



Document: ICT-688712-TRIANGLE/D3.2

Date: 20/07/2018

Dissemination: PU

Status: Final

Version: 1.1

Project: H2020-ICT-688712

Project Name:

5G Applications and Devices Benchmarking (TRIANGLE)

Deliverable D3.2

Report on the implementation of testing framework Release 2 and specification of testing framework Release 3

Date of delivery: 20/07/2018

Version: 1.1

Start date of Project: 01/01/2016

Duration: 36 months



Deliverable D3.2

Report on the implementation of testing framework Release 2 and specification of testing framework Release 3

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

Document Number:	ICT-688712-TRIANGLE/D3.1
Document Title:	Progress report on the testing framework Release 2 and specification of Release 3
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 (Alberto Salmerón, Almudena Díaz, Pedro Merino, Cesar A. García, Laura Panizo Jaime, Bruno García, Guillermo Chica, Verónica Tapia, 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:	30/09/2017
Work Package Leader:	Universidad de Málaga
Status:	Final
Version:	1.1
File Name:	TRIANGLE_Deliverable_D3.2_V1.1_FINAL

Abstract

This deliverable provides the description of the second release of the TRIANGLE testbed. It introduces current features provided by the TRIANGLE Portal and components integrated into the testbed. The document also introduces the specification of the features for the next release.

Keywords

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



Document history

V1.0	Initial Release of the document
V1.1	<p>Figure 2 has been updated in order to clarify that the Portal is only used for app testing.</p> <p>Section 9 has been updated with clarifications about how the user equipments need to be connectorized to be integrated into the testbed.</p>



Executive summary

This document is the second deliverable of WP3. WP3 is responsible for the development of the testing framework in TRIANGLE. Testing framework covers all the software, coordination/sequencing that control & connects to the test infrastructure. It is charge of handling and converting the end user test requests into actionable steps within the software and hardware portion of the testbed.

The document describes the second implementation of the TRIANGLE testbed and a complete description of the features and the control interface implemented to manage each one of the components of the testbed. The content of this document will be updated in D3.4, which will introduce the advances in the architecture and the features provided by testbed to the date of publication.

The focus of the document is to provide a clear understanding of how the testbed is being implemented to deliver testing and certification services to app developers, device makers and researchers. Release 2 of the testbed is focused on the improvement of the functionality offered by the Portal, improvements in the orchestration, more sophisticated network scenarios, and a more efficient integration of the tools and the addition of new features. The document also includes the results of a test executed to check the integration of the current components.



Contents

1	Introduction	1
2	TRIANGLE testbed architecture	3
2.1	TRIANGLE testbed workflow and architecture overview.....	3
3	Interface and visualization (Portal)	5
3.1	Information provided by users	5
3.2	Backend REST API.....	7
4	Orchestration.....	8
4.1	Orcomposutor	8
4.2	Test Automation platform (TAP)	9
4.3	New features in the Quamotion WebDriver	14
5	Measurements and data collection	17
5.1	KPIs computation	17
5.2	Metrics and mark computation	17
5.3	Instrumentation library.....	17
6	RAN (Radio Access Network)	19
7	EPC	20
7.1	EPC Architecture.....	20
7.2	EPC Triggered Procedures	20
8	Transport.....	22
9	UE (User Equipment and accessories).....	24
9.1	Supported UEs.....	24
10	Local applications and servers	26
10.1	DANE (DASH-Aware Network Element).....	26
11	Extensions and new features	29
11.1	GPS emulation.....	29
11.2	VR Applications Testing	33
11.3	Model Based Testing	42
11.4	Traffic Impairments	47
11.5	Remote PCAP Capabilities	48
12	Internal test experiment.....	49
12.1	Testbed setup	49
12.2	Test Configuration.....	49
12.3	Initial Measurements	50
13	TRIANGLE testbed Release 3 specifications	52
13.1	Interface and visualization (Portal)	52



13.2	Orchestration	52
13.3	Measurement and data collection.....	53
13.4	User equipment and accessories	53
13.5	Extensions and new features	53
14	References.....	54
15	Appendix 1: Portal API REST	55
15.1	Devices	55
15.2	Users	55
15.3	Apps.....	56
15.4	Features.....	57
15.5	Test cases.....	58
15.6	Scenarios	60
15.7	Campaigns.....	60
16	Appendix 2: TAP plugins. Implementation details.....	61
16.1	Quamotion WebDriver TAP plugin.....	61
16.2	OML TAP plugin.....	67
16.3	App instrumentation TAP plugin	70
16.4	Android TAP plugin	71
16.5	iOS TAP plugin	75
16.6	Impairments TAP plugin	78
16.7	RF Switch TAP plugin	79
16.8	GPS emulation TAP plugin.....	81
16.9	Dynamic Sequence plugin.....	84
16.10	Iteration-aware result listener plugin.....	85
16.11	KPIs calculation plugin	85
16.12	Plugins to reach TAP server.....	85
16.13	DEKRA TAP plugin	86
16.14	VELOX plugin	90
17	Appendix 3: Android Instrumentation Library Usage.....	93
17.1	Including in an Android project	93
17.2	Usage	94
17.3	Example.....	94
17.4	Using the Android Instrumentation Library from Unity	95
17.5	Retrieving messages.....	96
17.6	Measurements format	97



List of Figures

Figure 1 High-Level architecture of the testbed	1
Figure 2 Testbed workflow	3
Figure 3 Control and management entities of the testbed	4
Figure 4 Uses cases.....	5
Figure 5 Features selection and measurements points	6
Figure 6 External Parameter setting & command line execution.....	10
Figure 7 Test plan flow chart & TAP implementation	11
Figure 8 Hardware and software components of the testbed for Release 2.....	12
Figure 9: Quamotion WebDriver script editor.....	14
Figure 10 EPC architecture	20
Figure 11: MANO deployment architecture.....	22
Figure 12 User equipment connectorization	24
Figure 13 iPhone 7 plus	25
Figure 14 Samsung Galaxy S7.....	25
Figure 15 Contact antenna	26
Figure 16 SAND architecture within the TRIANGLE testbed, including DANE, DASH streaming server and DASH VR app.....	27
Figure 17 Setting the DANE's bandwidth within a TAP's test scenario	28
Figure 18 Connecting USRP output to the UE GPS antenna.....	30
Figure 19 Google Maps route and XML file	33
Figure 20 VR test module architecture	35
Figure 21 OpenCV system reference	38
Figure 22 VR test solution validation	40
Figure 23 VR validation	40
Figure 24 Universal Music Player model	43
Figure 25 Universal Music Player GUI.....	44
Figure 26 Extract of Promela specification for test case generation.....	45
Figure 27 Pruning never claim as automaton	46
Figure 28 - CDF of the CPU usage [%] for YouTube API and ExoPlayer.....	50
Figure 29 - CDF of the consumed Power [W] for YouTube and ExoPlayer	50
Figure 30 - Cumulative data received by YouTube and ExoPlayer	51
Figure 31 Main classes of TAP plugin for iOS	78
Figure 32 URSP instrument.....	83
Figure 33 PrepareEmulation step	83
Figure 34 StartEmulation step	83



