMedia File Playback - Search\t<boolean success>`

- Media File Playback - Seek
  `eu.TRIANGLE_project.appinstr.cs.SearchAndSeek.mediaFilePlaybackSeek(<success>)`
  Generated message:
  `Cs\tSearchSeek\tMedia File Playback - Seek\t<boolean success>`

- Media File Playback - First Search Result
  `eu.TRIANGLE_project.appinstr.cs.SearchAndSeek.mediaFilePlaybackFirstSearchResult()`
  Generated message:
  `Cs\tSearchSeek\tMedia File Playback - First Search Result`

Rewind and Fast Forward

- Media File Playback - Rewind
  `eu.TRIANGLE_project.appinstr.cs.RewindandFastForward.mediaFilePlaybackRewind(<success>)`
  Generated message:
  `Cs\tRewindFF\tMedia File Playback - Rewind\t<boolean success>`

- Media File Playback - Fast Forward
  `eu.TRIANGLE_project.appinstr.cs.RewindandFastForward.mediaFilePlaybackFastForward(<success>)`
  Generated message:
  `Cs\tRewindFF\tMedia File Playback - Fast Forward\t<boolean success>`

Playlist Skip Forward and Backwards

- Playlist - Skip Forward
  `eu.TRIANGLE_project.appinstr.cs.PlaylistSkipForwardandBackward.playlistSkipForward(<success>)`
  Generated message:
  `Cs\tSkipFwBw\tPlaylist - Skip Forward\t<boolean success>`

- Playlist - Skip Backwards
  `eu.TRIANGLE_project.appinstr.cs.PlaylistSkipForwardandBackward.playlistSkipBackwards(<success>)`
  Generated message:
  `Cs\tSkipFwBw\tPlaylist - Skip Backwards\t<boolean success>`

Download Media Content for Offline Playing

- Media Content Download - Start
  `eu.TRIANGLE_project.appinstr.cs.DownloadMediaContentForOfflinePlaying.mediaContentDownloadStart()`
  Generated message:
  `Cs\tDownloadMedia\tMedia Content Download - Start`

- Media Content Download - End
  `eu.TRIANGLE_project.appinstr.cs.DownloadMediaContentForOfflinePlaying.mediaContentDownloadEnd(<success>)`
  Generated message: `Cs\tDownloadMedia\tMedia Content Download - End\t<boolean success>`

15.3 Live Streaming Services

Live Video Playback

- Live Video Playback - Start
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.liveVideoPlaybackStart()`
  Generated message:
  `Ls\tLivePlayback\tLive Video Playback - Start`

- Live Video Playback - End
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.liveVideoPlaybackEnd(<success>)`
  Generated message:
  `Ls\tLivePlayback\tLive Video Playback - End\t<boolean success>`

- Live Video Playback - First Picture
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.liveVideoPlaybackFirstPicture()`
  Generated message:
  `Ls\tLivePlayback\tLive Video Playback - First Picture`

- Video Resolution
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.videoResolution(<resolution_x>, <resolution_y>)`
  Generated message:
  `Ls\tLivePlayback\tVideo Resolution\t<int resolution_x>\t<int resolution_y>`

- Live Video Playback - Stall Start
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.liveVideoPlaybackStallStart()`
  Generated message:
  `Ls\tLivePlayback\tLive Video Playback - Stall Start`

- Live Video Playback - Stall End
  `eu.TRIANGLE_project.appinstr.ls.LiveVideoPlayback.liveVideoPlaybackStallEnd()`
  Generated message:
  `Ls\tLivePlayback\tLive Video Playback - Stall End`

Broadcast Live Video

- Broadcast Live Video - Start
  `eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.broadcastLiveVideoStart()`
  Generated message:
  `Ls\tLiveBroadcast\tBroadcast Live Video - Start`

- Broadcast Live Video - End
  `eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.broadcastLiveVideoEnd(<success>)`
  Generated message:
  `Ls\tLiveBroadcast\tBroadcast Live Video - End\t<boolean success>`

- Broadcast Live Video - First Picture
  `eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.broadcastLiveVideoFirstPictur`

`e()`
Generated message:
`Ls\tLiveBroadcast\tBroadcast Live Video - First Picture`

- Video Resolution
`eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.videoResolution(<resolution_x>, <resolution_y>)`
Generated message:
`Ls\tLiveBroadcast\tVideo Resolution\t<int resolution_x>\t<int resolution_y>`

- Broadcast - Stall Start
`eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.broadcastStallStart()`
Generated message:
`Ls\tLiveBroadcast\tBroadcast - Stall Start`

- Broadcast - Stall End
`eu.TRIANGLE_project.appinstr.ls.BroadcastLiveVideo.broadcastStallEnd()`
Generated message:
`Ls\tLiveBroadcast\tBroadcast - Stall End`

15.4 Social Networking

Post Image

- Post Image - Start
`eu.TRIANGLE_project.appinstr.sn.PostImage.postImageStart()`
Generated message:
`Sn\tPostImage\tPost Image - Start`

- Post Image - End
`eu.TRIANGLE_project.appinstr.sn.PostImage.postImageEnd(<success>)`
Generated message:
`Sn\tPostImage\tPost Image - End\t<boolean success>`

Post Video

- Post Video - Start
`eu.TRIANGLE_project.appinstr.sn.PostVideo.postVideoStart()`
Generated message:
`Sn\tPostVideo\tPost Video - Start`

- Post Video - End
`eu.TRIANGLE_project.appinstr.sn.PostVideo.postVideoEnd(<success>)`
Generated message:
`Sn\tPostVideo\tPost Video - End\t<boolean success>`

Post Text

- Post Text - Start
`eu.TRIANGLE_project.appinstr.sn.PostText.postTextStart()`
Generated message:

`Sn\tPostText\tPost Text – Start`

- Post Text - End
  `eu.TRIANGLE_project.appinstr.sn.PostText.postTextEnd(<success>)`
  Generated message:
  `Sn\tPostText\tPost Text – End\t<boolean success>`

Post File

- Post File - Start
  `eu.TRIANGLE_project.appinstr.sn.PostFile.postFileStart()`
  Generated message:
  `Sn\tPostFile\tPost File – Start`

- Post File - End
  `eu.TRIANGLE_project.appinstr.sn.PostFile.postFileEnd(<success>)`
  Generated message:
  `Sn\tPostFile\tPost File – End\t<boolean success>`

Show Image

- Social Networking - Image Download Start
  `eu.TRIANGLE_project.appinstr.sn.ShowImage.socialNetworkingImageDownloadStart()`
  Generated message:
  `Sn\tShowImage\tSocial Networking – Image Download Start`

- Social Networking - Image Download End
  `eu.TRIANGLE_project.appinstr.sn.ShowImage.socialNetworkingImageDownloadEnd(<success>)`
  Generated message:
  `Sn\tShowImage\tSocial Networking – Image Download End\t<boolean success>`

Play Video

- Social Networking - Play Video Start
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingPlayVideoStart()`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking – Play Video Start`

- Social Networking - Play Video End
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingPlayVideoEnd(<success>)`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking – Play Video End\t<boolean success>`

- Social Networking - Video First Picture
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingVideoFirstPicture()`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking – Video First Picture`

- Social Networking - Video Resolution
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingVideoResolution(<resolution_x>, <resolution_y>)`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking - Video Resolution\t<int resolution_x>\t<int resolution_y>`

- Social Networking - Video Stall Start
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingVideoStallStart()`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking - Video Stall Start`

- Social Networking - Video Stall End
  `eu.TRIANGLE_project.appinstr.sn.PlayVideo.socialNetworkingVideoStallEnd()`
  Generated message:
  `Sn\tPlayVideo\tSocial Networking - Video Stall End`

File Downloading

- Social Networking - File Download Start
  `eu.TRIANGLE_project.appinstr.sn.FileDownloading.socialNetworkingFileDownloadStart()`
  Generated message:
  `Sn\tFileDownload\tSocial Networking - File Download Start`

- Social Networking - File Download End
  `eu.TRIANGLE_project.appinstr.sn.FileDownloading.socialNetworkingFileDownloadEnd(<success>)`
  Generated message:
  `Sn\tFileDownload\tSocial Networking - File Download End\t<boolean success>`

Play Live Video from User

- Social Networking - Live Streaming Start
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStreamingStart()`
  Generated message:
  `Sn\tPlayFromUser\tSocial Networking - Live Streaming Start`

- Social Networking - Live Streaming End
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStreamingEnd(<success>)`
  Generated message:
  `Sn\tPlayFromUser\tSocial Networking - Live Streaming End\t<boolean success>`

- Social Networking - Live Streaming First Frame
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStreamingFirstFrame()`
  Generated message:
  `Sn\tPlayFromUser\tSocial Networking - Live Streaming First Frame`

- Social Networking - Live Streaming Resolution
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStream`

`ingResolution(<resolution_x>, <resolution_y>)`
Generated message:
`Sn\tPlayFromUser\tSocial Networking - Live Streaming Resolution\t<int resolution_x>\t<int resolution_y>`

- Social Networking - Live Streaming Stall Start
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStreamingStallStart()`
  Generated message:
  `Sn\tPlayFromUser\tSocial Networking - Live Streaming Stall Start`

- Social Networking - Live Streaming Stall End
  `eu.TRIANGLE_project.appinstr.sn.PlayLiveVideoFromUser.socialNetworkingLiveStreamingStallEnd()`
  Generated message:
  `Sn\tPlayFromUser\tSocial Networking - Live Streaming Stall End`

Search Object

- Social Networking - Search Start
  `eu.TRIANGLE_project.appinstr.sn.SearchObject.socialNetworkingSearchStart(<success>)`
  Generated message:
  `Sn\tSearch\tSocial Networking - Search Start\t<boolean success>`

- Social Networking - Search First Result
  `eu.TRIANGLE_project.appinstr.sn.SearchObject.socialNetworkingSearchFirstResult()`
  Generated message:
  `Sn\tSearch\tSocial Networking - Search First Result`

15.5 High Speed Internet

File Download

- File Download - Start
  `eu.TRIANGLE_project.appinstr.hs.FileDownload.fileDownloadStart(<transfer_id>)`
  Generated message:
  `Hs\tDownload\tFile Download - Start\t<int transfer_id>`

- File Download - End
  `eu.TRIANGLE_project.appinstr.hs.FileDownload.fileDownloadEnd(<transfer_id>, <success>)`
  Generated message:
  `Hs\tDownload\tFile Download - End\t<int transfer_id>\t<boolean success>`

File Upload

- File Upload - Start
  `eu.TRIANGLE_project.appinstr.hs.FileUpload.fileUploadStart(<success>)`
  Generated message:

```
Hs\tUpload\tFile Upload - Start\t<boolean success>
```

- File Upload - End
  `eu.TRIANGLE_project.appinstr.hs.FileUpload.fileUploadEnd(<transfer_id>, <success>)`
  Generated message:
  `Hs\tUpload\tFile Upload - End\t<int transfer_id>\t<boolean success>`

Pause and Resume Download

- File Download - Pause
  `eu.TRIANGLE_project.appinstr.hs.PauseandResumeDownload.fileDownloadPause(<success>)`
  Generated message:
  `Hs\tDownloadPause\tFile Download - Pause\t<boolean success>`

- File Download - Resume
  `eu.TRIANGLE_project.appinstr.hs.PauseandResumeDownload.fileDownloadResume(<success>)`
  Generated message:
  `Hs\tDownloadPause\tFile Download - Resume\t<boolean success>`

Pause and Resume Upload

- File Upload - Pause
  `eu.TRIANGLE_project.appinstr.hs.PauseandResumeUpload.fileUploadPause(<success>)`
  Generated message:
  `Hs\tUploadPause\tFile Upload - Pause\t<boolean success>`

- File Upload - Resume
  `eu.TRIANGLE_project.appinstr.hs.PauseandResumeUpload.fileUploadResume(<success>)`
  Generated message:
  `Hs\tUploadPause\tFile Upload - Resume\t<boolean success>`

15.6 Virtual Reality

Virtual Reality Session

- Scenario Selected
  `eu.TRIANGLE_project.appinstr.vr.VirtualRealitySession.scenarioSelected()`
  Generated message:
  `Vr\tVrSession\tScenario Selected`

- 3D Visual Context Loaded
  `eu.TRIANGLE_project.appinstr.vr.VirtualRealitySession._3DVisualContextLoaded()`
  Generated message:
  `Vr\tVrSession\t3D Visual Context Loaded`

- Immersion Session Started
  `eu.TRIANGLE_project.appinstr.vr.VirtualRealitySession.immersionSessionStarted()`
  Generated message:

`Vr\tVrSession\tImmersion Session Started`

- Immersion Session Ended
  `eu.TRIANGLE_project.appinstr.vr.VirtualRealitySession.immersionSessionEnded(<success>)`
  Generated message:
  `Vr\tVrSession\tImmersion Session Ended\t<boolean success>`

- Immersion Session Resolution
  `eu.TRIANGLE_project.appinstr.vr.VirtualRealitySession.immersionSessionResolution()`
  Generated message:
  `Vr\tVrSession\tImmersion Session Resolution`

15.7 Augmented Reality

Augmented Reality Session

- Aim To Physical Marker
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.aimToPhysicalMarker()`
  Generated message:
  `Ar\tArSession\tAim To Physical Marker`

- Aim at Location
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.aimAtLocation()`
  Generated message:
  `Ar\tArSession\tAim at Location`

- Virtual Layer Displayed
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.virtualLayerDisplayed()`
  Generated message:
  `Ar\tArSession\tVirtual Layer Displayed`

- Augmentation Session Started
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.augmentationSessionStarted()`
  Generated message:
  `Ar\tArSession\tAugmentation Session Started`

- Augmentation Session Ended
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.augmentationSessionEnded(<success>)`
  Generated message:
  `Ar\tArSession\tAugmentation Session Ended\t<boolean success>`

- Clear Augmentation Layer - Start
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.clearAugmentationLayerStart()`
  Generated message:
  `Ar\tArSession\tClear Augmentation Layer – Start`

- Clear Augmentation Layer - End
  `eu.TRIANGLE_project.appinstr.ar.AugmentedRealitySession.clearAugmentationLayerEn`

`d(<success>)`
Generated message:
`Ar\tArSession\tClear Augmentation Layer - End\t<boolean success>`

15.8 Gaming

Game Session

- Game Session Start
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameSessionStart()`
  Generated message:
  `Ga\tGameSession\tGame Session Start`

- Game Started
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameStarted()`
  Generated message:
  `Ga\tGameSession\tGame Started`

- Game Session End
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameSessionEnd(<success>)`
  Generated message:
  `Ga\tGameSession\tGame Session End\t<boolean success>`

- Game Content Stall Start
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameContentStallStart()`
  Generated message:
  `Ga\tGameSession\tGame Content Stall Start`

- Game Content Stall End
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameContentStallEnd()`
  Generated message:
  `Ga\tGameSession\tGame Content Stall End`

- Game Video Resolution
  `eu.TRIANGLE_project.appinstr.ga.GameSession.gameVideoResolution()`
  Generated message:
  `Ga\tGameSession\tGame Video Resolution`

Pause and Resume

- Game Pause
  `eu.TRIANGLE_project.appinstr.ga.PauseandResume.gamePause(<success>)`
  Generated message:
  `Ga\tGamePause\tGame Pause\t<boolean success>`

- Game Resume
  `eu.TRIANGLE_project.appinstr.ga.PauseandResume.gameResume(<success>)`
  Generated message:
  `Ga\tGamePause\tGame Resume\t<boolean success>`

Saved Game Session

- Saved Game Load Start
  `eu.TRIANGLE_project.appinstr.ga.StartSavedGameSession.savedGameLoadStart()`
  Generated message:
  `Ga\tSavedGame\tSaved Game Load Start`

16 Annex 4: OpenStack API access

In the URLs, the service names need to be replaced with IP addresses from section 10.1.7.

Identity API: http://<Keystone>:5000/v3
The Identity service provided by the Keystone package provides authentication tokens that are used by the other REST APIs to authenticate and authorize the client. A client first needs to authenticate itself and the Identity Service and request an authentication token, this authentication token can then be used for a period of time to make follow-up requests to the other REST APIs. The exact process is described in[44], though summarizes to the following actions

1. Make an HTTP POST request to the Identity API service containing the headers and data from Table 64, with variables filled in from Table 50:
2.

Table 64 Identity Service API Request Token

```
Content-Type: application/json
{
  "auth": {
    "identity": {
      "methods": [
        "password"
      ],
      "password": {
        "user": {
          "domain": {
            "name": "$OS_USER_DOMAIN_NAME"
          },
          "name": "$OS_USERNAME",
          "password": "$OS_PASSWORD"
        }
      }
    },
    "scope": {
      "project": {
        "domain": {
          "name": "$OS_PROJECT_DOMAIN_NAME"
        },
        "name": "$OS_PROJECT_NAME"
      }
    }
  }
}
```

3. In response, if authentication succeeds, the Identity API will respond similar to Table 65, where HTTP 201 created indicates a successful authentication, the HTTP Header *X-Subject-Token* indicated the token, and response DATA provides further information about the session and user account.

Table 65 Identity API Service Authentication Token Response

```
HTTP 201 Created
X-Subject-Token: d55fa31bec30448386e34e082aa6e0fe
{
  "token": {
    "is_domain": false,
    "methods": [
      "password"
    ],
    "roles": [
      {
        "id": "9fe2ff9ee4384b1894a90878d3e92bab",
        "name": "_member_"
      },
      {
        "id": "86599d047b944cf4ac64dc5b676518a0",
        "name": "Admin"
      }
    ],
    "expires_at": "2017-12-13T15:24:39.000000Z",
    "project": {
      "domain": {
        "id": "default",
        "name": "Default"
      },
      "id": "086a8aacd0bc4aaebdb87ba3f8e8a556",
      "name": "admin"
    },
    "user": {
      "password_expires_at": null,
      "domain": {
        "id": "default",
        "name": "Default"
      },
      "id": "056b7395f54c486a85e130f4c69bdc74",
      "name": "admin"
    },
    "audit_ids": [
      "aZRwAlMBRGa_uhjOugT__Q"
    ],
    "issued_at": "2017-12-13T14:24:39.000000Z"
  }
}
```

4. Store the token returned in HTTP Header *X-Subject-Token* into the variable *$OS_TOKEN* and include it in the HTTP Header *X-Auth-Token* in future requests to the other REST APIs. For example, using *curl* you can request an overview of all other available APIs from the Identity service: `` `curl -v -H "X-Auth-Token: $OS_TOKEN" $OS_AUTH_URL/auth/catalog | python -m json.tool` ``. Additionally, you may need the project-id of the OpenStack project you're authenticated to which is stored in the JSON-encapsulated response in the property *token.project.id* and store it into *$OS_PROJECT_ID*.