Figure 35 Dynamic Sequence to enable master & slave sequences	85
Figure 36 DEKRA Tool TAP Instrument	87
Figure 37 DEKRA Tool TAP DUT	88
Figure 38 DEKRA Tool TAP Test Step (YouTube test).....	89
Figure 39 DEKRA Tool TAP Run/Stop Test	89
Figure 40 - Expected VELOX Step Order	92



List of Tables

Table 1 - Orcompositor REST API	9
Table 2 – TAP plugins implemented in the Release 2	12
Table 3 – Commands provided by the Quamotion Webdriver script editor	15
Table 4: EPC Elements loopback IPs and not depicted interfaces.....	20
Table 5: OPNFV FUEL installer details.....	22
Table 6 Current status of devices integrated into the testbed	26
Table 7 – VR Application User Experience Key Performance Indicators	34
Table 8 – Robotic arm position range and reference	35
Table 9 – Robotic arm reference values	37
Table 10 – Validation results for determining matching threshold	41
Table 11 – AUE/VR/001 Validation results	41
Table 12 App user flow generation - Experiments	47
Table 13 Quamotion WebDriver instrument settings.....	61
Table 14 Quamotion WebDriver instrument methods	61
Table 15 Quamotion WebDriver New session step settings	62
Table 16 Quamotion WebDriver Close session step settings	63
Table 17 Quamotion WebDriver User action steps settings.....	63
Table 18 Quamotion WebDriver Enter text step settings	64
Table 19 Quamotion WebDriver Query step settings.....	64
Table 20 Quamotion WebDriver Query step results.....	66
Table 21 Quamotion WebDriver Replay step settings	66
Table 22 OMLServerInstrument settings	67
Table 23 OMLServerInstrument public properties	67
Table 24 OMLServerInstrument methods.....	67
Table 25 ICsInstruments public properties	68
Table 26 OML TAP plugin Configure OML step settings	68
Table 27 OML TAP plugin Send CSV step settings	69
Table 28 OML result listener settings	70
Table 29 Settings for the Parse Measurements step	70
Table 30 Settings for the Parse Regex in Logcat step	71
Table 31 Android TAP plugin instruments	71
Table 32 Settings shared by the Android steps	72
Table 33 Settings for the Custom command on the Adb Command step.....	72
Table 34 Settings for the Custom command on the Adb Command step.....	72
Table 35 Settings for the Custom command on the Adb command step.....	73



Table 36 Settings for the Custom command on the Adb Command step.....	73
Table 37 Settings for the Start, StartService and Broadcast commands on Activity Manager	73
Table 38 Settings for the Force Stop command on Activity Manager.....	74
Table 39 Settings for Adb Airplane Mode step.....	74
Table 40 Settings for the Logcat step	74
Table 41 Settings for the Retrieve Background Logcat step	75
Table 42 - iOS TAP plugin instrument	76
Table 43 - iOS TAP plugin test steps.....	76
Table 44 Settings for the Set Link Impairments step.....	78
Table 45 Settings for the Reset Impairments step	79
Table 46 Settings for the Retrieve Background Logcat step	80
Table 47 Settings for the Open Path step.....	80
Table 48 Settings for the Connector switching step	80
Table 49 – GPS API Rest.....	82
Table 50 – DEKRA Tool RC Server channel	86
Table 51 – DEKRA Tool TAP Instrument.....	86
Table 52 – DEKRA Tool TAP DUT	87
Table 53 – DEKRA Tool TAP Test Steps	88
Table 54 – DEKRA Tool RC Server Get Results	89
Table 55 - VELOX TAP Steps	91



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-everything
WCDMA	Wide Code Division Multiple Access
WLAN	Wireless Local Area Network
WP	Work Package
WPAN	Wireless Personal Area Networks

1 Introduction

D3.1 described in detail all the instruments, tools and hardware composing the testbed and the outcome of the first integration of some of these components. For the configuration, control and coordination of the components an orchestration framework, based on TAP, was defined. The general architecture of the testbed (shown in Figure 1) the orchestration of the framework and the components of the testbed were all documented in D3.1.

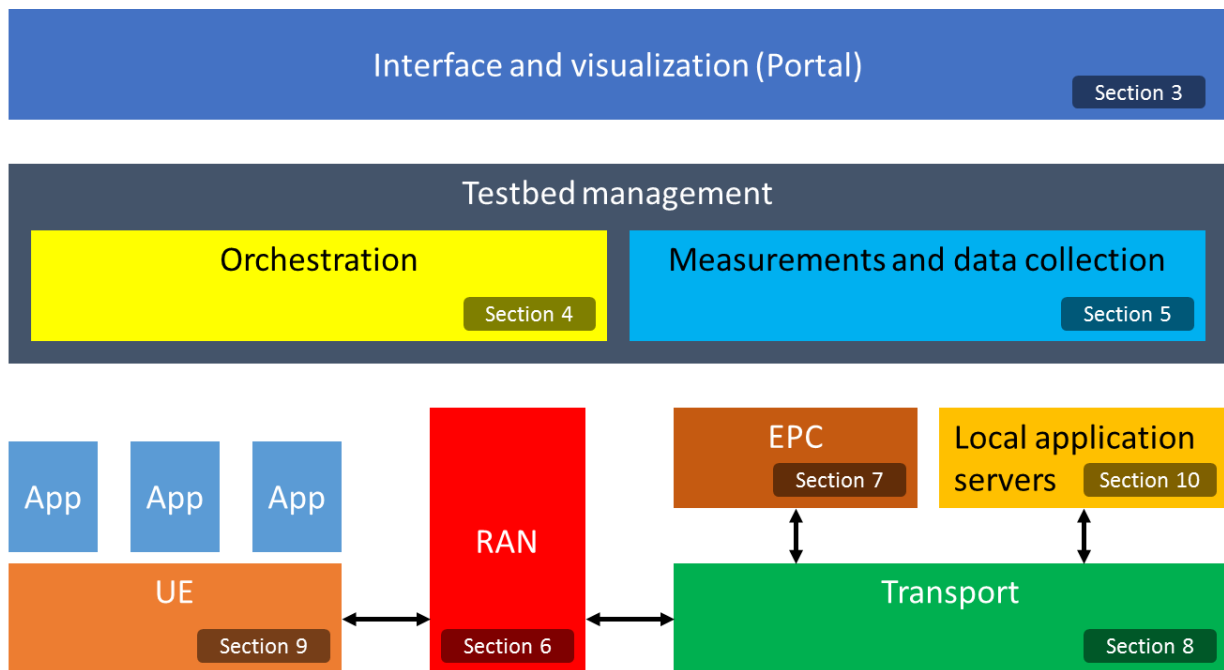


Figure 1 High-Level architecture of the testbed

In D3.2, we will follow this high-level architecture to introduce the innovations and improvements introduced in each of their components. The document starts with Section 2 which provides a quick overview of the general workflow of the testbed so that the reader can better understand the internal functioning of the system.

Section 3 introduces the improvements carried out in the Portal. The Portal is the main entry point to the testbed for application developers. In the initial version of the testbed a very limited set of functionalities was exposed through the Portal. In this version of the testbed (Release 2), and once the applications are uploaded, developers can declare a set of features about their applications. In accordance with these features the Portal offers the code snippets the developer should introduce into the code of their applications to measure the KPIs associated to the declared features. In this sense, the Portal offers a common approach to measure the QoE of applications offering the same or similar features. Custom KPI can also be defined using the instrumentation library (see Section 5).

The key components of the orchestration framework are explained in section 4. As already introduced in D3.1, each elements that needs to be configured and controlled as part of the testbed has an associated TAP driver. Section 4 explains all the drivers developed so far. These drivers provide the configuration and control steps used to compose the TAP test plan templates used to execute the test campaigns defined in the Portal. Moreover, researchers and device markers can use these drivers directly to define their own customized TAP test plans. The TRIANGLE testbed offers to device markers and researchers a virtual machine hosted in the



testbed. Through the TAP environment installed in these virtual machines the experimenters have a secure and flexible access to all the components of the testbed.

Section 5 introduces advances to support the collection of measurements which enable the computation of QoE metrics. The first release of the testbed (Release 1) offered isolated measurements of CPU, RAM, traffic, power consumption, etc. In this release (Release 2) we have implemented a first version of the instrumentation library defined in D3.1. This library enables to link all the measurements collected by the testbed with the features exercised at any time by the application, for example it is able to measure the power consumption while playing a video. Moreover, this library enables to measure internal delays of the application when using these features, being delays a key factor when computing QoE. A more detailed description of the ETL module which computes the KPIs and the TRIANGLE mark will be provided in D3.3.

Sections 6, 7, 8, 9 and 10 describe the novelties introduced in each one of the segments of the testbed: Radio access, EPC, Transport, UE and Servers, respectively. Section 11 describes new features introduced in the testbed, like GPS emulation, model based testing and the robotic arm for Virtual reality applications. Section 12 provides the results obtained after running internally experiments to exercise the testbed and probe its ability to characterize the performance of mobile apps. Finally, Section 13 enumerates the features that the project will introduce in Release 3.

2 TRIANGLE testbed architecture

In this section we will provide a quick overview of the testbed workflow which will enable the reader to understand the architecture of the testbed and the role of each one of the components

2.1 TRIANGLE testbed workflow and architecture overview

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 where the users declare the features of their Apps (implementation statements), and any required additional information (such as the app user flow which contain 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.

To configure and execute a test the orchestrator (implemented in Python as shown in Figure 3), will make use of the information provided by the end user at the Portal. The information is retrieved by the Orchestrator using the REST API provided by the Portal and described in Appendix 1.

With this information the Orchestrator instantiate the Composer to compose a TAP test plan that includes the network scenario setup, the app installer, the app user flow and the measurements that have to be gathered to compute a given set of KPIs. The KPIs depends on the features declared by the App. The TAP test plan is built from existing TAP templates and TAP scenarios as shown in Figure 3.

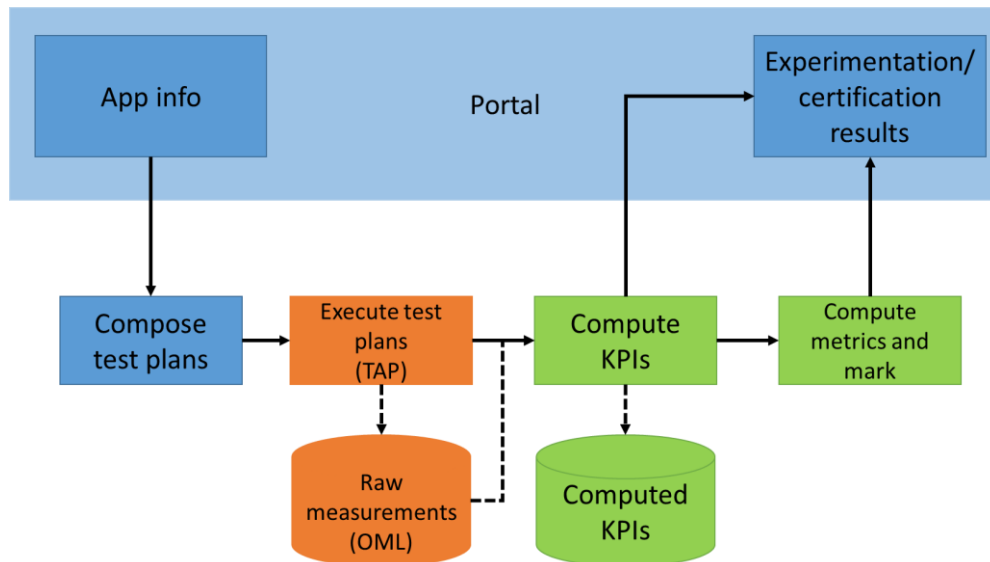


Figure 2 Testbed workflow

In the next step of the testbed workflow the Executor is instantiated to execute this TAP test plan with TAP. The TAP test plan configures all equipment needed to run the tests (such as the UXM), 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.