Using this token, the following APIs can be accessed (replace service names with IP addresses from Section 10.1.7 and use the appropriate $OS_PROJECT_ID from a project you are authorized on):

Name	**Service Type**	**URL**
Cinder V3	Volume	http://<Cinder-API>:8776/v3/<$OS_PROJECT_ID>
Cinder V2	Volume	http://<Cinder-API>:8776/v2/<$OS_PROJECT_ID>
Glance	Image	http://<Glance>:9292
Nova	Compute	http://<Nova-Cloud-Controller>:8774/v2/<$OS_PROJECT_ID>
Keystone	Identity	http://<Keystone>:5000/v2.0
Neutron	Network	http://<Neutron-API>:9696
Placement (Nova)	Compute Placement	http://<Nova-Cloud-Controller>:8778
Heat	Orchestration[9]	http://<Heat>:8004/v1/<$OS_PROJECT_ID>
Heat-CFN	CloudFormation-compatible Orchestration	http://<Heat>:8000/v1

The OpenStack API Documentation (OpenStack API Documentation, sd) give a full description of all APIs and their exact commands.

[9] Note: a different solution, i.e., Open Source Mano, is used as the orchestrator.

17 Annex 5: Sample work-flow

In this annex we will present a sample workflow which can be used by the experimenter using MANO integrated with TAP. We will show steps needed to create a basic TRIANGLE topology and perform a basic connectivity test. We assume the experimenter is using the infrastructure ("Stable Cloud5") described in the previous sections of this document.

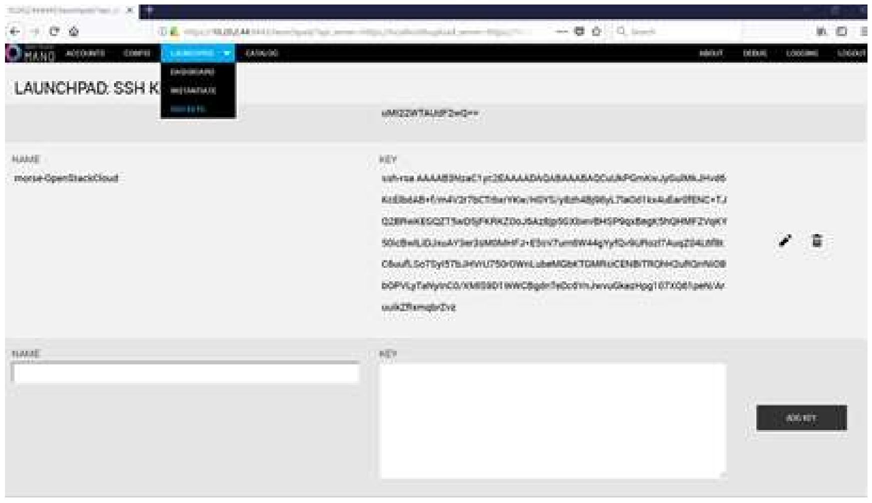
17.1 Bootstrapping environment

We will describe the one-time-effort actions required to bootstrap the environment for a particular experiment.

17.1.1 Key deployment

Most of the VM images which are to be used in OpenStack environment use the *cloud-init* mechanism to preconfigure the images, among others, ssh keys are injected to the instance for authentication. Frequently, password logins are disabled due to security reasons and the only way to access an instance is to have public-private key pairs deployed. MANO allows for automation of this process but first needs to have the public key(s) stored.

1. Open MANO GUI at https://10.20.2.44:8443/launchpad and log in (default credentials are admin/admin, though these may be changed)
2. Navigate to *Launchpad->SSH keys* menu
3. Insert a public key and give it a name which will be used in further steps to identify the specific user's identity or machine's deployment key.



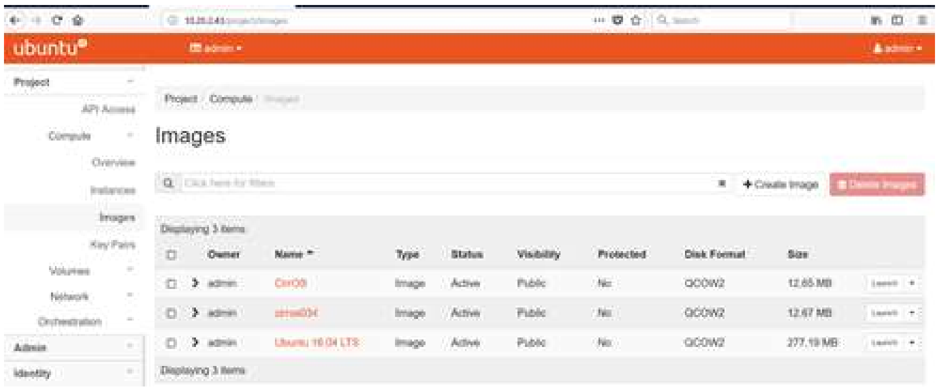
NAME: morse-OpenStackCloud
KEY: ssh-rsa [illegible]

Figure 129: Example inserting a public key

17.1.2 Image deployment

Virtual Machines instantiated by MANO are based on the images managed by OpenStack Glance service. The delivered Stable Cloud5 is pre-populated with common images, however, the experimenters may need to upload their specific images.

1. Download (or prepare) an image source file. The sources for popular images are listed here: [45].
2. Log in to the OpenStack Dashboard at http://10.20.2.43 ; the default credentials are admin/admin though may be changed.
3. Navigate to Project -> Compute -> Images and press “Create Image” button, see Figure 130.
4. Give an image a name (it will be used to identify it by MANO later on), choose the source file (downloaded or prepared), configure the right format and give RAM and disk parameters (observe, that RAM is given in MB while the disk size is given in GB) and press the “Create Image” button, see Figure 131



Displaying 3 items

	Owner	Name	Type	Status	Visibility	Protected	Disk Format	Size
	admin	CirrOS	Image	Active	Public	No	QCOW2	12.65 MB
	admin	cirros034	Image	Active	Public	No	QCOW2	12.67 MB
	admin	Ubuntu 16.04 LTS	Image	Active	Public	No	QCOW2	277.19 MB

Displaying 3 items

Figure 130: Image creation (part 1)

17.2 Preparing descriptors

MANO uses Descriptor packages for Virtual Network Functions (VNFDs) and Network Services (NSDs). In their most basic form, these are *yaml* configuration files archived and compressed through *tar* and *gzip*, possibly accompanied by some other auxiliary files (e.g., icons). A collection of sample descriptors can be found here: [46]. We will now demonstrate how to prepare a basic TRIANGLE topology which constitutes of three client machines and one server, connected to the same network.

17.2.1 Preparing Virtual Network Function Descriptor

1. Open MANO GUI at https://10.20.2.44:8443/ and log in (default credentials are admin/admin)
2. Navigate to the *Catalog* menu and press “+ Add VNFD” button. A new VNF and VDU (Virtual Deployment Unit) appear, see Figure 132
3. Click on the VNF in the main part of the screen and change its name and (after pressing “more” in the right part of the screen) its ID, see Figure 133. After pressing “Update” a new VNFD (with its newly chosen name) will appear on the right hand side due to selecting a new ID. Make sure you continue with configuring this specific VNFD. An old VNFD (vnfd-1) can be deleted.
4. Scroll down to “Connection Point” (“more” must be pressed earlier) and rename it to “eth0” (Figure 134)
5. Click on VDU in the middle of a screen. Adjust its Name, VM flavor (make sure the values here meet the minimal requirements for a given image, see Section 17.1.2), Image name (make sure it exists in OpenStack), External Interface (changed in previous bullet) and ID, see Figure 135. Press Update to save the changes.
6. Return to the VNF config and adjust connectivity, as VDU id has just been changed (Figure 136)

7. For the reference, the *yaml* configurationfile containing the resulting descriptors is provided in Table 66

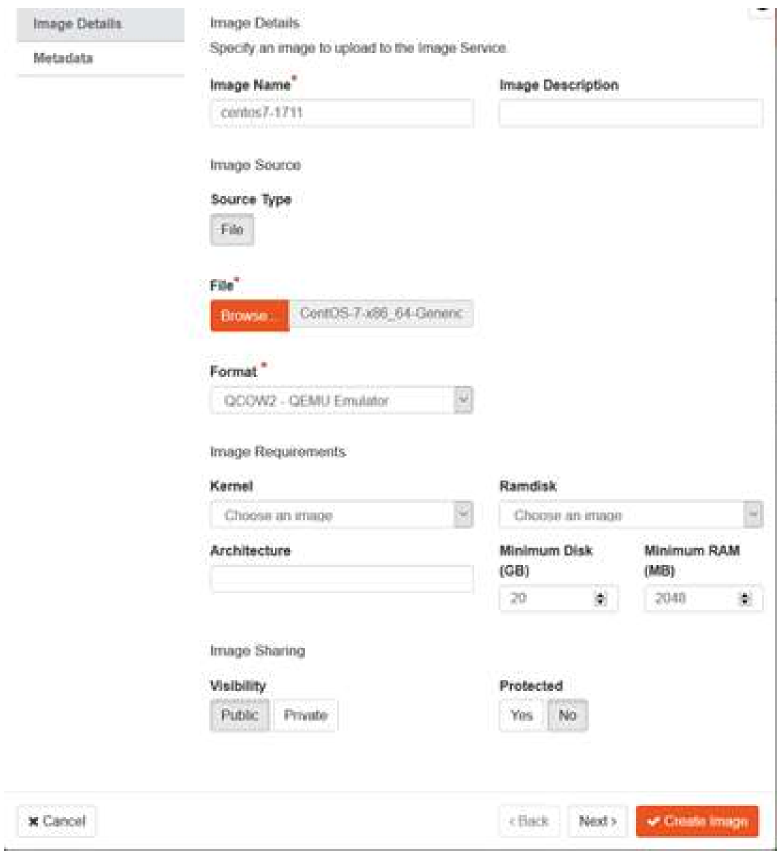


Image Details
Metadata

Image Details
Specify an image to upload to the Image Service.

Image Name* centos7-1711
Image Description

Image Source
Source Type: File
File*: Browse... CentOS-7-x86_64-Generic
Format*: QCOW2 - QEMU Emulator

Image Requirements
Kernel: Choose an image
Ramdisk: Choose an image
Architecture:
Minimum Disk (GB): 20
Minimum RAM (MB): 2048

Image Sharing
Visibility: Public Private
Protected: Yes No

✖ Cancel ‹ Back Next › ✔ Create Image

Figure 131: Image creation (part 2)

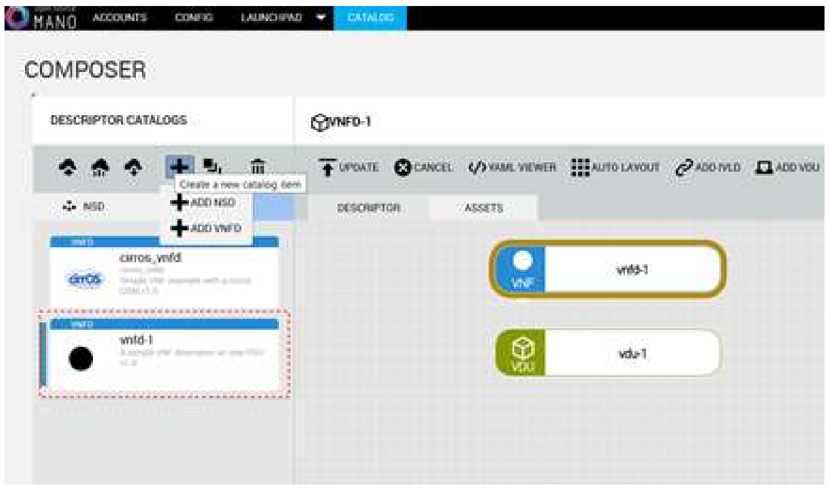


Figure 132: VNFD creation (adding VNDF 1)

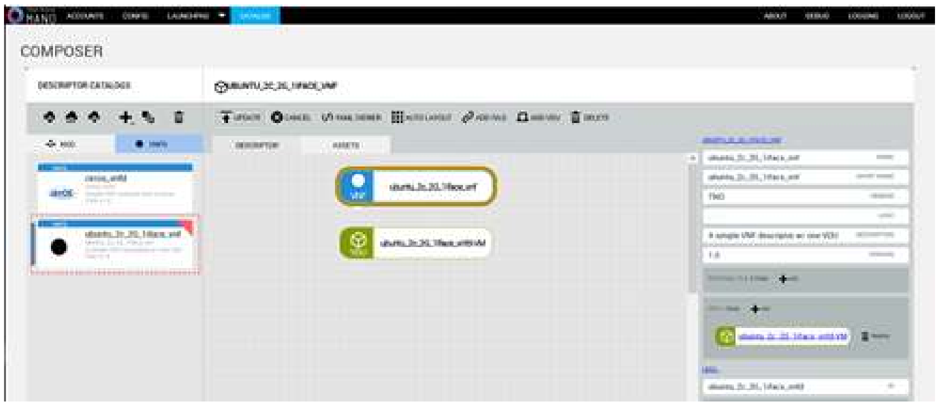


Figure 133: VNFD creation (name and ID)

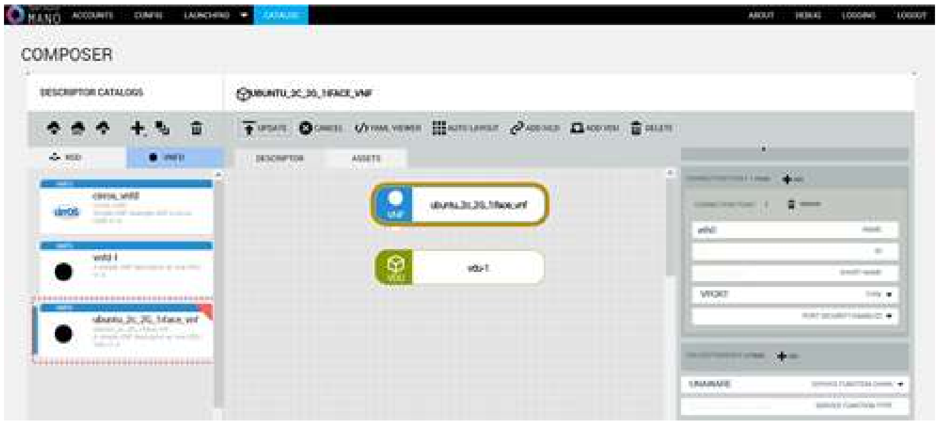
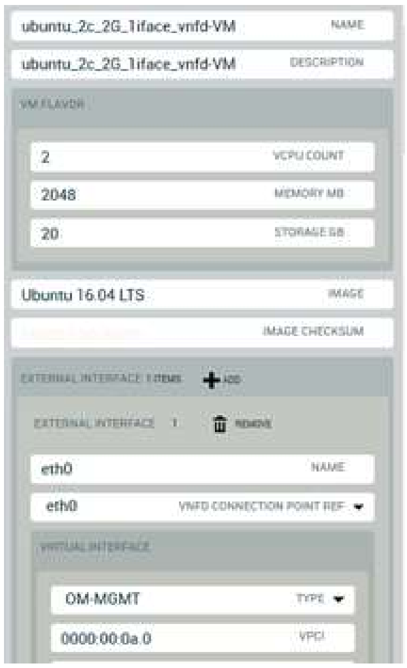
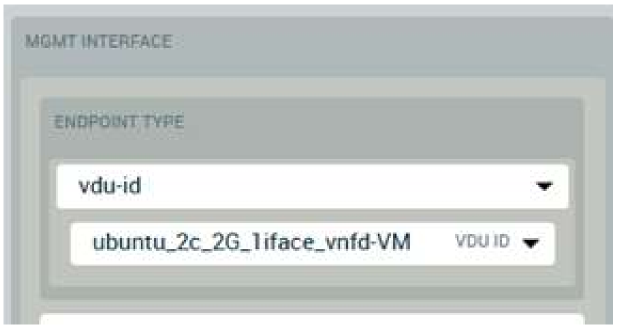


Figure 134: VNFD creation (connectivity)



ubuntu_2c_2G_1iface_vnfd-VM	NAME
ubuntu_2c_2G_1iface_vnfd-VM	DESCRIPTION
VM FLAVOR	
2	VCPU COUNT
2048	MEMORY MB
20	STORAGE GB
Ubuntu 16.04 LTS	IMAGE
	IMAGE CHECKSUM
EXTERNAL INTERFACE 1 ITEMS	ADD
EXTERNAL INTERFACE 1	REMOVE
eth0	NAME
eth0	VNFD CONNECTION POINT REF
VIRTUAL INTERFACE	
OM-MGMT	TYPE
0000:00:0a.0	VPCI

Figure 135: VNFD creation (VDU details)



MGMT INTERFACE	
ENDPOINT TYPE	
vdu-id	
ubuntu_2c_2G_1iface_vnfd-VM	VDU ID

Figure 136: VNFD creation (VNF connectivity)

Table 66 Sample client VNFD

```
ubuntu_2c_2G_1iface_vnf


---
  id: "ubuntu_2c_2G_1iface_vnfd"
  name: "ubuntu_2c_2G_1iface_vnf"
  short-name: "ubuntu_2c_2G_1iface_vnf"
  vendor: "TNO"
  description: "A simple VNF descriptor w/ one VDU"
  version: "1.0"
  mgmt-interface:
     vdu-id: "ubuntu_2c_2G_1iface_vnfd-VM"
  connection-point:
     -
       name: "eth0"
       type: "VPORT"
  vdu:
     -
       id: "ubuntu_2c_2G_1iface_vnfd-VM"
       name: "ubuntu_2c_2G_1iface_vnfd-VM"
       description: "ubuntu_2c_2G_1iface_vnfd-VM"
       vm-flavor:
          vcpu-count: 2
          memory-mb: 2048
          storage-gb: 20
       guest-epa:
          cpu-pinning-policy: "ANY"
       image: "Ubuntu 16.04 LTS"
       supplemental-boot-data:
          boot-data-drive: "false"
       external-interface:
          -
            name: "eth0"
            vnfd-connection-point-ref: "eth0"
            virtual-interface:
               type: "OM-MGMT"
               vpci: "0000:00:0a.0"
               bandwidth: 0
  service-function-chain: "UNAWARE"
  meta:               "{\"containerPositionMap\":{\"a66a5a22-5fc0-4c82-87fe-
a4a41a90751d\":{\"top\":30,\"left\":260,\"right\":510,\"bottom\":85,\"width\
":250,\"height\":55},\"a66a5a22-5fc0-4c82-87fe-a4a41a90751d/vdu-
1\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"h
eight\":55},\"ubuntu_2c_2G_1iface_vnf\":{\"top\":30,\"left\":260,\"right\":5
10,\"bottom\":85,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnf/vdu-
```

1\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnf/ubuntu_2c_2G_1iface_vnf\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnf/ubuntu_2c_2G_1iface_vnfd-
\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnf/ubuntu_2c_2G_1iface_vnfd-
VM\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnfd\":{\"top\":30,\"left\":260,\"right\":510,\"bottom\":85,\"width\":250,\"height\":55},\"ubuntu_2c_2G_1iface_vnfd/ubuntu_2c_2G_1iface_vnfd-
VM\":{\"top\":130,\"left\":260,\"right\":510,\"bottom\":185,\"width\":250,\"height\":55}}}"

18 Annex 6: DEKRA Performance Tool Capabilities

This annex summarizes the list of measurement capabilities provided by the DEKRA wireless Performance Tool which have been integrated into release 1 of the TRIANGLE testbed.