Finally the Orchestrator instantiates the ETL module (Extract, Transform and Load) to compute the KPIs from these measurements. The computed results will be stored in a separate database.

For certification, a set of metrics will be derived out of the computed KPIs. To rate the product using the TRIANGLE mark, the metrics will be compared against a set of reference values. These metrics will be stored in a separate database.

Finally, the results will be presented to the end user in the Portal.

This workflow translates into the components shown in Figure 3. For the sake of clarity in Figure 3 only shows control and management entities. Software and hardware components of the testbed are shown in Figure 8.

During the implementation the Orchestrator, the Compositor and the Executor has been implemented as part of the same component, called the Orcompositor. The Orcompositor 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 Orcompositor is described in more detail in Section 4.

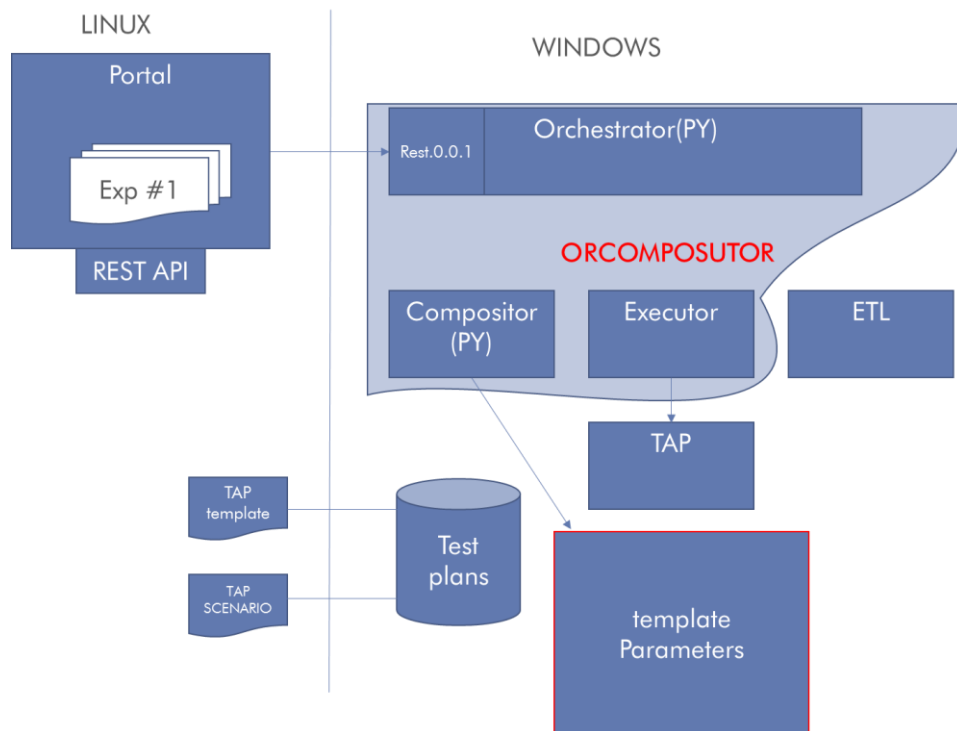


Figure 3 Control and management entities of the testbed

3 Interface and visualization (Portal)

Portal is the main entry point of 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 4). Each use case has associated a set of features as shown in Figure 5. These features will define what can be tested through experiments, or which tests specifications will be applicable when they opt for certification (certification will be available in Release 3).

ExoPlayer2 Demo v.2.0.0.34-0e111b

Home / Apps / ExoPlayer2 Demo / v.2.0.0.34-0e111b

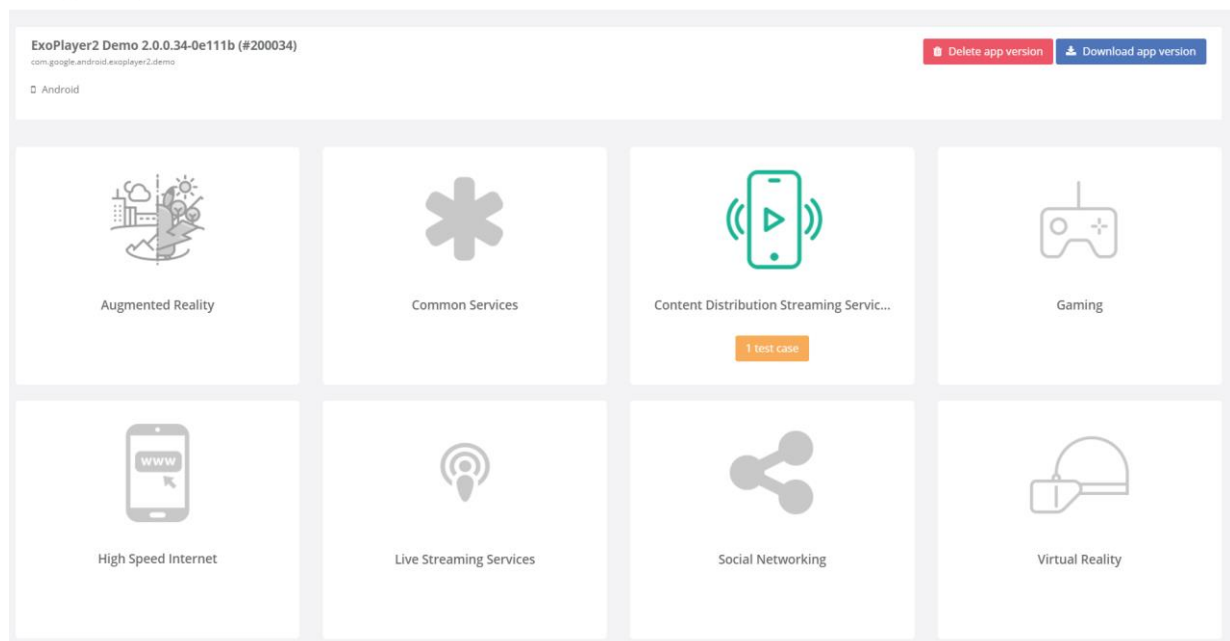


Figure 4 Uses cases

Users can then 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 his app to measure the KPIs associated to the features declared.

3.1 Information provided by users

App developers provides 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

Delete app version

Download app version

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 5 Features selection and measurements points

3.1.1 App info

The following is 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 will be 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.



3.1.2 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.

3.2 Backend REST API

The Portal now 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 4.1), 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 Appendix I.



4 Orchestration

Once all the required information has been entered into the Portal, the TRIANGLE testbed end user can proceed with an 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 Orcomposutor 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 will be the same, regardless of the body of the test plan. Thus, the Orcomposutor will reuse 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 will get the app user flow from the Portal, and will set 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 be then 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 directly.

4.1 Orcomposutor

Orcomposutor (ORchestrator-COMPOSer-execUTOR) 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 (which run on separate machines) by:

- Accepting test campaign execution requests from the Portal
- Composing the TAP test plans required to run a test campaign and its test cases
- Executing the TAP test plans
- Uploading 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 3.2), and with the OML database.

4.1.1 REST API

Orcomposutor has a REST API that exposes a single method.

Table 1 - Orcomposutor REST API

URL	Method	Description
<code>/run_campaign?id=<id></code>	GET	Handles requests to execute a campaign. Returns immediately. Backend will be updated with campaign progress and results.

This method returns immediately, informing the caller that the campaign was started successfully. If there is another test campaign being executed on the testbed at the moment, Orcomposutor will return an error code. The backend will be updated periodically with the progress of the test campaign execution, and any results that are produced after executing each test case.

4.1.2 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. Since a single test campaign can define more than one test case, it is possible that Orcomposutor must 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 to be executed. 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 these 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.

4.1.3 Results

Once a TAP test plan has finished 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

4.2 Test Automation platform (TAP)

4.2.1 New features in TAP)



The TAP automation software introduces three new major features into the TRIANGLE testbed Release 2:

- The compatibility of Test Plan Reference test step with External Parameter
- TAP server
- Move from .NET to .NET Core

4.2.2 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.

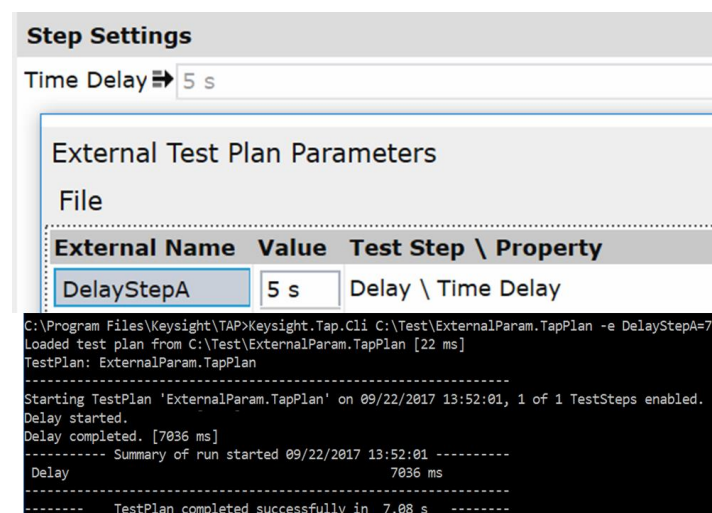


Figure 6 External Parameter setting & command line execution

On Figure 6 shows an example of a TAP test plan 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.

The new TAP release (TAP 2018 or simply 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.

4.2.3 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.

4.2.4 .NET Core

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

4.2.5 TAP test plan templates

TAP test plans to be executed on the TRIANGLE testbed will be created from generic templates, and programmatically parameterized by the Orcomposutor according to the domain, use case, test case and network scenario the user wants to test in.

The skeleton of a template consists of the following example steps:

- Instruments initialization & set-up according to domain, test case and network scenario, for instance (highlighted in green on Figure 7)
 - o UE initialization with the installation of the application under test
 - o Power analyser set up to measure device power consumption
 - o UXM configuration for a specific network scenario
- A repeat loop, which will run successive iterations of the same application flows, to gather multiple measurements to reach meaningful and converged test results. (In orange on Figure 7)

During this loop:

- o Parameters representing network and channel conditions will evolve dynamically, within a single network scenario, to represent the variability of connectivity conditions for the end user in the field (purple on Figure 7)
- o The application flow will be executed, delivered by the application developer (blue on Figure 7)

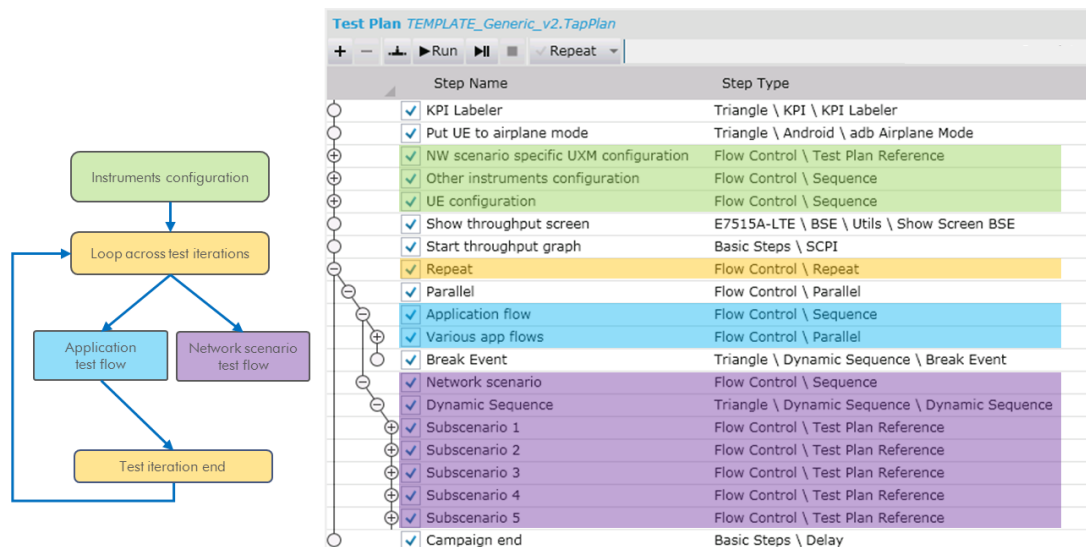


Figure 7 Test plan flow chart & TAP implementation



The different sections will take advantage of the “Test Plan Reference being an External Parameter” TAP feature introduced in 4.2.1, replacing the application test flow depending on the test case, substituting the network scenario depending on the emulated channel conditions the experimenter requests and so on.