18.1 QoS Measurements

Table 67 Performance Tool QoS Measurement Capabilities

QoS Measurements	
KPI Resolution	One KPI record per Layer 7 SDU
Throughput	Over Time, Average, CDF (Cumulative Distributed Function), PDF (Probability Distributed Function)
One-way Delay	Over Time, Average, CDF, PDF, Percentile (t)
One-way Delay Variation	Over Time, Average, CDF, PDF, Percentile (t)
One-way Packet Loss Rate	Over Time, Average, Consecutive Loss Packets
One-way Packet Loss Distribution	RTT, Number of retransmissions, Duplicated ACK, Windows Size
Others	Inter-Departure-Time, Inter-Arrival-Time

18.2 API-Driven QoE Measurements

Table 68 Performance Tool QoE Measurement Capabilities

App	**QoE Measurement**
VoIP	Estimated MOS
YouTube™	Playback Quality (t), Playback Time (t), Quality Usage CDF, Playback Size/Duration, Bufferings, MOS*
Spotify™	Track Size, Playback Duration, Bufferings, MOS*
Facebook™	Time to post text/pic/video
Web Browsing	Time to load web page
File Transfer	Time to transfer the file
Ping	RTT

* estimaded MOS

18.3 RF and System Measurements at UE

Table 69 Performance Tool RF and System Measurements

Type	**Measurements**

WLAN	RSSI, Noise Level, PHY Rate, Channel, SSID, Link Quality, Roaming
3G/GPRS	MNC, MMC, LAC, RSSI, BER, RAT, Cell-Id
CDMA/EVDO	System ID, Network ID, BSID, Cell-Id, RSSI, ECIO
LTE	Cell-Id, TAC, RSSI, RSRQ, RSRP, SNR
System	Data Connection In Use (WLAN/Cellular), Data Interface Usage (Mbit/s), Battery (%), CPU Usage (%), GPS coordinates

18.4 Built-in Traffic Generator

Table 70 Performance Tool Built-In Traffic Generator

Type	**Properties**
Protocols	TCP, UDP, IPv4, IPv4
SDU Size	Constant, Uniform, Exponential, Pareto, Cauchy, Normal, Poisson, Gamma, Weibull
SDU Inter-Departure Time	
Pre-defined Profiles	VoIP (MOS), Max. Performance, Online Gaming, RTP video, Live TV, Live Radio, PTT
Supported Patterns	Ramps, Bursts, Loops

19 Annex 7: WLAN Automation and LWIP validation results

This annex contains more relevant results from the validation of the WLAN AP automation feature which has developed in the Release 4 of the TRIANGLE testbed.

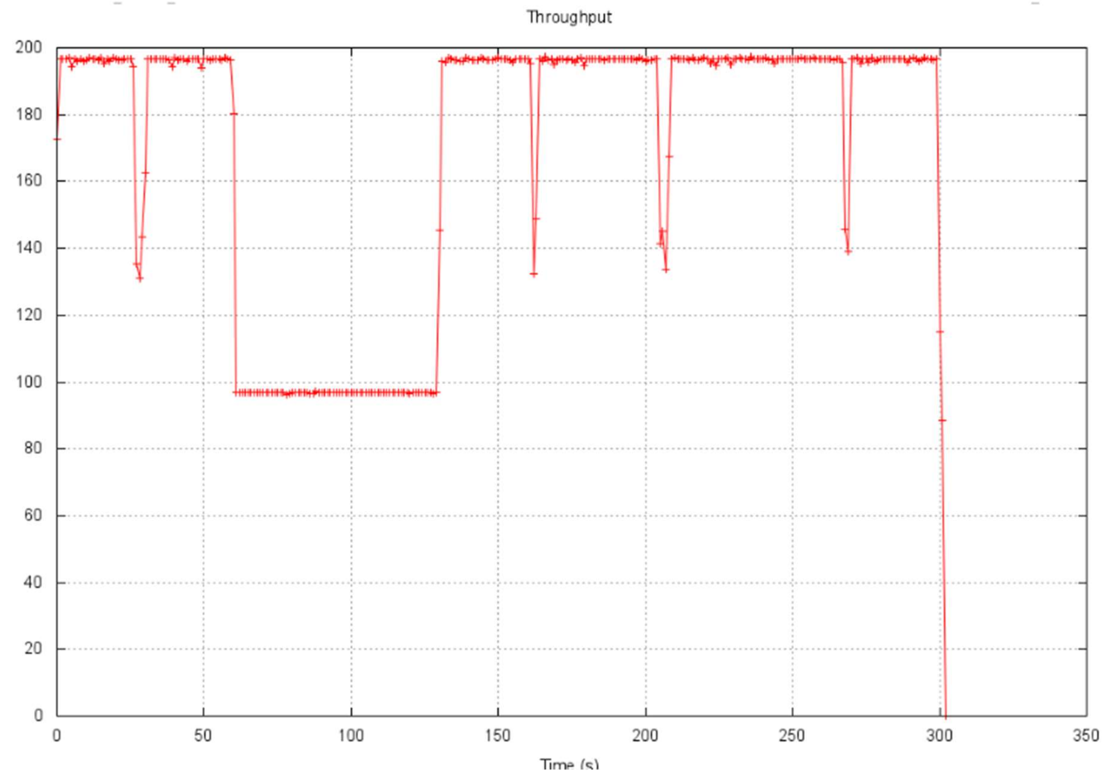
Test 1: WLAN On/Off

LTE: 50 %, WLAN: 50%

t = {0. 60} : LTE On, WLAN AP On

t = {60. 120}: LTE On, WLAN AP Off

t = {120. 300}: LTE On, WLAN AP On

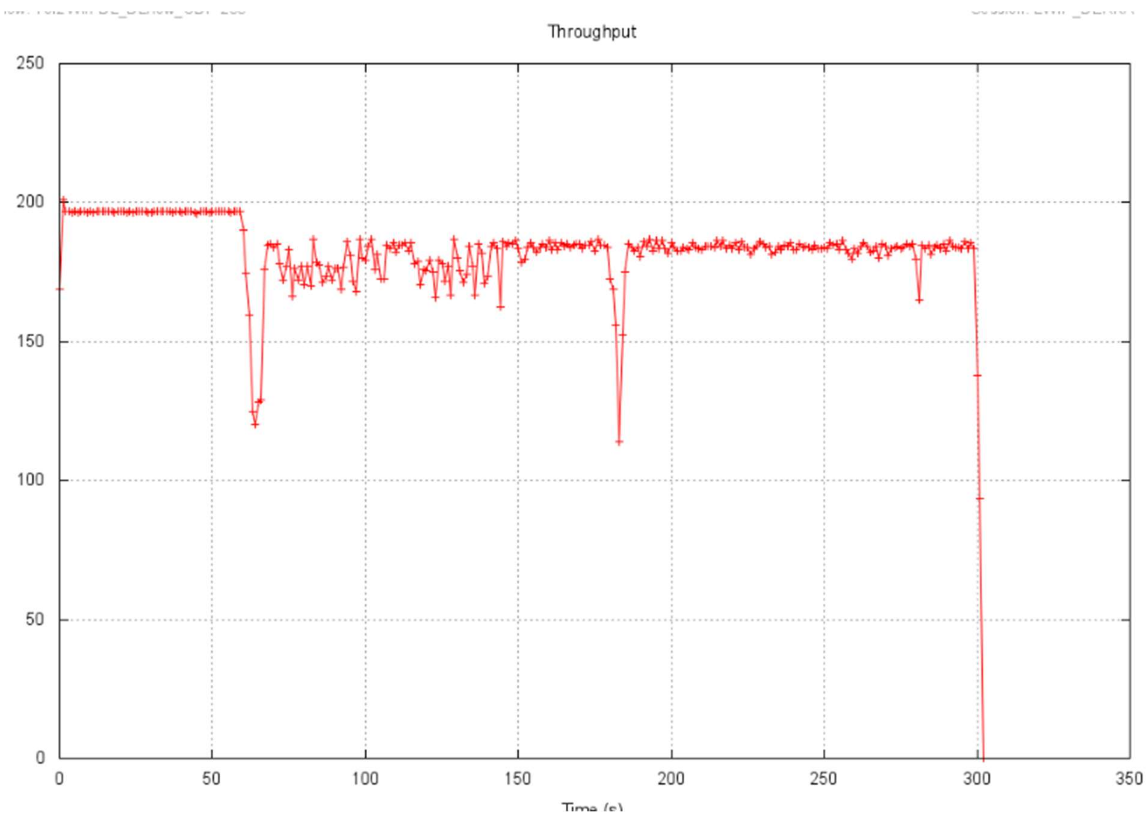


Test 2: WLAN Transmission Power

LTE: 50 %, WLAN: 50%

t = {0. 60}: LTE On, WLAN AP On at 100% tx level

t = {60. 300}: LTE On, WLAN AP On at 10% tx level



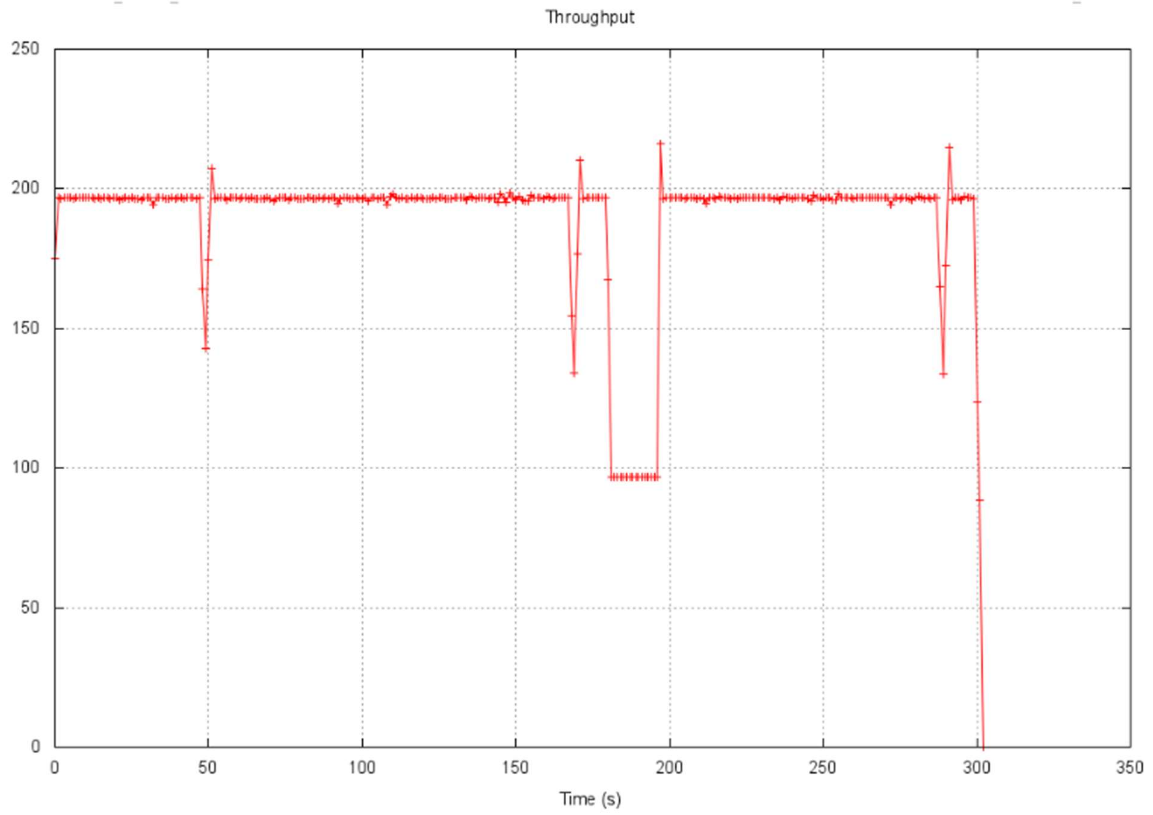
Test 3: WLAN Channel Change

LTE: 50 %, WLAN: 50%

t = {0. 180}: LTE On, WLAN AP On at channel 1

t = {180. 300}: LTE On, WLAN AP On at channel 11

In this test, the WLAN channel change is not seamless and the Wi-Fi client needs some time (around 10 seconds) to get the WLAN link up and transmitting on the new channel



20 Annex 8: Robotic Arm Implementation Details

This annex contains the details about the remote-control interface (integration with TAP) exposed by the robotic arm platform.

SET WRITE MODE

Sets the capture mode.

- ON: It saves all the captured screenshots to disk. Use this mode to obtain the target files for testing.
- OFF: It does not save the captured screenshots to disk. Use this mode for testing.

Syntax

```
APP:SETWRITEMODE mode
```

Parameters

Name	Type	Possible values
Mode	string	{on, off}

RESET

Check that the robotic arm is operational and then it sets yaw, pitch and roll to 0.

Syntax

```
ARM:RESET
```

Parameters

None.

ROLL

Roll the arm to a given position at a given speed.

Syntax

```
ARM:ROLL value, speed
```

Parameters

Name	Type	Possible values
`value`	integer	-90, 90
`speed`	enum	VERY_LOW, LOW, MEDIUM, HIGH

PITCH

Pitch the arm to a given position at a given speed.

Syntax

```
ARM:PICTH value, speed
```

Parameters

Name	Type	Possible values
`value`	integer	-90, 90
`speed`	enum	VERY_LOW, LOW, MEDIUM, HIGH

YAW

Yaw the arm to a given position at a given speed.

Syntax

```
ARM:YAW value, speed
```

Parameters

Name	Type	Possible values

`value`	integer	-180, 180
`speed`	enum	VERY_LOW, LOW, MEDIUM, HIGH

START SCREEN CAPTURE

Starts screen mirroring engine.

Invoke this function before any other function from Device Interface group.

Syntax

```
OBJECT:START orientation
```

Parameters

Name	Type	Possible values
`orientation`	integer	{VERTICAL, HORIZONTAL}

STOP SCREEN CAPTURE

Stops screen mirroring engine.

Syntax

```
OBJECT:STOP
```

Parameters

None

TAP AT LENS CENTER

Taps at the lens (VR view) center.

This function does not provoke any movement of the arm.

Syntax

```
OBJECT:TAPATLENSCENTER
```

Parameters

None.

FIND AND TAP AT OBJECT

Finds one object and tap at its center.

This function does not provoke any movement of the arm.

Syntax

```
OBJECT:FINDANDTAP delay, number, objects
```

Parameters

Name	Type	Possible values
`delay`	double	{0, 60}
`number`	integer	{1, 10}
`objects`	string	Target image file name. Size of “number”.

FIND AND SWIPE AT OBJECT

Finds one object from the device screen and swipe at its center.

This function does not provoke any movement of the arm.

Syntax

```
OBJECT:FINDANDSWIPE delay, number, objects
```

Parameters

Name	Type	Possible values
`Delay`	double	{0, 60}
`Number`	integer	{1, 10}
`objects`	string	Target image file name. Size of “number”.

MOVE FIND AND TAP AT OBJECT

Moves the robotic arm until the device screen shows one object, and then it taps at the object center.

Syntax

```
OBJECT:MOVEFINDANDTAP delay, number, objects
```

Parameters

Name	Type	Possible values
`delay`	double	{0, 60}
`number`	integer	{1, 10}
`objects`	string	Target image file name. Size of “number”.

MOVE FIND AND AIM AT OBJECT

Moves the robotic arm until the device screen shows one object, and then it moves the robotic arm until the object gets at the center of the lens aim.

Syntax

```
OBJECT:MOVEFINDANDAIM delay, number, objects
```

Parameters

Name	Type	Possible values
`delay`	double	{0, 60}
`number`	integer	{1, 10}
`objects`	string	Target image file name. Size of “number”.

FIND AND MEASURE

Measures the time until finds one object in the device screen. This function implements the performance indicator “time to load an object”.

This function does not provoke any movement of the arm.

Syntax

```
OBJECT:FINDANDMEASURE number, objects
```

Parameters

Name	Type	Possible values
number	integer	{1, 10}
objects	string	Target image file name. Size of “number”.

START SCREEN CAPTURE

Starts screen mirroring engine.

Invoke this function before any other function from Device Interface group.

Syntax

```
OBJECT:START orientation, [real width], [real height], virtual
height, matching score
```

Parameters

Name	Type	Possible values
orientation	integer	{VERTICAL, HORIZONTAL}
real width	Integer	{1:1440}
real height	Integer	{1:2960}
virtual height	Integer	{1:2960}
matching score	Integer	{0:100}

FIND AND PRESS AT OBJECT

Finds one object and press at its center.

This function does not provoke any movement of the arm.

Syntax

```
OBJECT:FINDANDPRESS delay, number, objects, duration, timeout
```

Parameters

Name	Type	Possible values
`delay`	double	{0, 60}
`number`	integer	{1, 10}
`objects`	string	Target image file name. Size of “number”.
`duration`	integer	Duration of the press action in milliseconds: {50:10000}
`timeout`	integer	{1:300} seconds

FIND TAP AND MEASURE

Finds and taps on a target image and it measures the time to find a second image. This function implements the performance indicator “lagging”.

Syntax

```
OBJECT:FINDTAPANDMEASURE first, second, timeout
```

Parameters

Name	Type	Possible values
`first`	string	First target image file name on which the system will send the “tap” event.
`second`	string	Second target image file name on which the system will calculated time to appears on the device screen.
`timeout`	integer	{1:300} seconds

FIND PRESS AND MEASURE

Finds and presses on a target image and it measures the time to find a second image. This function implements the performance indicator “lagging”.