4.2.6 TAP plugins

This section introduced the TAP plugins implemented to configure and control the different components of the TRIANGLE testbed. Figure 8 shown the hardware and software components of the testbed and the drivers associated to each one of them.

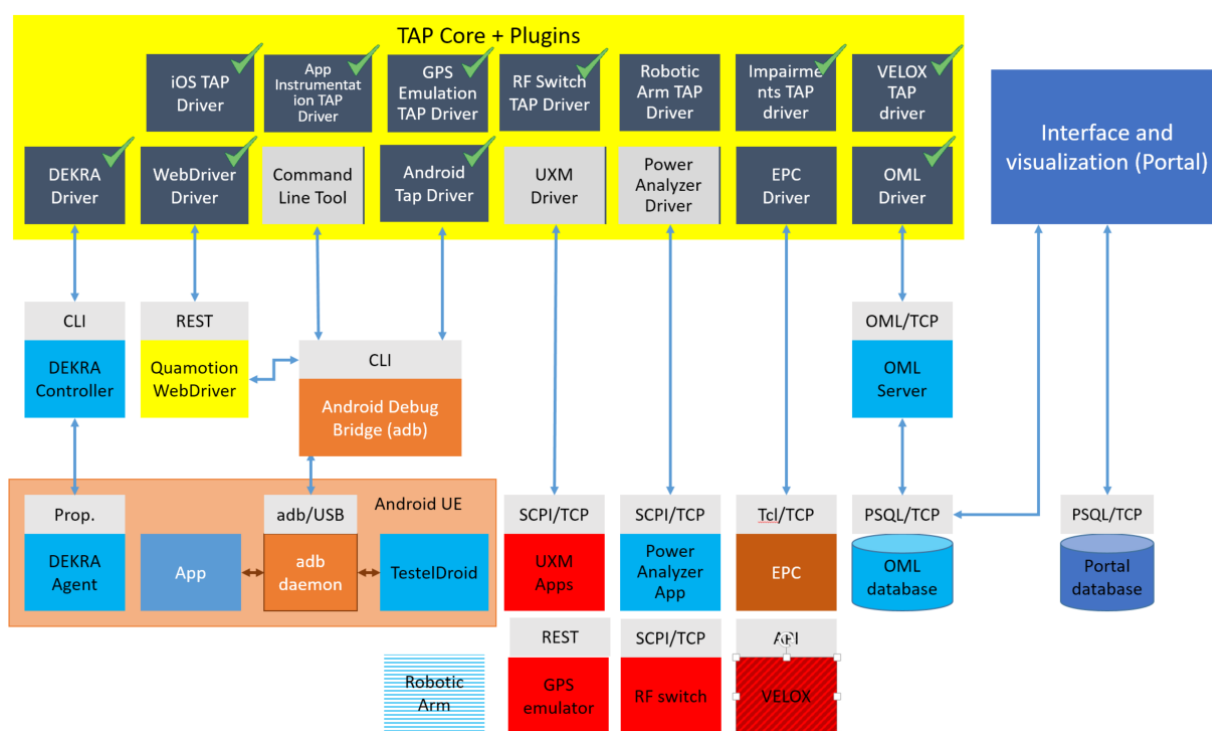


Figure 8 Hardware and software components of the testbed for Release 2

Table 2 provides a brief description of each one of the TAP plugins implemented. All the implementations details of these TAP plugins are provided in Appendix 2.

Table 2 – TAP plugins implemented in the Release 2

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.



<i>OML server</i>	This plugin allows TAP to send the results reported during an experiment to an OML server (which will store them in a database).
<i>Instrumentation library</i>	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.
<i>Android</i>	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.
<i>iOS</i>	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.
<i>RF switch</i>	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.
<i>GPS emulation</i>	This plugin provides the instruments and the steps to configure and control the GPS emulation feature introduced in Section 11.
<i>Dynamic Sequence Plugin</i>	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.
<i>Iteration-aware result listener plugin</i>	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.
<i>KPIs calculation plugin</i>	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.
<i>Plugins to reach TAP server</i>	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.

4.3 New features in the Quamotion WebDriver

The following new features are added to the Quamotion WebDriver in the second release of the TRIANGLE Testbed

- Application flow editor
- Full device automation
- Verification of user interface properties
- Remote control on Android and iOS
- Record application flows (Click, enter text, back)
- Hybrid and browser application support
- Support for iOS 11 and Android 8 versions

4.3.1 Application flow editor

The Quamotion WebDriver allows to create, record and edit an application flow for both Android as well as iOS applications (see Figure 9: Quamotion WebDriver script editor). There is no need to write code for basic scenarios.

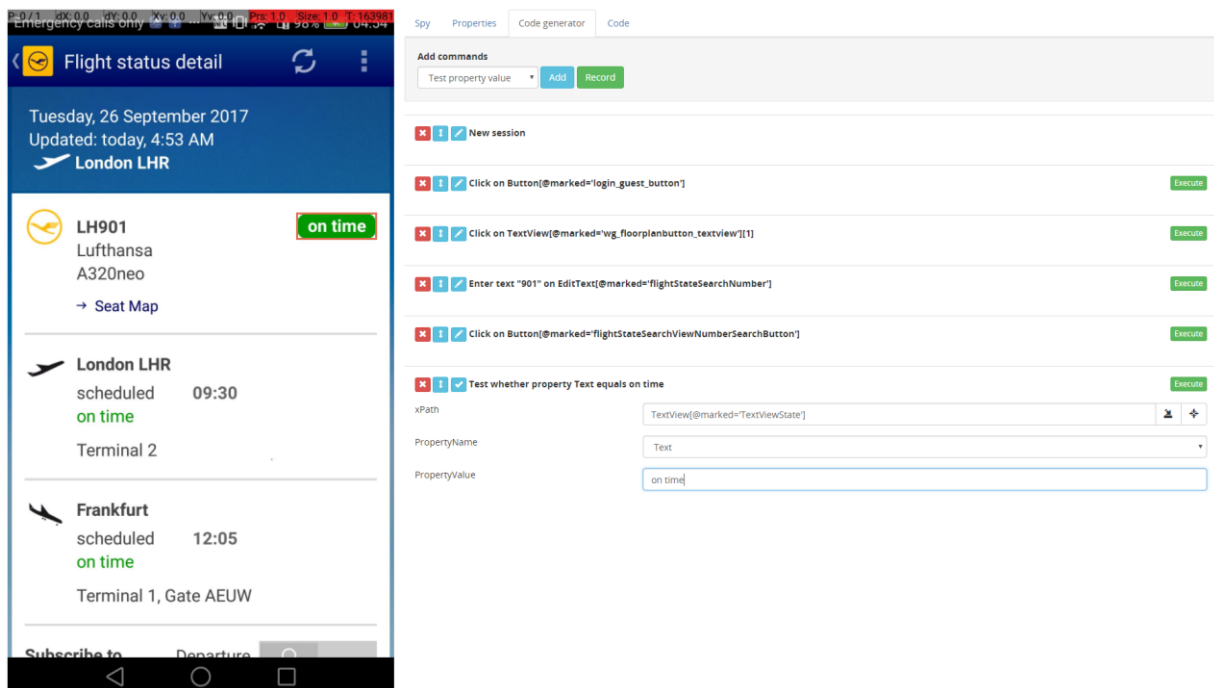


Figure 9: Quamotion WebDriver script editor



The script editor provides templates for the most common commands such as those described in Table 3 – Table 3.

Table 3 – Commands provided by the Quamotion Webdriver script editor

Command	Description	Properties
<i>New session</i>	Create a new session	
<i>Remove session</i>	Remove the current session	
<i>Click element</i>	Click on an element with the given XPath	xPath
<i>Send keys</i>	Send keys to the keystroke	Text
<i>Dismiss keyboard</i>	Dismiss the keyboard	
<i>Clear text</i>	Clear the text of the active text field	
<i>Enter text</i>	Enter text performs: 1. Click element 2. Clear text 3. Send keys 4. Dismiss keyboard	xPath Text
<i>Go back</i>	Press back button	
<i>Implicit wait</i>	Set the maximum allowed time to wait for an element	Time (milliseconds)
<i>Explicit wait</i>	Wait for the given time	Time (milliseconds)
<i>Test element</i>	Test whether an element with the given XPath exists	xPath
<i>Test property value</i>	Validate a property of an element	xPath Property name Expected property value
<i>Get property</i>	Get the value of a property	xPath Property name
<i>Set property</i>	Set the value of a property	xPath Property name Property value
<i>Get element</i>	Get the first element corresponding to the given XPath	xPath
<i>Get elements</i>	Get all elements corresponding to the given XPath	xPath



The application flow can be exported to a script language of choice or can be exported in a JSON format which is supported by the TRIANGLE portal.

4.3.2 Full device automation

The Quamotion WebDriver allows to automate parts of the user interface outside the sandbox of an application. This allows to automate e.g. system notifications, applications pulled from the app store (iOS), etc.

The Quamotion WebDriver API, clients and frontend can be used in the same way as for app automation.

4.3.3 Verification of user interface properties

Properties of user interface elements can be retrieved and verified. This gives tangible feedback on whether the application flow did succeed in a test campaign. The validation results can be queried in the dashboard or can be queried through the provided APIs.

4.3.4 Remote control on Android and iOS

The Quamotion WebDriver provides VNC functionality for Android and iOS devices. It is now possible to interact with a connected mobile device from the browser.

4.3.5 Record application flows (Click, enter text, back)

Application flows can be generated by interacting with a mobile device. Clicks, enter text and back actions are intercepted and generate the appropriate command in the Application flow editor.

The recorded script can be edited during and after recording.

4.3.6 Hybrid and browser application support

Quamotion improved the support for hybrid and browser applications. The most common gestures like click, enter text, swipe, scroll-to are now available for web.

4.3.7 Support for iOS 11 and Android 8 versions

The Quamotion WebDriver supports the latest versions of Android and iOS. Users can still choose their preferred operating system (Linux, Mac and Windows) to install and use the Quamotion WebDriver e.g. there is no need for a Mac computer to automate an iOS device.



5 Measurements and data collection

5.1 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) will perform 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 Orcompositor, 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.

5.2 Metrics and mark computation

For each campaign, a number of metrics need to be computed. The input for this process are the computed KPI values from the previous step. The computation of the metrics and marks is part of the ETL module which will be described in detail in D3.3.

5.3 Instrumentation library

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 extract measurements from inside their applications. The measurements performed through the Instrumentation Library will be 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. At the moment, 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.

5.3.1 Library contents

The instrumentation library will be used by app developers to provide measurement from within their own apps. These measurements are necessary to compute 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

5.3.2 Standard measurements

As part of the TRIANGLE project, a set of “standard” measurements have been defined. These measurements 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

5.3.3 Custom measurements

The library provides means to app developers to provide additional measurements, called “custom” measurements. These measurements will be parsed and stored alongside the rest of the measurements, but they will not be 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



6 RAN (Radio Access Network)

No updates.

7 EPC

7.1 EPC Architecture

The testbed can create automatically an EPC architecture to be used in the different experiments. The current architecture that is deployed with the EPC plugin is depicted in the following figure.