Syntax

```
OBJECT:FINDPRESSANDMEASURE first, second, timeout
```

Parameters

Name	Type	Possible values
`first`	string	First target image file name on which the system will send the “press” event.
`second`	string	Second target image file name on which the system will calculated time to appears on the device screen.
`timeout`	integer	{1:300} seconds

21 Annex 9: TRIANGLE Report

The following report has been automatically generated by the Release 4 of the TRIANGLE web portal.

Tested Item

Type of product	App
Product name	com.google.android.exoplayer2.demo
SW version	2804
Operating system	Android
OS version	<S8_OS>
Supported use cases	Content Distribution Streaming Services
Supported features	Media File Playback

Test Lab

Lab	UMA Calle Arquitecto Francisco Peñalosa 18, 29010 Málaga
Testing period	Testing started on 2018-12-01 and finished on 2018-12-01
Date	2018-12-01

TRIANGLE mark

Compliance	IN COMPLIANCE
Score	TRIANGLE mark score 3.93

1 Identification of the test environment

1.1 Reference devices (only for Apps)

Following mobile phones have been used as hosts for the App

Ref	Description	HW version	SW version	Serial number
4	Samsung Galaxy S8	<S8>	<S8_OS>	ce11171b9acc0d3205

2 Identification of the test equipment

The testbed used to perform the testing is composed by the following equipment:

Reference	Equipment	Serial Number
1	E7515A UXM Wireless Test Set	TH53460091-01
2	N6705 DC Power Analyzer	DC451357

3 Test Results

The following table provides a summary of the test cases performed to obtain the TRIANGLE mark

Test case	Result
AUE/CS/001	Pass
AEC/CS/001	Pass
RES/CS/001	Pass

Note: The results relate only to the items tested.

22 Annex 11: TRIANGLE Mark

22.1 KPI computation

The following KPIs have been computed from the measurements performed in the test cases listed in section 3.

Domain	**KPI**	**Result**
Device Resources Usage	average Non Interactive Playback average CPU usage [HS_DP]	4.722
Device Resources Usage	average Non Interactive Playback average Memory usage [HS_DP]	2.827
Device Resources Usage	average Non Interactive Playback average CPU usage [SU_FE]	4.741
Device Resources Usage	average Non Interactive Playback average Memory usage [SU_FE]	2.82
Device Resources Usage	average Non Interactive Playback average CPU usage [SU_SB]	4.774
Device Resources Usage	average Non Interactive Playback average Memory usage [SU_SB]	2.829
Device Resources Usage	average Non Interactive Playback average CPU usage [SU_SO]	4.744
Device Resources Usage	average Non Interactive Playback average Memory usage [SU_SO]	2.833
Device Resources Usage	average Non Interactive Playback average CPU usage [SU_ST]	4.766
Device Resources Usage	average Non Interactive Playback average Memory usage [SU_ST]	2.819
Device Resources Usage	average Non Interactive Playback average CPU usage [UR_DN]	4.754
Device Resources Usage	average Non Interactive Playback average Memory usage [UR_DN]	2.804
Device Resources Usage	average Non Interactive Playback average CPU usage [UR_DT]	4.715
Device Resources Usage	average Non Interactive Playback average Memory usage [UR_DT]	2.8
Device Resources Usage	average Non Interactive Playback average CPU usage [UR_IB]	4.814

Device Resources Usage	average Non Interactive Playback average Memory usage [UR_IB]	2.821
Device Resources Usage	average Non Interactive Playback average CPU usage [UR_IO]	4.762

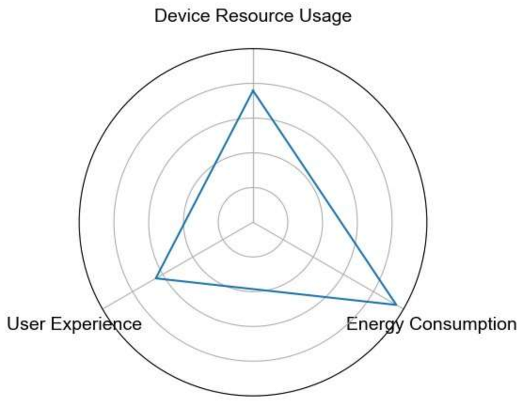
22.2 TRIANGLE mark

The TRIANGLE mark detailed below has been obtained from the KPIs indicated in previous section.

Domain	**Score**
Device Resource Usage	3.79
Energy Consumption	4.76
User Experience	3.23

TRIANGLE MARK

3.93



Spider diagram

23 Annex 10: DEKRA wireless Performance Tool Automation Interface

23.1 Interface Description

DEKRA-Controller is constituted by two separate processes: The graphical user interface (GUI) and the controller itself (`PerformanceTester.exe`).

The automation described in this annex consists on replacing the DEKRA-Controller GUI process by an external Automation Suite as depicted in the figure below.

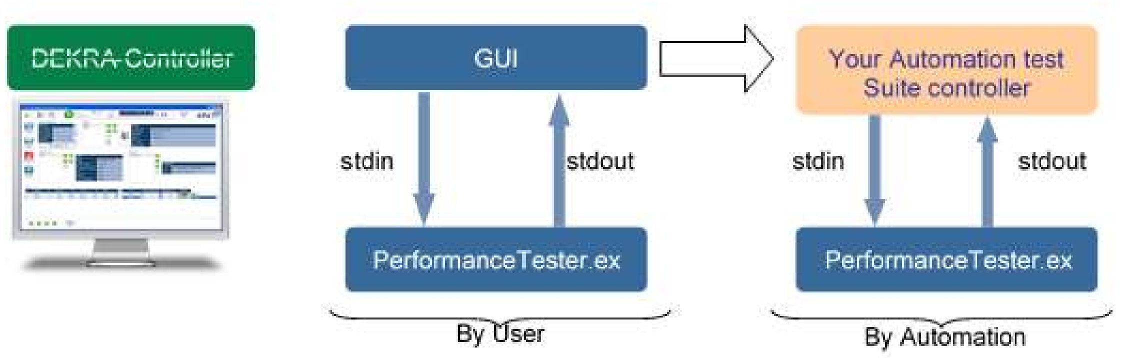


Figure 137 DEKRA wireless Performance Automation Interface (rel'1)

Basically, the Automation Suite shall launch the process PerformanceTester.exe to execute a measurement:

```
C:\Program Files (x86)\DEKRA wireless Performance Tool
Controller\PerformanceTester.exe Input.xml
```

The process PerformanceTester.exe needs one argument called Input.xml which contains the required configuration to execute the measurement.

- `Input.xml` is a file generated by the GUI process right after "Play" button is clicked by the User in the regular way (i.e., commanded by User through GUI). This file contains all the configuration information (agents, data profiles, etc.).
- `Input.xml` is described in section 3.

The Automation Suite can use *stdin* (section 4) and *stdout* (section 5) streams to exchange information with the process `PerformanceTester.exe` while the measurement is running.

`PerformanceTester.exe` generates the results files of the test in a folder that can be learned by the automation suite from *stdout* messages.

23.2 Recommended automation methodology

The recommended automation methodology is to record into an XML file, the set of steps by first executing them manually.

To do so, for a given test case (Test Case A):

1. Open the GUI
2. Create the Configuration in the GUI
3. Click Play button (even if the test setup is not ready, the XML file is generated just after clicking Play button
4. Go to %TEMP% and catch "Input.xml".
5. Rename "Input.xml" to "Test_Case_A_input.xml"

For other test cases repeat steps 2 through 5. At the end a sort of library of reference input XML files should be available.

Accordingly, the Automation Suite (e.g., script) can be something like this:

```
//Input Parameters:
  • A. Test_Case_X_input.xml for a given Test Case "X"
  • B. Actual Agents Information
  • C. Number of iterations

//Script Content
Actual_Input.xml=Modify(Test_Case_X_input.xml,B)
For i=0;i<C
    execute(PerformanceTester.exe Actual_Input.xml)
```

The goal of generating the reference XML files before executing any test case is to minimize the complexity of the "`Modify`" function in the Automation Suite.

Ideally, once this is done, the Automation Suite should just need to change DEKRA-Agents IP addresses from the XML file. The script should not need to modify more fields from the XML file.

23.3 Input XML file

This section describes the structure of `Input.xml` file.

All fields are mandatory.

The fields that may need to be modified in the Automation Suite have been highlighted in red.

```
<input>
  <CaptureTraffic>
    <main_path/> Working directory of the tool (1)
    <temp_path/> Temp folder of the system
    <session_folder/> Name of the project (1)
    <time_folder/> If set to "none", folder with data and time will be created. Otherwise the given folder will be used to
    store the results (folder must exist) (10)
    <repetition_folder>none</repetition_folder> Do not modify
    <PTversion>50007</PTversion> Do not modify
    <pt_user>DEKRA wireless</pt_user> Do not modify
    <generate_record_file>0</generate_record_file> GUI: Post Mortem Recovery (1:Yes, 0:No) Do not modify
    <record_file_interval>60</record_file_interval> GUI: Interval Between Backups. Do not modify
    <delete_record_file>1</delete_record_file> GUI: Delete Backup Files After Test. Do not modify
    <basic_mode/> GUI: Test Mode (0: Basic Mode, 1: Advanced Mode)
    <using_wpcap/> GUI: Test Mode (0: L7 Mode, 1: L3 Mode)
    <time_duration>87000</time_duration> Do not modify
    <packet_duration>50000000</packet_duration> Do not modify
    <waiting_client_agents/> GUI: Control Plane Connection Timeout
    <waiting_server_agents/> GUI: Control Plane Connection Timeout
    <packet_loss_th/> GUI: Loss Threshold
    <poisson_lambda/> GUI: Graph Resolution
    <thput_PDF_interval/> GUI: Thput PDF Width (kbit/s)
    <delay_PDF_interval/> GUI: OWD/IPDV PDF Width
    <jitter_PDF_interval/> GUI: OWD/IPDV PDF Width
    <peak_percentile/> GUI: Peak Definition
    <wifi_interval>1000</wifi_interval> Do not modify
    <cell_interval>1000</cell_interval> Do not modify
    <generate_XML>0</generate_XML> Do not modify
    <compensate_clock_skew>1</compensate_clock_skew> Do not modify
    <max_jitter_supported/> GUI: Maximum Jitter
    <target_layer/> GUI: L3 Target Layer
    <sessionrestoretime/> GUI: Reconnection Period
    <maxconnectattempts/> GUI: Max Reconnection Attempts
    <udpnatkeepaliveinterval/> GUI: UDP NAT KeepAlive Period
    <udpnatkeepalive/> GUI: UDP NAT KeepAlive During Traffic Flow
    <tcpconnecttimeout/> GUI: Data Plane Connection Timeout
    <udpconnecttimeout/> GUI: Data Plane Connection Timeout
    <resenddata/> GUI: Retransmission Period
    <pt_os/> Version of Windows
    <pt_cpus/> Number of CPUs
    <pt_parallelw2008/> Parallelization in control plane
    <remove_samples/> GUI: Remove samples from KPI
    <graph_resolution/> GUI: Graph Size
    <graph_extension/> GUI: Graph Extension
    <generate_vector_delay/> GUI: Save KPI at Packet Level
    <generate_vector_jitter/> GUI: Save KPI at Packet Level
    <generate_vector_loss/> GUI: Save KPI at Packet Level
    <generate_vector_packetsize/> GUI: Save KPI at Packet Level
    <generate_vector_idt/> GUI: Save KPI at Packet Level
    <generate_vector_txtime/> GUI: Save KPI at Packet Level
    <map_max_thr/> GUI: Map Tput Max Value
<number_of_MP/> Number of Agents participating in the test. Each of them represented in a <MP> structure
    <MP>
      <caption/> GUI: Caption
```

10 Results are generated at <main_path><session_folder><time_folder>

```
        <index/> Agent Index: starting at 0, must be incremental
        <mode>PASSIVE</mode> Control Plane Mode: Do not modify
        <identifier>Agent1</identifier> Client Mode ID: Do not modify
        <version>50007</version> Version of the Agent (5.00.07) Must match!
        <capture_data/> If the Agent must capture data in L3 (0:No, 1:Yes)
        <wifi_stats/> GUI: Capture WLAN Parameters
        <wifi_scanning/> GUI: Capture WLAN Neighbours Parameters
        <external_wifi/> GUI: Obtain WLAN Info Remotely From (-1: Unused, Other: MP index)
        <cell_stats/> GUI: Capture Cellular Parameters
        <external_cell/> GUI: Obtain Cellular Info Remotely From (-1: Unused, Other: MP index)
        <system_stats>1</system_stats> Capture System Stats: Do not modify
        <com_port>None</com_port> GUI: Cellular COM Port (Windows only) Do not modify
        <sim_pin>None</sim_pin> GUI: Cellular SIM PIN (Windows only) Do not modify
        <gps_stats/> GUI: Capture GPS Info
        <external_gps/> Obtain GPS Info Remotely From (-1: Unused, Other: MP index)
        <is_sniffer>no</is_sniffer> Do not modify
        <sniffer_mac_offset>0</sniffer_mac_offset> Do not modify
        <management_ip/> GUI: Control IP
        <management_port/> GUI: Control Port
        <data_ip/> GUI: Data IP
        <phone_number>None</phone_number> Do not modify
        <number_of_attenuator>0</number_of_attenuator> Do not modify
        <number_of_ap_automation>0</number_of_ap_automation> Do not modify
        <number_of_powersources>0</number_of_powersources> Do not modify
        <number_of_turntables>0</number_of_turntables> Do not modify
        <wifi_monitor>0</wifi_monitor> Do not modify
        <use_dsla>0</use_dsla> Do not modify
        <dev_control>0</dev_control> Do not modify
    </MP>
    <number_of_flows/> Number of Flows in the test. Each of them represented in a <flow> structure
    <flow>
        <caption/> GUI: Caption
        <index/> Flow Index: starting at 0, must be incremental
        <group/> GUI: Group
        <streams> GUI: TCP Streams. For UDP set to 1
        <generateTraffic/> 1: Data Traffic Generator, 0: Data Traffic Monitor (L3)
        <MP_DSCP/> DSCP filter for L3, -1 to ignore
        <MP_SPI>0</MP_SPI> Do not modify
        <MP_transport/> Transport protocol filter for L3 (UDP/TCP)
        <MP_src_port/> Source port filter for L3, 0 to ignore
        <MP_dst_port/> Destination port filter for L3, 0 to ignore
        <MP_pdu_size/> PDU size filter for L3, 0 to ignore
<number_of_MPs_participating/> Number of Agents participating in flow (could be >2 if using Via) for L3. Each of
them represented in a <MP_info> structure
        <MP_info>
<MP_index/> Index of the Agent participating (matching <MP><index>)
<source_ip_address/> Source IP address filter for L3 for this Agent
<destination_ip_address/> Destination IP address filter for L3 for this Agent
        </MP_info>
        <is_MOS/> Compute MOS stats for this flow (should be UDP flow)
        <jBuffer/> GUI: Jitter buffer
```

```
      <codec/> If MOS stats computed, which codec should apply
      <TG_configure_trigger>0</TG_configure_trigger> Do not modify
       <TG_configure_delay/> GUI: If “Configure before Delay”: 0, If “Configure after Delay”: Delay of the flow
      <TG_start_trigger>0</TG_start_trigger> Do not modify
      <TG_start_delay/> GUI: Delay of the flow (ms)
      <TG_end_trigger>0</TG_end_trigger> Do not modify
      <TG_end_delay/> GUI: Duration of the flow (ms)
      <TG_transport/> GUI: Transport Protocol
      <TG_max_datarate/> GUI: Max Performance
      <TG_data_port/> GUI: Transport Port
      <TG_traffic_mode/> "active": Receiver acts as Server, "passive”: Receiver acts as Client
      <TG_ip_sender_data/> Local IP address of the Sender
      <TG_ip_sender_data_public/> Public IP address of the Sender
      <TG_ip_receiver_data/> Local IP address of the Receiver
      <TG_ip_receiver_data_public>Public IP address of the Receiver
      <TG_number_steps>Number of TG Steps. Each of them represented in a <TG_step> structure
      <TG_repeat_steps> GUI: 0: Normal Mode, 1: Loop Mode
      <TG_step>
         <step_duration>86400000</step_duration> Duration of the step in ms. If it’s the last step, set to 86400000
         <idt_type>C</idt_type> GUI: Type of SDU Rate, C for Constant
         <pkts_per_s1>830</pkts_per_s1> GUI: Number of SDU/s
         <ps_type>c</ps_type> GUI: Type of Packet Size, c for Constant
         <pkts_size1>753</pkts_size1> GUI: SDU Size
      </TG_step>
    </flow>
    <number_of_voice_calls>0</number_of_voice_calls> Do not modify
  </CaptureTraffic>
</input>
```

23.4 Stdin Interface

Once `PerformanceTester.exe` is launched, it can be commanded with two different commands which must be sent to the process using the *stdin* interface.

- “`C\n`”: Stops the test. It is equivalent to clicking the Stop button (when operated by the GUI). This command is ignored if it is not sent between the occurrence of the *stodout* messages: “Capturing Traffic Starts” and “Capturing Traffic Ends”.
- “`AX\n`”: Ignore Agent with index X. Agent with index X will be ignored for the rest of the test (will be considered disconnected). Take into account that the Agent will continue performing the actions if it was configured to, and has not been stopped. It is equivalent to clicking Force Stop button (when operated by the GUI).

23.5 Stdout Interface

Once `PerformanceTester.exe` is launched, the process sends log messages through the *stdout* interface.

Each output line includes date and time (2016-03-07 08:32:23 – <log>). This string (date and time) has been omitted from the examples below for simplicity. Some other log messages have been also removed for simplicity.

The log messages are also written in the Controller_log_PT.txt file.

The following general rules apply to the logs generated by `PerformanceTester.exe`:

- <mp><x> refers to the Agent defined in the `input.xml` with index "x"
- <flow><x> refers to the Flow defined in the `input.xml` with index "x"
- <flow><x><mp><y> refers to the Flow defined in the input.xml with index "x". "y" indicates the agent participating in the flow (0 refers to the sender, 1 to the receiver)

The messages logs that can be interesting to be parsed by the Automation Suite have been highlighted in red in the next subsections.

23.5.1 Start of test messages

Below is an example of "start of test" log messages.

```
<pm><info><general><time_path>2016-04-06 08h 42m 28s</time_path></general></info></pm>
<pm><info><general><path>C:\Users\Administrator\Documents\DEKRA Performance Tool\CPU Performance\2016-04-06 08h 42m 28s</path></general></info></pm>
```

Important: This log message indicates the root folder where the results files will be stored. In this example: C:\Users\Administrator\Documents\DEKRA Performance Tool\CPU Performance\2016-04-06 08h 42m 28s\

Below is an example log messages that show when the execution phase starts.

```
<pt><version><32 bit>5.0.7<
<pt><sessionID>1457335943<
ConnectionPhase
===============
<pm><info><mp><x>Connecting...</x></mp></info></pm>
<pm><info><mp><x>Connected</x></mp></info></pm>
<pm><info><general>ConnectionPhase successfully done!</general></info></pm>

ConfigurationPhase
==================
<pm><info><mp><x>Configuring...</x></mp></info></pm>
<pm><info><general>ConfigurationPhase successfully done!</general></info></pm>

SynchronizationPhase
====================
<pm><info><mp><x>Calibrating...<
<pm><info><mp><x><sync_prog>0 to 100 value</sync_prog></0></mp></info></pm>
<pm><info><mp><x>Sync done<
<pm><info><general>SynchronizationPhase successfully done!</general></info></pm>

TestExecutionPhase
==================
<pm><info><general>Capturing Traffic Starts...</general></info></pm>
```

At this point the test has started. During the execution of the test, and before the reception of the “end time” message, the test can be stopped writing “C\n” in the stdin of PerformanceTester.exe

23.5.2 Traffic Flows Messages

Below is an example of "traffic flows" log messages.

```
<tg><info><flow><x>Traffic Flow Configured received from MP0
<tg><info><flow><x>Traffic Flow Started received from MP0
<tg><info><flow><x>Traffic Flow Ended received from MP0

<pm><info><general>Capturing Traffic Ends!</general></info></pm>
<pm><info><mp><x>Stopping Generation...<
<pm><info><mp><x>Stopping...<

SynchronizationPhase
====================
<pm><info><mp><x>Calibrating...<
<pm><info><mp><x><s2ync_prog>0 to 100 value</s2ync_prog></0></mp></info></pm>
<pm><info><mp><x>Sync done<
<pm><info><general>SynchronizationPhase2 successfully done!</general></info></pm>

GetRecordsPhase
===============
<pm><info><mp><x>GettingRecords...<
...
<pm><info><general>GetRecordsPhase successfully done!</general></info></pm>

Start of Test: 1457335945s 375005us
End of Test:   1457335957s 285005us
RFC5835 Duration 11 s 910000 us, 12 intervals of 1000 ms<

<pm><info><general><devices><results>\Devices\</results></devices></general></info></pm>

<pm><info><general><mp><x><agent_path>\AgentCaption\<
```

\AgentCaption\ stands for the relative folder where results for Agent (e.g., RSSI capture, battery, etc.), with index “x”, are being stored. Folder name matches Agent caption given in input.xml. In this example: C:\Users\Administrator\Documents\DEKRA Performance Tool\CPU Performance\2016-04-06 08h 42m 28s\Agent Caption\

```
<pm><info><mp><x>Computing System results for Agent2<
<pm><info><mp><x><system><first_battery>10<
<pm><info><mp><x><system><last_battery>10<
<pm><info><mp><x><system><battery_duration>-1<
<pm><info><mp><x><system><avg_cpu>36.69<
<pm><info><mp><x><system><max_cpu>51<
<pm><info><general><mp><x>Total computation time: 0 s<
```

Example of output for UDP

Outputs for flows depends on the Mode (L3/L7, Basic/Advanced)

```
<pm><info><flow><x><mp><0><data_rate>5.000<
<pm><info><flow><x><mp><0><max_dr>5.000<
<pm><info><flow><x><mp><0><peak_dr>5.000<
<pm><info><flow><x><mp><1><delay_av>1.707<
<pm><info><flow><x><mp><1><delay_med>0.561<
<pm><info><flow><x><mp><1><delay_min>0.409<
<pm><info><flow><x><mp><1><loss>0.000<
<pm><info><flow><x><mp><1><period>0<
<pm><info><flow><x><mp><1><jitter>0.475<
<pm><info><flow><x><mp><1><data_rate>5.002<
<pm><info><flow><x><mp><1><max_dr>5.018<
<pm><info><flow><x><mp><1><peak_dr>5.018<
<pm><info><flow><x><results>\Flow 0 (Agent1~Agent2@UDP-5M)<
```

The path in the last line is the relative folder where results for Flow "x" will be stored. Folder name matches Flow caption given in input.xml. In this example: C:\Users\Administrator\Documents\DEKRA Performance Tool\CPU Performance\2016-04-06 08h 42m 28s\ Flow 0 (Agent1~Agent2@UDP-5M\

-Example of output for TCP

Outputs for flows depends on the Mode (L3/L7, Basic/Advanced)

```
<pm><info><flow><x><mp><1><throughput>5.000<
<pm><info><flow><x><mp><1><max_throughput>4.999<
<pm><info><flow><x><mp><1><peak_throughput>4.999<
<pm><info><flow><x><mp><1><delay_av>0.021<
<pm><info><flow><x><mp><1><jitter>0.009<
<pm><info><flow><x><mp><1><data_txmed>6.275<
```

The average KPI results from flow <x> are highlighted above.

```
<pm><info><flow><x><results>\Flow 0 (Agent1~Agent2@TCP-5M)<
```

The path in the last line is the relative folder where results for Flow "x" will be stored. Folder name matches Flow caption given in input.xml. In this example: C:\Users\Administrator\Documents\DEKRA Performance Tool\CPU Performance\2016-04-06 08h 42m 28s\ Flow 0 (Agent1~Agent2@TCP-5M\

```
<pm><info><flow><x>Generating TXT Data ...<
<pm><info><flow><x>TXT Data generation time: 0 s<
<pm><info><flow><x>Generating Graphs ...<
<pm><info><flow><x>Graphs generation time: 0 s<

<pm><info><flow><x><mp><0><data_txrx>6.245<
<pm><info><flow><x><mp><1><data_txrx>6.245<
<pm><info><flow><x><mp><0><time_txrx>9.993<
```

```
<pm><info><flow><x><mp><1><time_txrx>9.988<
<pm><info><flow><x>Total computation time: 1 s<

<pm><info><group><1><average>5.002<
<pm><info><group><1><peak>5.018<
<pm><info><general><groups>Generating Groups Graphs...</groups></general></info></pm>
<pm><info><group><1><path>\Groups\Group 1\<

<pm><info><group><1><all_flows>;0;
<pm><info><general>Graphs plotting OK</general></info></pm>
```

This indicates that the flows have ended correctly (results can be read from their respective folders). If the process ends and this line does not appear, an error has occurred.

Device Automation Tasks Messages

Example of output for Web Browsing

```
<pm><info><mp><x><devcontrol><web><avg_setup>0.136<
<pm><info><mp><x><devcontrol><web><avg_session>1.484<
<pm><info><mp><x><devcontrol><web><avg_data_rate>1.390<
<pm><info><mp><x><devcontrol><web><success>100.00<
<pm><info><mp><x><devcontrol><web><failed>0.00<
<pm><info><mp><x><devcontrol><web><dropped>0.00<
<pm><info><mp><x><devcontrol><web><total>1<
```

Example of output for Youtube

```
<pm><info><mp><x><devcontrol><youtube><init_buffer>1.197<
<pm><info><mp><x><devcontrol><youtube><n_rebuf>0.000<
<pm><info><mp><x><devcontrol><youtube><rebuf_index>0.000<
<pm><info><mp><x><devcontrol><youtube><max_rebuf>0.000<
<pm><info><mp><x><devcontrol><youtube><mos>3.897<
<pm><info><mp><x><devcontrol><youtube><playback>7.194<
<pm><info><mp><x><devcontrol><youtube><total_rebuf_time>0.000<
<pm><info><mp><x><devcontrol><youtube><success>100.00<
<pm><info><mp><x><devcontrol><youtube><failed>0.00<
<pm><info><mp><x><devcontrol><youtube><total>1<
```

23.5.3 End of test Messages

Below are examples of "end of test" log messages.

```
StatsCalculationPhase
=====================
<pm><info><flow><x>KPIs computation starts
<pm><info><flow><x>KPIs computation time: 0 s<

<pm><info><general><end_time>2016-03-07 08h 32m 41s</end_time></general></info></pm> Test has ended
```

This indicates the process `PerformanceTester.exe` is about to finish.

24 Annex 11: MEC User Manual

Our extension has the goal of integrating the MEC paradigm in the TRIANGLE testbed. We consider MEC services composed of chains of Virtual Machines (VMs) potentially deployed in different Points of Presence (PoPs). The communication and information exchanged among VMs of the same service chain are provided by overlay networks, while a default public network provides the external connectivity to the Internet. An additional public network, called SG net, can be used to attach the service chain to the Serving Gateway (SGW) for interactions with the UE in the testbed.

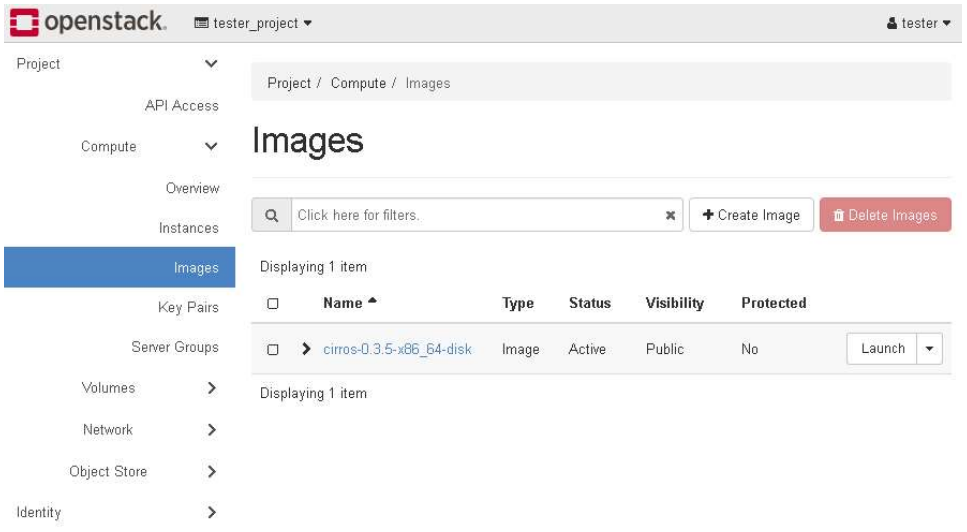
For the realization of the interface allowing experimenters to create their MEC service chains, we have used DevStack, a GitHub-based deployment of OpenStack that can run in a VM. The user interface is provided by means of the Horizon dashboard.

24.1 Creation of a Service Template

The experimenters can create service templates by using the Horizon dashboard to define the service chain and specify which components will be connected to the user personal network or, if needed, to additional back-end networks. Additional options can be defined, which will be described in the following step-by-step procedure.

24.1.1 Create an Image

To create a new image that will be made available for the creation of a service, go to the Horizon dashboard, select Project, then Compute, Images, and click on Create Image.



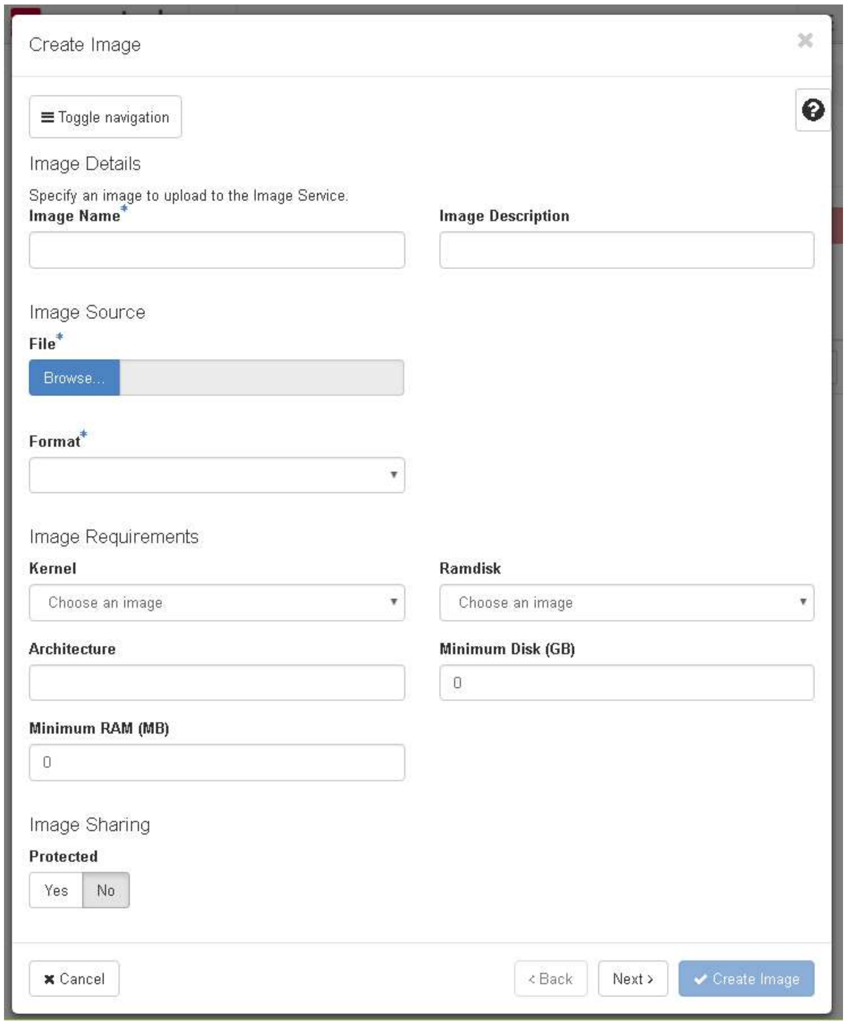
Displaying 1 item

	Name	Type	Status	Visibility	Protected
	cirros-0.3.5-x86_64-disk	Image	Active	Public	No

Displaying 1 item

Figure 138: Images management

Fill in the Image Name field, then click on Browse… to select the folder containing the image source file, and specify the format. Finally, click on Create Image to complete the operation.


Create Image

Toggle navigation

Image Details

Specify an image to upload to the Image Service.

Image Name*

Image Description

Image Source

File*

Browse...

Format*

Image Requirements

Kernel: Choose an image

Ramdisk: Choose an image

Architecture

Minimum Disk (GB): 0

Minimum RAM (MB): 0

Image Sharing

Protected: Yes [✓ No]

Cancel | < Back | Next > | Create Image

Figure 139: Images creation

24.1.2 Create the Personal and Back-end Networks

The experimenters can view and manage the network from the Network Topology tab, which initially shows only the public network.

To add a new network, select Project, then Network, Network Topology, and click on Create Network.

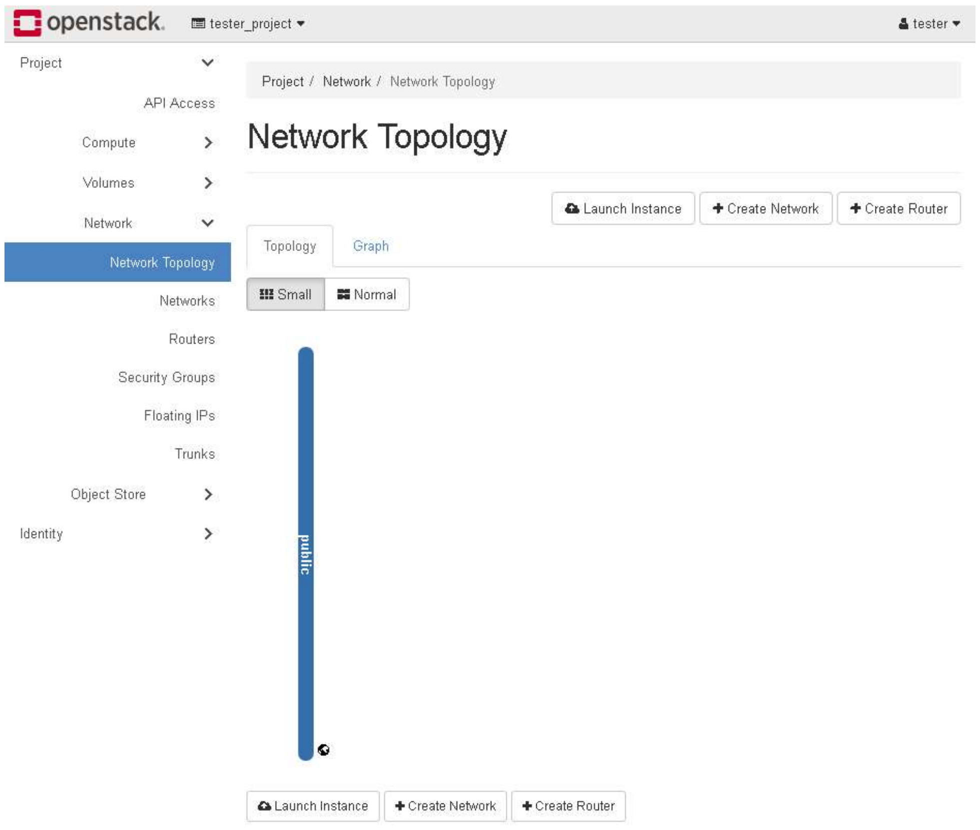
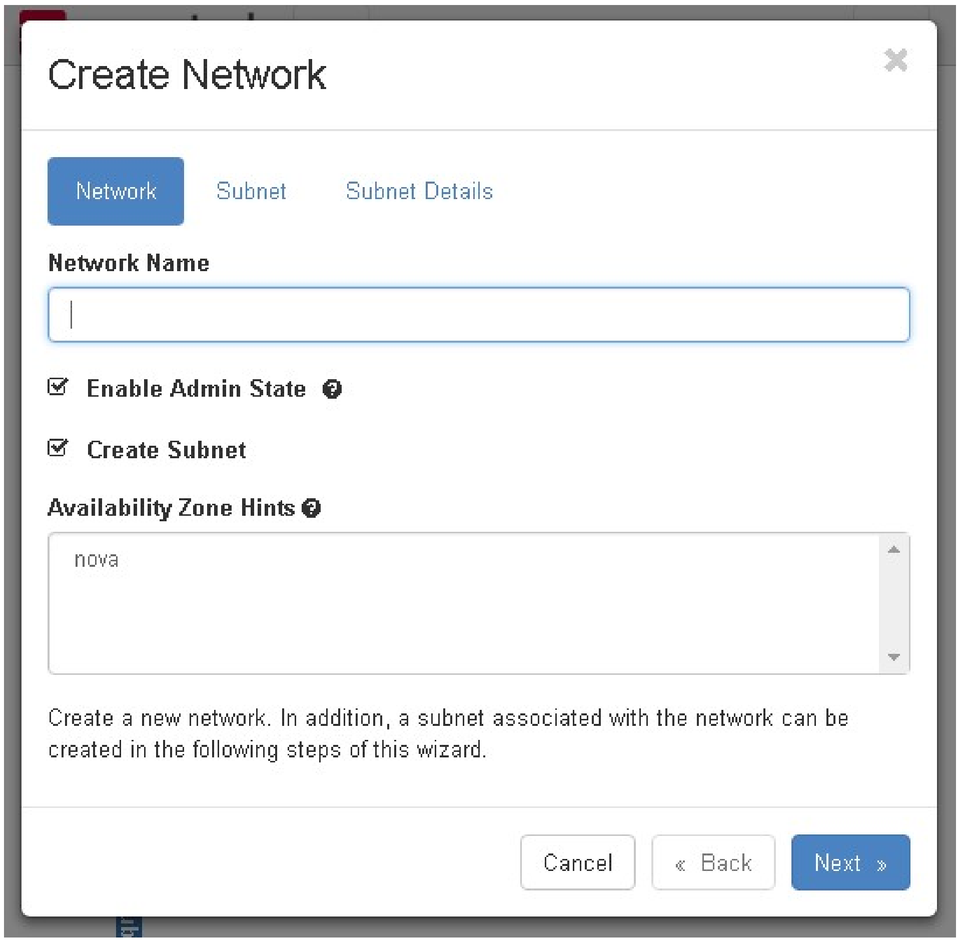


Figure 140: Network topology

In the Create Network dialog box, add Network Name and click on Next.


Create Network

Network Name

☑ Enable Admin State

☑ Create Subnet

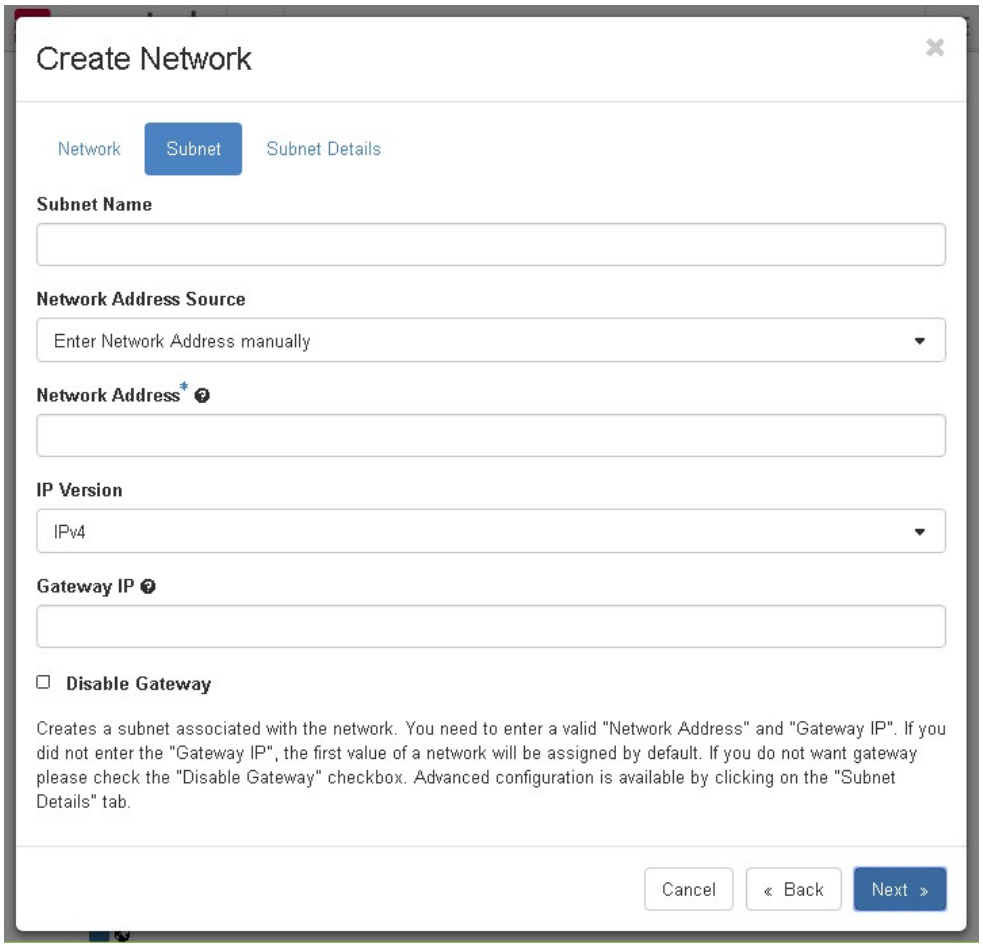
Availability Zone Hints

nova

Create a new network. In addition, a subnet associated with the network can be created in the following steps of this wizard.

Figure 141: Network topology

Add Subnet Name, Network Address and, if needed, Gateway IP. Click on Next.



Create Network

Subnet Name

Network Address Source: Enter Network Address manually

Network Address*

IP Version: IPv4

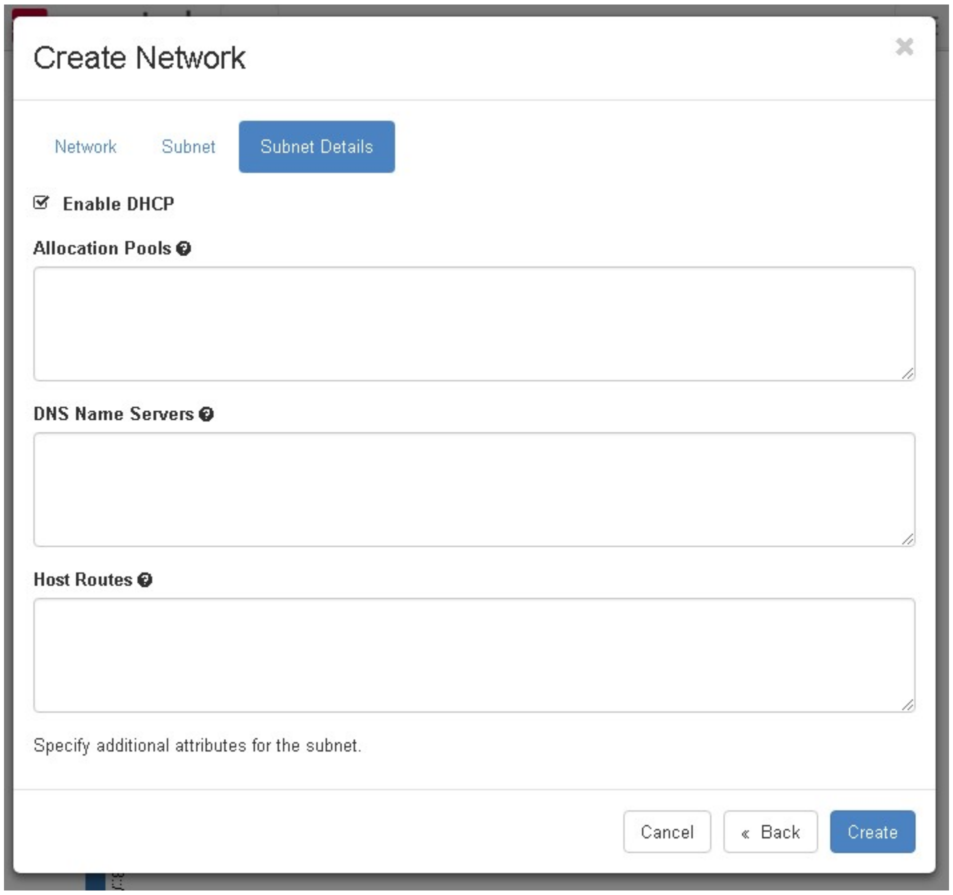
Gateway IP

☐ Disable Gateway

Creates a subnet associated with the network. You need to enter a valid "Network Address" and "Gateway IP". If you did not enter the "Gateway IP", the first value of a network will be assigned by default. If you do not want gateway please check the "Disable Gateway" checkbox. Advanced configuration is available by clicking on the "Subnet Details" tab.

Figure 142: Network creation (I)

Additional attributes related to the subnets can be specified in the Subnet Details tab.



☑ Enable DHCP

Allocation Pools

DNS Name Servers

Host Routes

Specify additional attributes for the subnet.

Figure 143: Network creation (II)

24.1.3 Bind the Service Instances to the BNs

Once all the networks have been created, the experimenters can proceed with the addition of the virtual machines that compose the service chain.

To add a service instance, select Project, then Network, Network Topology, and click on Launch Instance.

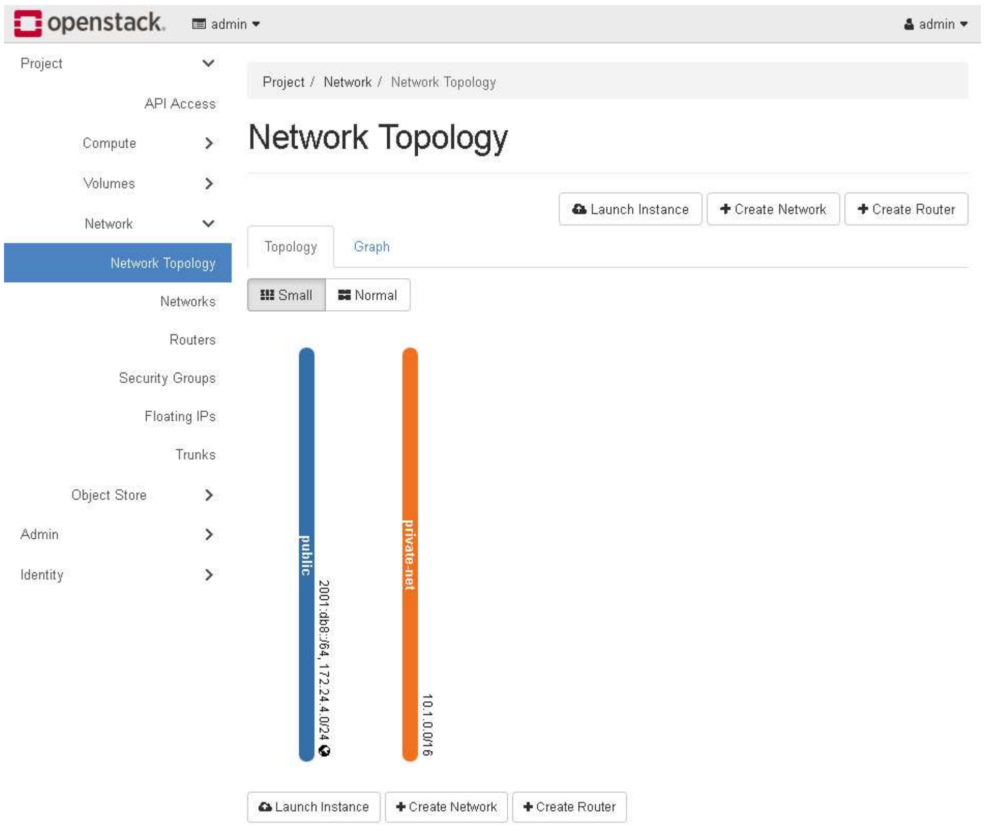
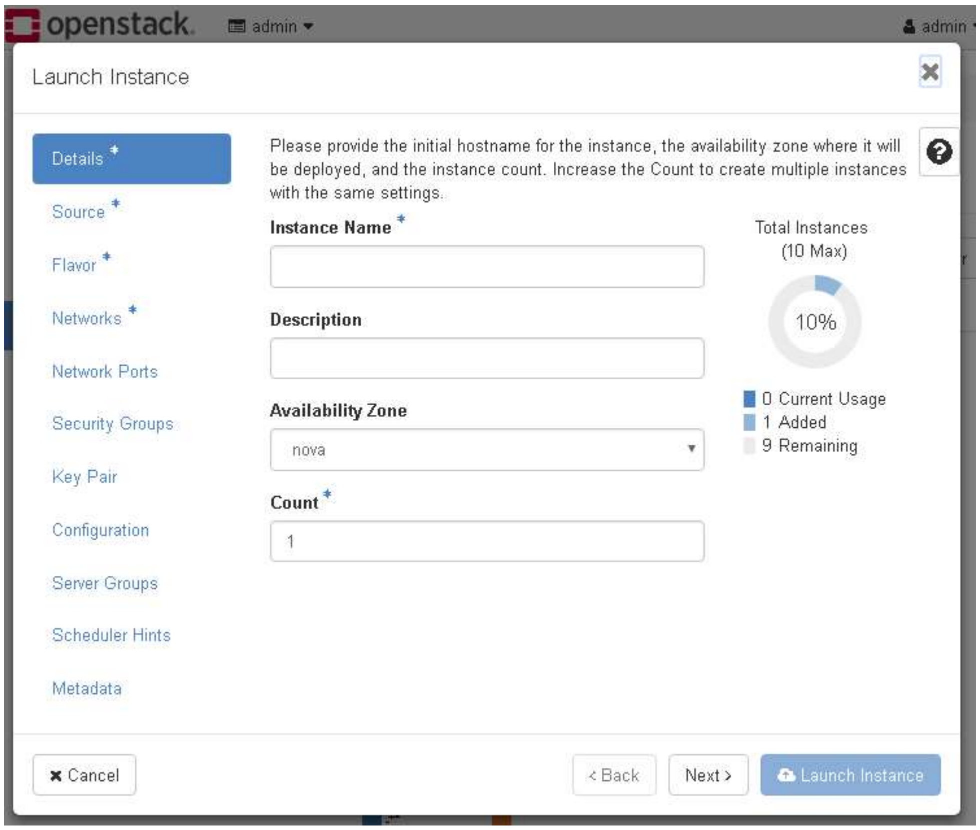


Figure 144: Network topology

In the Launch Instance dialog box, click on the Details tab and insert the Instance Name.



Please provide the initial hostname for the instance, the availability zone where it will be deployed, and the instance count. Increase the Count to create multiple instances with the same settings.

Instance Name *

Description

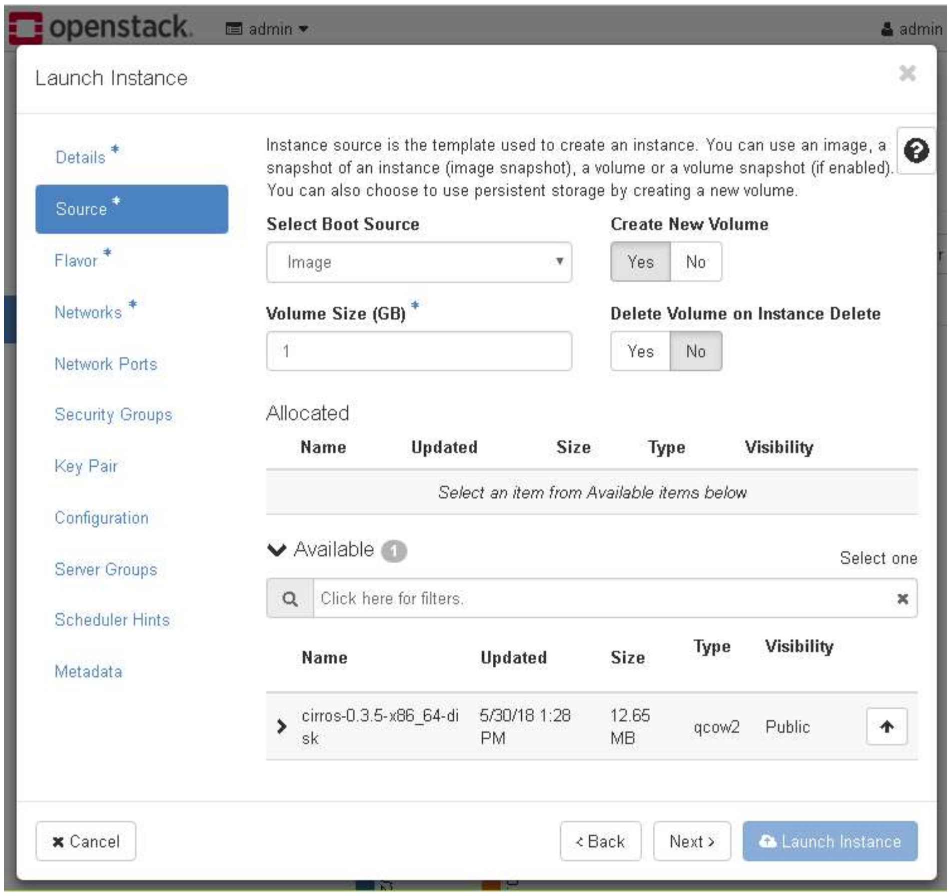
Availability Zone: nova

Count *: 1

Total Instances (10 Max)
10%
0 Current Usage
1 Added
9 Remaining

Figure 145: Instance configuration (I)

Click on the Source tab, select Image from the Select Boot Source drop-down menu, and then select an item from Available items by clicking on the up arrow next to the name.



Instance source is the template used to create an instance. You can use an image, a snapshot of an instance (image snapshot), a volume or a volume snapshot (if enabled). You can also choose to use persistent storage by creating a new volume.

Select Boot Source: Image

Create New Volume: [✓ Yes] No

Volume Size (GB) *: 1

Delete Volume on Instance Delete: Yes [✓ No]

Allocated

Name	Updated	Size	Type	Visibility
Select an item from Available items below				

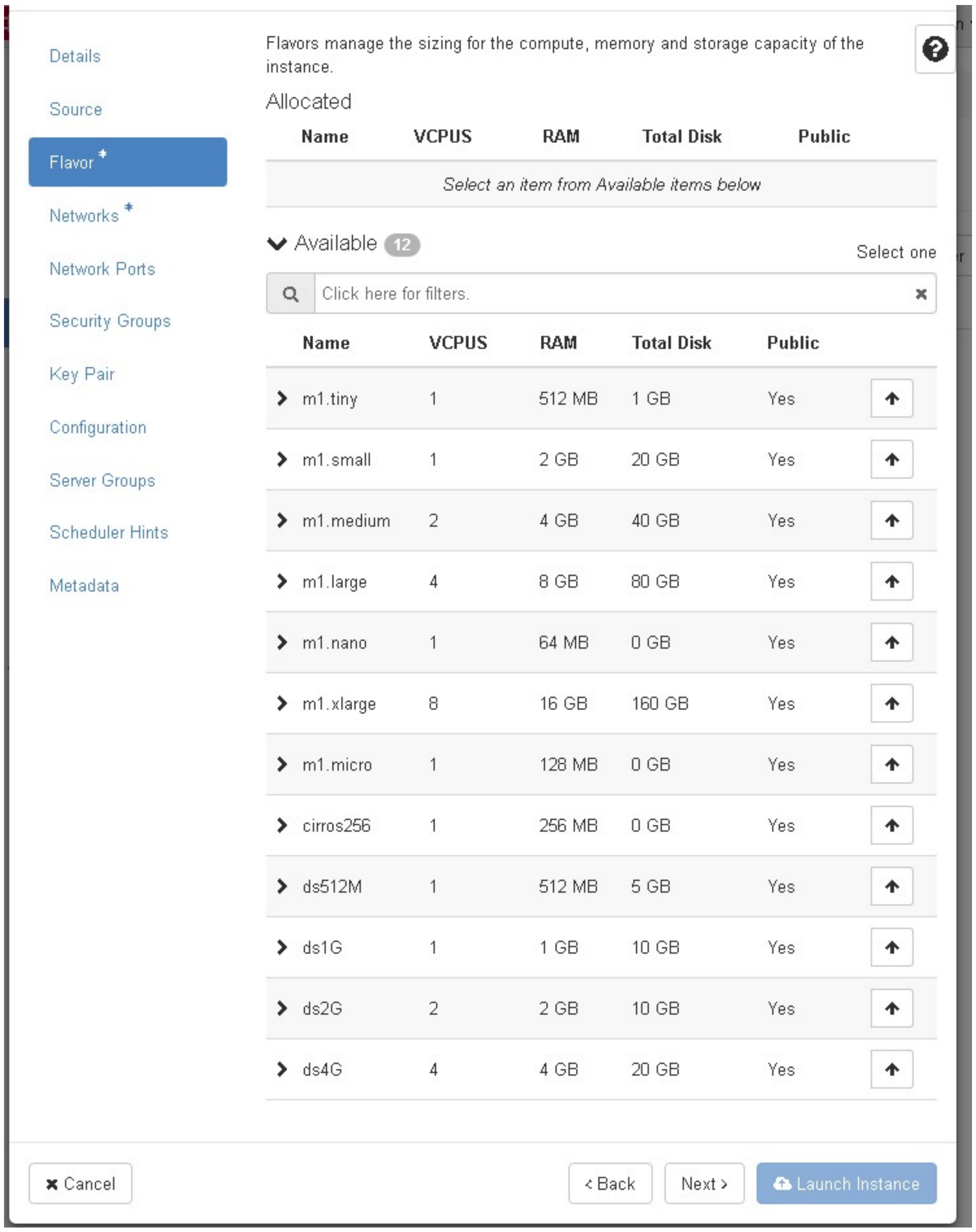
Available 1 Select one

Click here for filters.

Name	Updated	Size	Type	Visibility
cirros-0.3.5-x86_64-disk	5/30/18 1:28 PM	12.65 MB	qcow2	Public

Figure 146: Instance configuration (II)

Click on the Flavor tab and select an available flavor by clicking on the up arrow next to the name.



Flavors manage the sizing for the compute, memory and storage capacity of the instance.

Allocated

Name	VCPUS	RAM	Total Disk	Public
Select an item from Available items below				

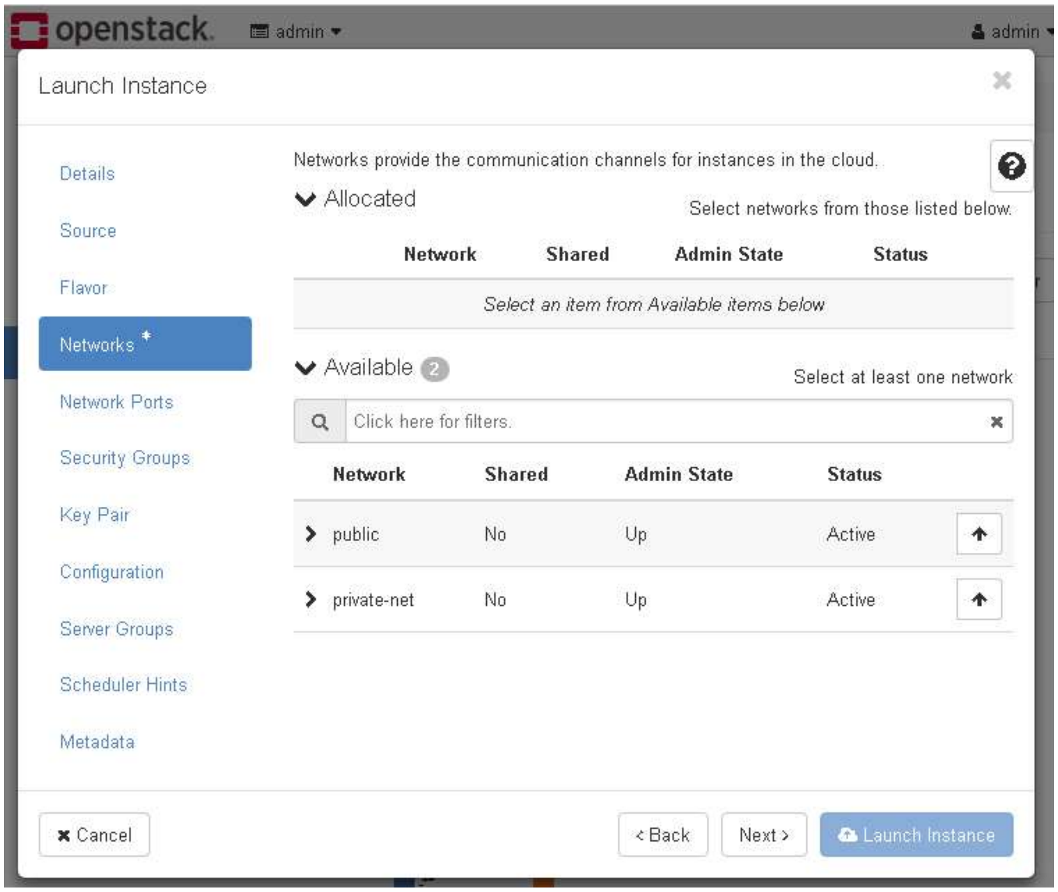
Available 12 Select one

Click here for filters.

Name	VCPUS	RAM	Total Disk	Public
m1.tiny	1	512 MB	1 GB	Yes
m1.small	1	2 GB	20 GB	Yes
m1.medium	2	4 GB	40 GB	Yes
m1.large	4	8 GB	80 GB	Yes
m1.nano	1	64 MB	0 GB	Yes
m1.xlarge	8	16 GB	160 GB	Yes
m1.micro	1	128 MB	0 GB	Yes
cirros256	1	256 MB	0 GB	Yes
ds512M	1	512 MB	5 GB	Yes
ds1G	1	1 GB	10 GB	Yes
ds2G	2	2 GB	10 GB	Yes
ds4G	4	4 GB	20 GB	Yes

Figure 147: Flavors

In the Networks tab, select the networks to be associated to the service instance and then click on Launch Instance. This operation entails the actual deployment of the service in the TRIANGLE testbed, and its orchestration is managed by means of the TAP plugin.



Networks provide the communication channels for instances in the cloud.

Allocated

Select networks from those listed below.

Network	Shared	Admin State	Status
Select an item from Available items below			

Available 2

Select at least one network

Click here for filters.

Network	Shared	Admin State	Status
public	No	Up	Active
private-net	No	Up	Active

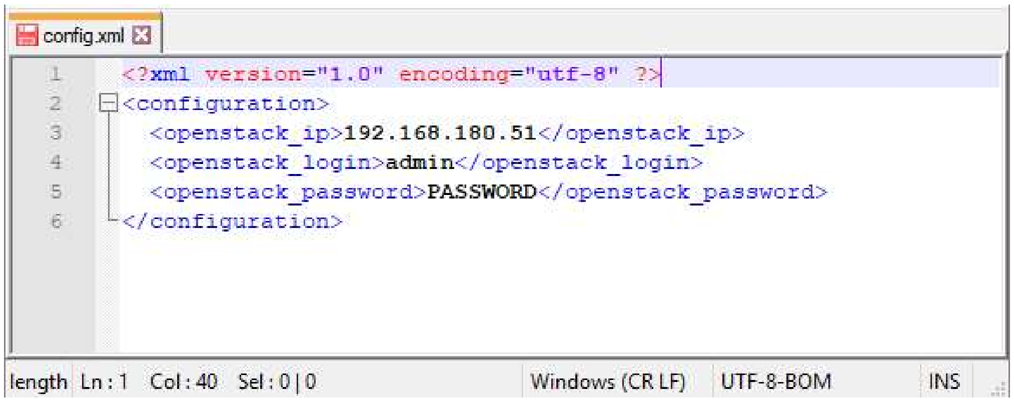
Figure 148: Instance configuration

24.2 MEC Service Setup and Orchestration

The Keysight Test Automation Platform (TAP) is used in the TRIANGLE testbed for configuring and running the test. We have designed a TAP plugin for our extension to start, stop, suspend and resume individual VMs, along with providing the authentication management. Additional operations related to migration could be added in the case of more than one server made available in the TRIANGLE testbed for the MEC extension. As final operation, a script is provided to cleanup the system and make it available for the following experiment.

24.2.1 Structure of the Plugin

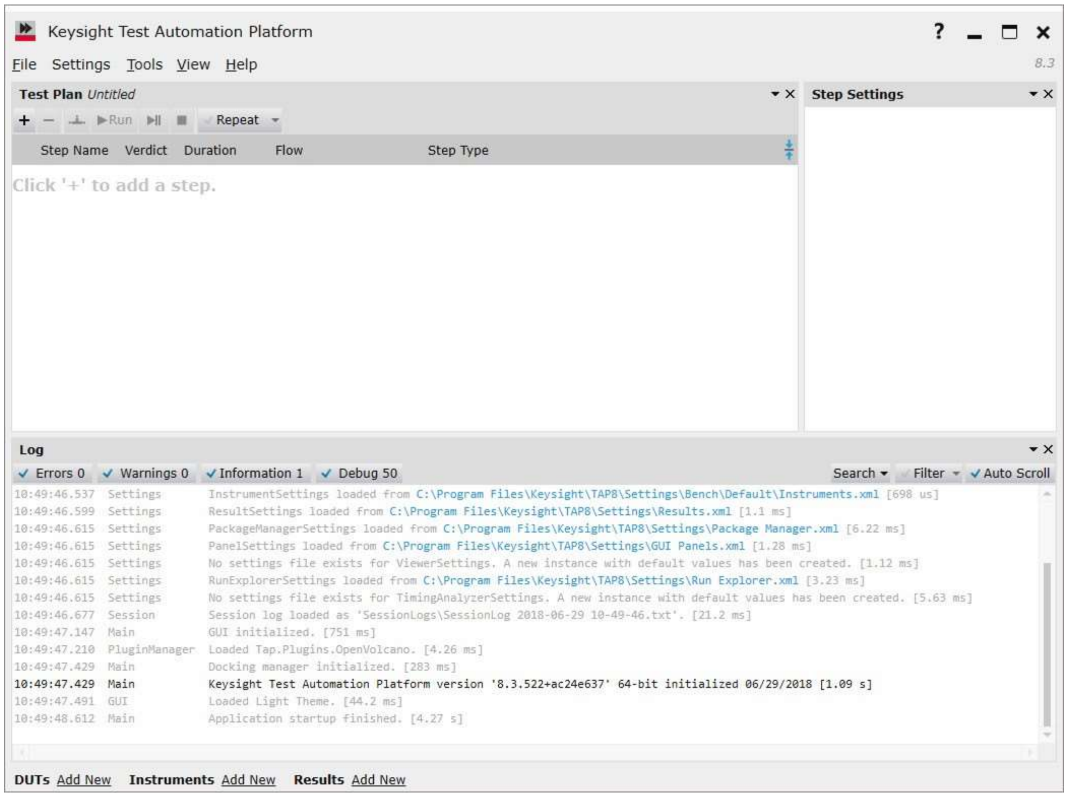
In the config.xml file insert IP address, login and password of the OpenStack server you want to control.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <openstack_ip>192.168.180.51</openstack_ip>
  <openstack_login>admin</openstack_login>
  <openstack_password>PASSWORD</openstack_password>
</configuration>
```

Figure 149: TAP plugin config file

In the TAP GUI, click on the + button to open the Test Steps menu.



Keysight Test Automation Platform
File Settings Tools View Help
8.3
Test Plan Untitled
Run
Repeat
Step Name Verdict Duration Flow Step Type
Click '+' to add a step.
Step Settings
Log
Errors 0 Warnings 0 Information 1 Debug 50
Search Filter Auto Scroll
DUTs Add New Instruments Add New Results Add New

```
10:49:46.537  Settings       InstrumentSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Bench\Default\Instruments.xml [698 us]
10:49:46.599  Settings       ResultSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Results.xml [1.1 ms]
10:49:46.615  Settings       PackageManagerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Package Manager.xml [6.22 ms]
10:49:46.615  Settings       PanelSettings loaded from C:\Program Files\Keysight\TAP8\Settings\GUI Panels.xml [1.28 ms]
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
```

Figure 150: TAP environment

Add steps by clicking on the corresponding Add button.

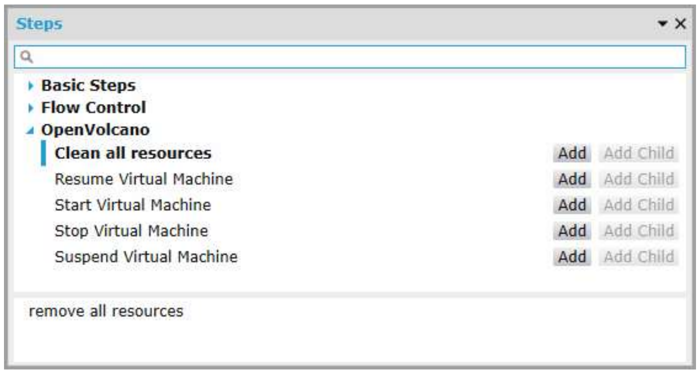
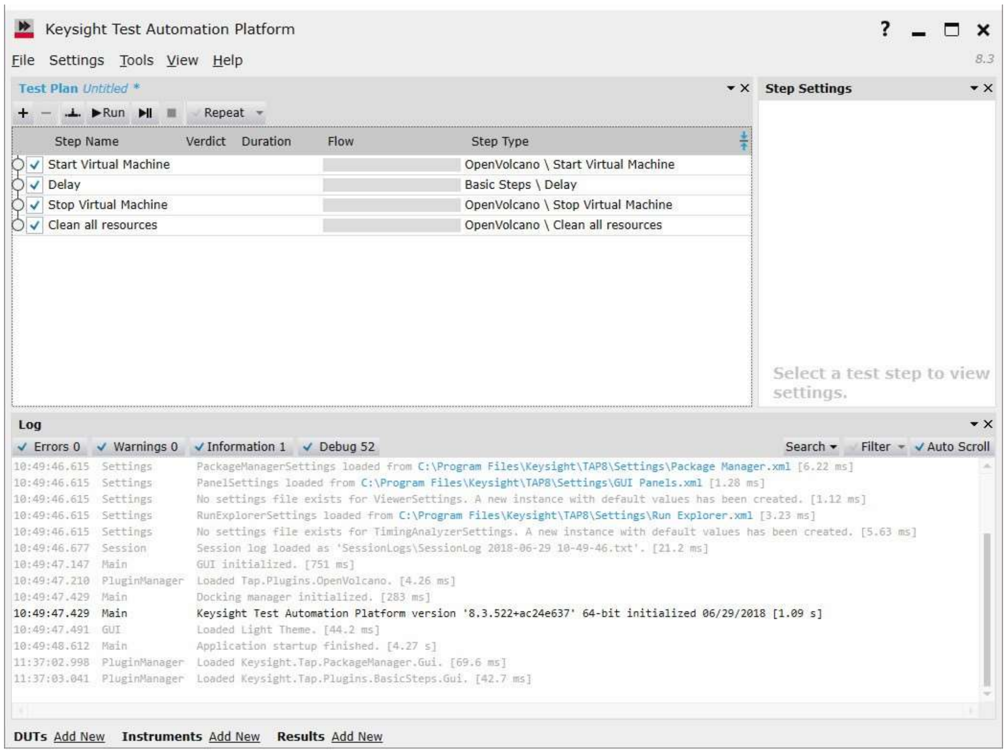


Figure 151: OpenVolcano plugin

The image below is an example of a test composed of four steps: Start VM, Delay, Stop VM, Cleanup.

The Run button executes the whole list of active steps (the ones checked in blue).


Keysight Test Automation Platform
File Settings Tools View Help
8.3
Test Plan Untitled *
Run
Repeat
Step Settings
Select a test step to view settings.
Log
Search
Filter
Auto Scroll
DUTs Add New Instruments Add New Results Add New

Step Name	Verdict	Duration	Flow	Step Type
✓ Start Virtual Machine				OpenVolcano \ Start Virtual Machine
✓ Delay				Basic Steps \ Delay
✓ Stop Virtual Machine				OpenVolcano \ Stop Virtual Machine
✓ Clean all resources				OpenVolcano \ Clean all resources

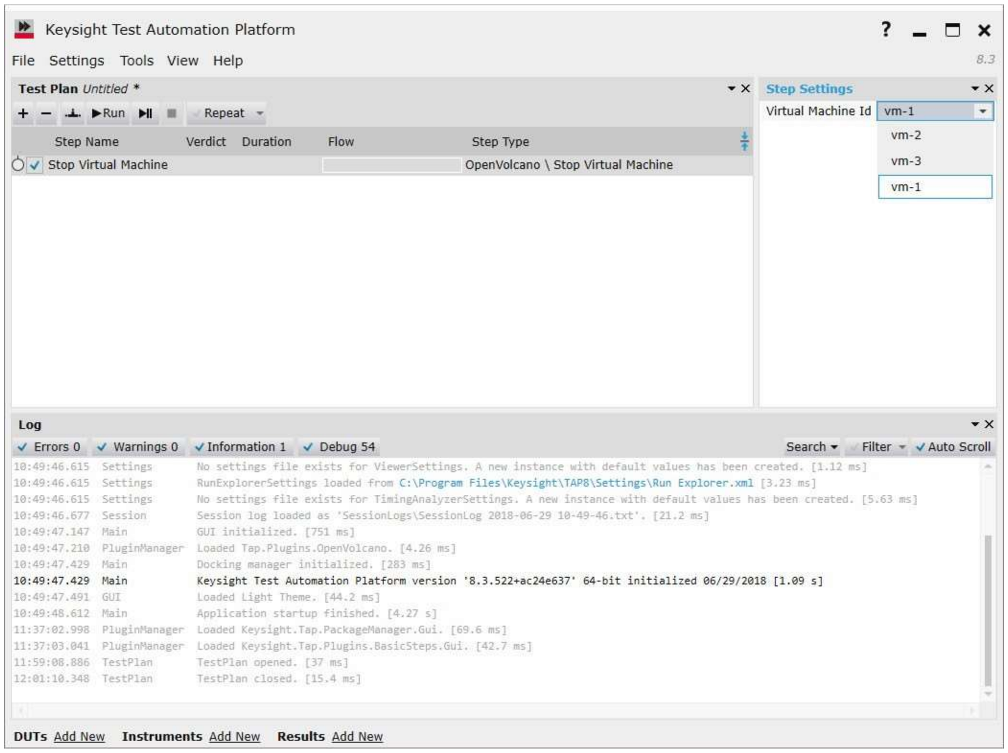
```
✓ Errors 0  ✓ Warnings 0  ✓ Information 1  ✓ Debug 52
10:49:46.615  Settings       PackageManagerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Package Manager.xml [6.22 ms]
10:49:46.615  Settings       PanelSettings loaded from C:\Program Files\Keysight\TAP8\Settings\GUI Panels.xml [1.28 ms]
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
11:37:02.998  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [69.6 ms]
11:37:03.041  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [42.7 ms]
```

Figure 152: TAP test plan example

24.2.2 Stop Service / “Stop Virtual Machine”

Since the operation of stopping a VM requires some time that depends on the nature of the VM, it is good practice to add a delay after the Stop command to be sure VM is actually stopped. To do this, click on the + button to open the Test Steps menu, select Delay, then click Add Child.

Add the Stop VM step, then go to Step Settings and choose to which VMs to apply the command.

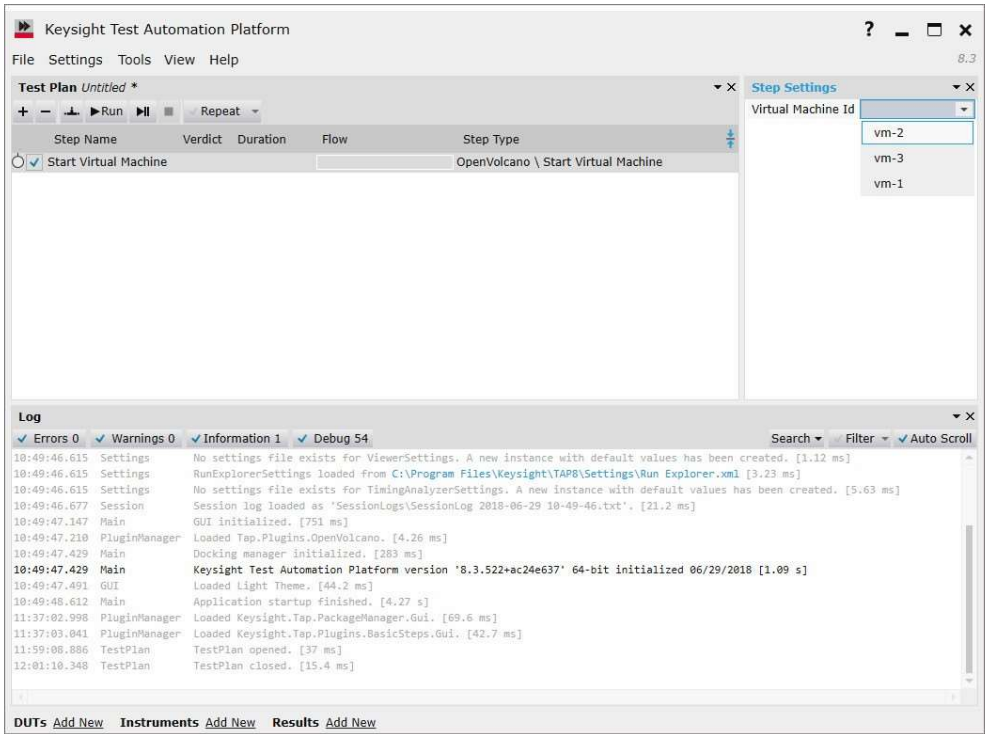

Keysight Test Automation Platform
File Settings Tools View Help
8.3
Test Plan Untitled *
Run
Repeat
Step Name Verdict Duration Flow Step Type
Stop Virtual Machine
OpenVolcano \ Stop Virtual Machine
Step Settings
Virtual Machine Id vm-1
vm-2
vm-3
vm-1
Log
Errors 0 Warnings 0 Information 1 Debug 54
Search Filter Auto Scroll
DUTs Add New Instruments Add New Results Add New

```
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
11:37:02.998  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [69.6 ms]
11:37:03.041  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [42.7 ms]
11:59:08.886  TestPlan       TestPlan opened. [37 ms]
12:01:10.348  TestPlan       TestPlan closed. [15.4 ms]
```

Figure 153: TAP plugin step (I)

24.2.3 Start Service / "Start Virtual Machine"

Add the **Start VM** step, then go to **Step Settings** and choose to which VMs to apply the command.



Virtual Machine Id

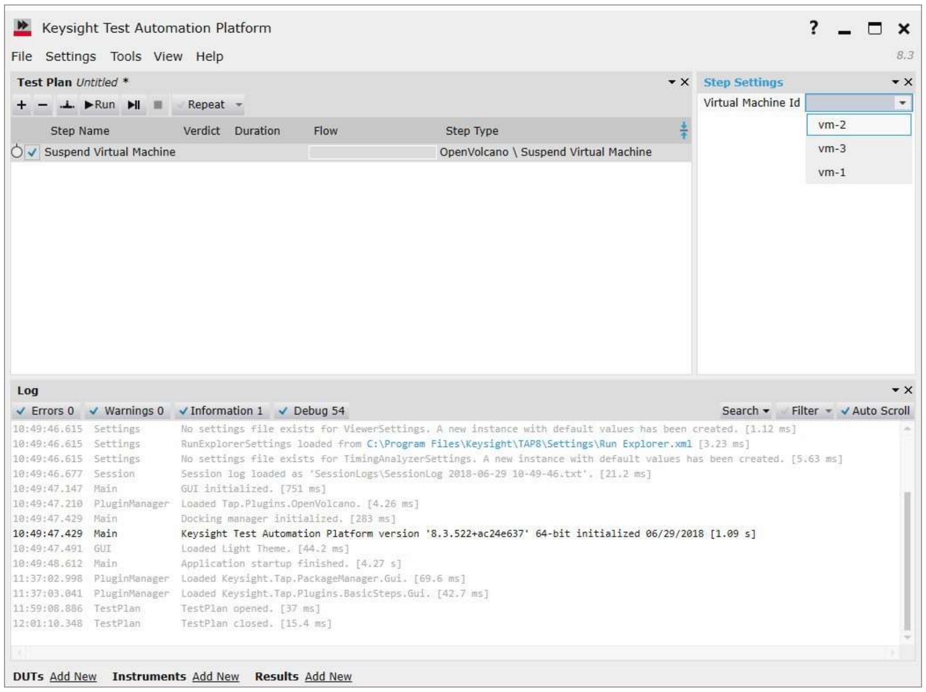
Step Name	Verdict	Duration	Flow	Step Type
Start Virtual Machine				OpenVolcano \ Start Virtual Machine

```
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
11:37:02.998  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [69.6 ms]
11:37:03.041  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [42.7 ms]
11:59:08.886  TestPlan       TestPlan opened. [37 ms]
12:01:10.348  TestPlan       TestPlan closed. [15.4 ms]
```

Figure 154: TAP plugin step (II)

24.2.4 Suspend Service / "Suspend Virtual Machine"

Add the **Suspend VM** step, then go to **Step Settings** and choose to which VMs to apply the command.



Virtual Machine Id

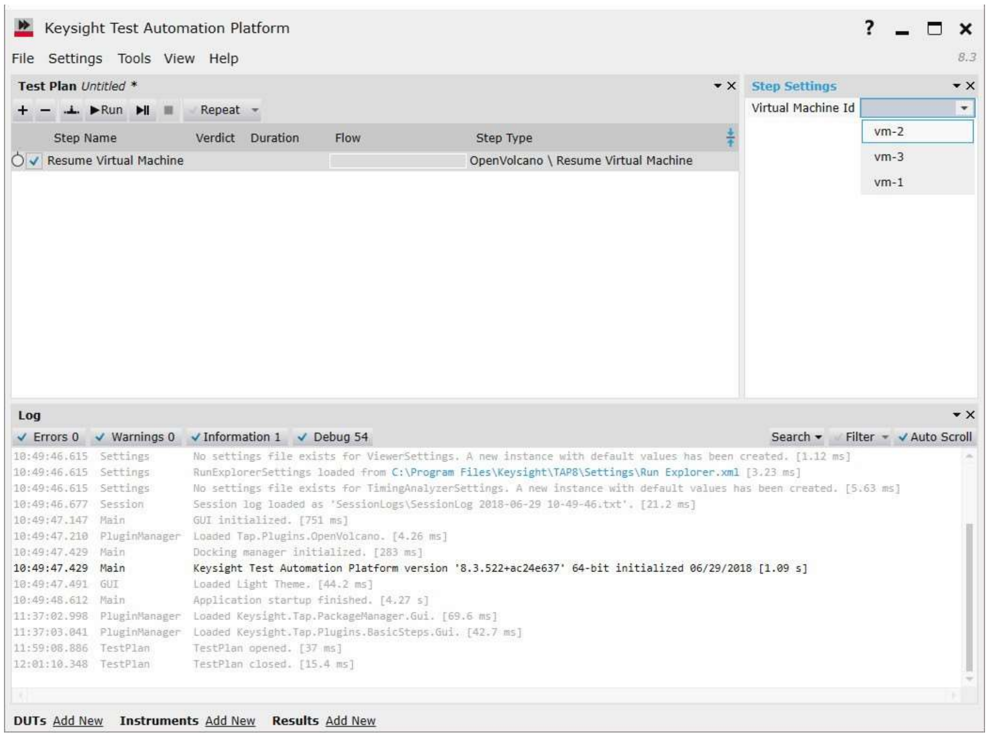
Step Name	Verdict	Duration	Flow	Step Type
Suspend Virtual Machine				OpenVolcano \ Suspend Virtual Machine

```
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
11:37:02.998  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [69.6 ms]
11:37:03.041  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [42.7 ms]
11:59:08.886  TestPlan       TestPlan opened. [37 ms]
12:01:10.348  TestPlan       TestPlan closed. [15.4 ms]
```

Figure 155: TAP plugin step (III)

24.2.5 Resume Service / "Resume Virtual Machine"

Add the **Resume VM** step, then go to **Step Settings** and choose to which VMs to apply the command.


Keysight Test Automation Platform
File Settings Tools View Help
8.3
Test Plan Untitled *
Run
Repeat
Step Name Verdict Duration Flow Step Type
Resume Virtual Machine
OpenVolcano \ Resume Virtual Machine
Step Settings
Virtual Machine Id
vm-2
vm-3
vm-1
Log
Errors 0 Warnings 0 Information 1 Debug 54
Search Filter Auto Scroll
DUTs Add New Instruments Add New Results Add New

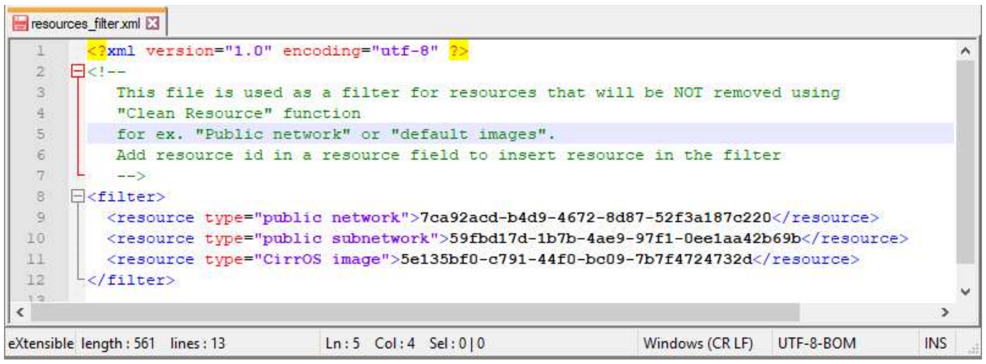
```
10:49:46.615  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.12 ms]
10:49:46.615  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.23 ms]
10:49:46.615  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [5.63 ms]
10:49:46.677  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 10-49-46.txt'. [21.2 ms]
10:49:47.147  Main           GUI initialized. [751 ms]
10:49:47.210  PluginManager  Loaded Tap.Plugins.OpenVolcano. [4.26 ms]
10:49:47.429  Main           Docking manager initialized. [283 ms]
10:49:47.429  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.09 s]
10:49:47.491  GUI            Loaded Light Theme. [44.2 ms]
10:49:48.612  Main           Application startup finished. [4.27 s]
11:37:02.998  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [69.6 ms]
11:37:03.041  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [42.7 ms]
11:59:08.886  TestPlan       TestPlan opened. [37 ms]
12:01:10.348  TestPlan       TestPlan closed. [15.4 ms]
```

Figure 156: TAP plugin step (IV)

24.2.6 Clean Up testbed

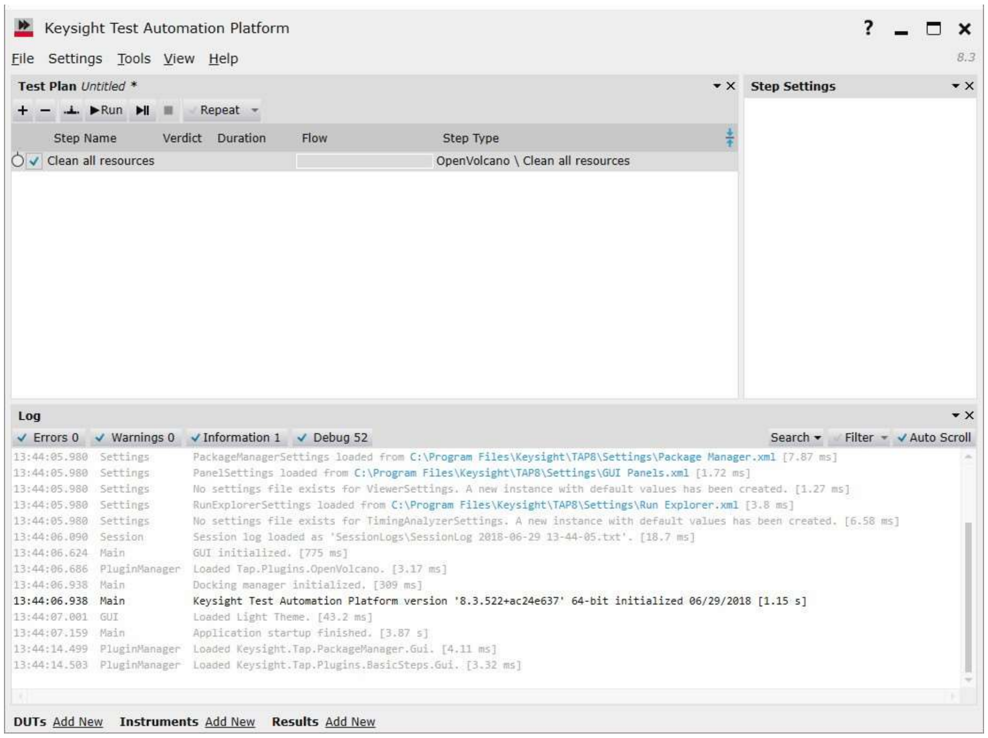
Since the **Clean Up** step deletes the whole content of the project, before launching it make sure to fill the *resource_filter.xml* file (xml format) with all the objects (e.g., networks, servers, interfaces, etc.) that still need to be available for the following run. Such objects include, at the very least the **public** and the **SG net** networks.

In *resource_filter.xml,* objects in the **resource** field are identified by their OpenStack ID as indicated in the dashboard. The type **field** is actually not utilized but useful for a more "user-friendly" identification of the objects.



```
<?xml version="1.0" encoding="utf-8" ?>
<!--
   This file is used as a filter for resources that will be NOT removed using
   "Clean Resource" function
   for ex. "Public network" or "default images".
   Add resource id in a resource field to insert resource in the filter
   -->
<filter>
  <resource type="public network">7ca92acd-b4d9-4672-8d87-52f3a187c220</resource>
  <resource type="public subnetwork">59fbd17d-1b7b-4ae9-97f1-0ee1aa42b69b</resource>
  <resource type="CirrOS image">5e135bf0-c791-44f0-bc09-7b7f4724732d</resource>
</filter>
```

Figure 157: TAP plugin step (V)


Keysight Test Automation Platform
File Settings Tools View Help
Test Plan Untitled *
Step Settings
Run
Repeat
Step Name Verdict Duration Flow Step Type
Clean all resources
OpenVolcano \ Clean all resources
Log
Errors 0 Warnings 0 Information 1 Debug 52
Search Filter Auto Scroll
DUTs Add New Instruments Add New Results Add New

```
13:44:05.980  Settings       PackageManagerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Package Manager.xml [7.87 ms]
13:44:05.980  Settings       PanelSettings loaded from C:\Program Files\Keysight\TAP8\Settings\GUI Panels.xml [1.72 ms]
13:44:05.980  Settings       No settings file exists for ViewerSettings. A new instance with default values has been created. [1.27 ms]
13:44:05.980  Settings       RunExplorerSettings loaded from C:\Program Files\Keysight\TAP8\Settings\Run Explorer.xml [3.8 ms]
13:44:05.980  Settings       No settings file exists for TimingAnalyzerSettings. A new instance with default values has been created. [6.58 ms]
13:44:06.090  Session        Session log loaded as 'SessionLogs\SessionLog 2018-06-29 13-44-05.txt'. [18.7 ms]
13:44:06.624  Main           GUI initialized. [775 ms]
13:44:06.686  PluginManager  Loaded Tap.Plugins.OpenVolcano. [3.17 ms]
13:44:06.938  Main           Docking manager initialized. [309 ms]
13:44:06.938  Main           Keysight Test Automation Platform version '8.3.522+ac24e637' 64-bit initialized 06/29/2018 [1.15 s]
13:44:07.001  GUI            Loaded Light Theme. [43.2 ms]
13:44:07.159  Main           Application startup finished. [3.87 s]
13:44:14.499  PluginManager  Loaded Keysight.Tap.PackageManager.Gui. [4.11 ms]
13:44:14.503  PluginManager  Loaded Keysight.Tap.Plugins.BasicSteps.Gui. [3.32 ms]
```

Figure 158: TAP plugin step (VI)

24.2.7 Available Files and Scripts

- config.xml: login, password and IP of the OpenStack server
- resources_filter.xml: script used to cleanup the DevStack user
- Tap.Plugins.OpenVolcano.dll: TAP plugin