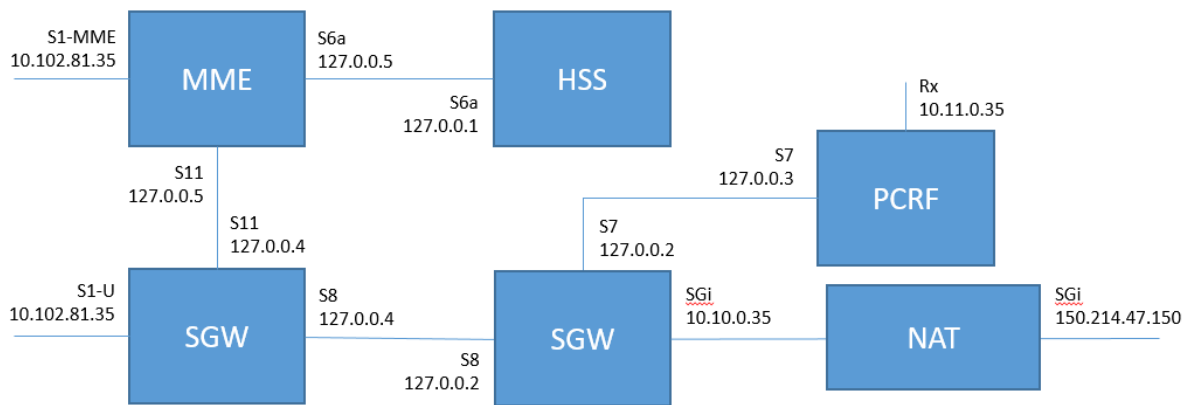


Figure 10 EPC architecture

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

Table 4: 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 interfaces or component on the network and in future releases of the testbed the use of more reference deployments will be explored to support other scenarios.

7.2 EPC Triggered Procedures

Besides automatically deploying the EPC the plugin is also able to trigger certain procedures on the network. The ones that are currently supported are the following:

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

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

8 Transport

TNO has deployed a MANO in the TRIANGLE testbed. After an initial study phase, and in agreement with the TRIANGLE partners, the MANO provided by Openstack, Tacker, was selected. Tacker, along with the full NFV stack provided by Openstack, was installed on one of the TRIANGLE local servers at UMA (Figure 11).

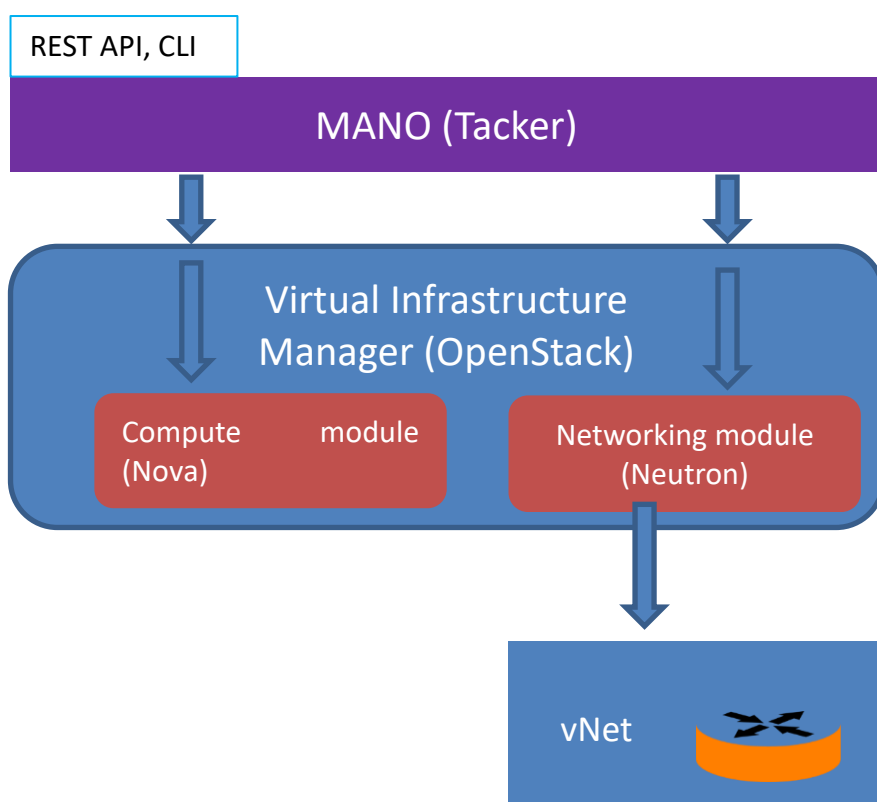


Figure 11: MANO deployment architecture

For this installation, the FUEL installer from OPNFV was selected since it was the only OPNFV installer which delivered a usable environment.

Table 5: OPNFV FUEL installer details

Virtual edition installs flawlessly?	OS	Tacker installed?	Openstack	OpenDaylight	OVS-DPDK	Reboot survive
No (manual patching needed, manual plugins patching and compilation needed)	Ubuntu 16.04	Yes, version (0.0.1)	Newton	L3 but plug-in crashes	Yes (on hardware)	Yes

Using MANO, we managed to on-board a VNF and deploy an instance whose state was to be monitored by MANO. Should the instance stops responding to ping messages, MANO takes an



Document: ICT-688712-TRIANGLE/D3.2

Date: 20/07/2018

Dissemination: PU

Status: Final

Version: 1.1

action of respawning it. This functionality is planned to be used for all deployed services after full integration of our extensions.

9 UE (User Equipment and accessories)

9.1 Supported UEs

The user equipment has to be physically connected to the testbed as shown in Figure 12. First, the UXM Wireless Test Set integrates channel emulation and digital generation of impairments such as AWGN, which is a critical feature to achieve high accuracy when setting SNR conditions. In particular, standard multipath fading profiles defined by 3GPP are supported to emulate reference propagation conditions. In order to preserve the radio conditions configured at the UXM Wireless Test Set the radio connection is conducted through cables and the devices are enclosed in a shielding box.

For testing purposes most UEs typically contain small antenna connectors which are normally hidden from the user, these connectors are used for the cable connection to the UXM. In order to integrate a new device in the testbed the type of antenna connector has to be identified and its location in the main board discovered. Usually, manufactures don't offer this information. After successive and thorough searches on the Internet the type of connectors used by Samsung have been identified and the antennas located as shown in Figure 14. The device presents a total of 4 antennas: 1 main and 1 diversity for low frequencies (LF) and 1 main and 1 diversity for high frequencies (HF).

Second, to analyze properly power consumption, the device must be powered directly by the N6705B power analyzer. The battery has to be removed and a physical connector has to be built based on the type of the battery connector available in the device.

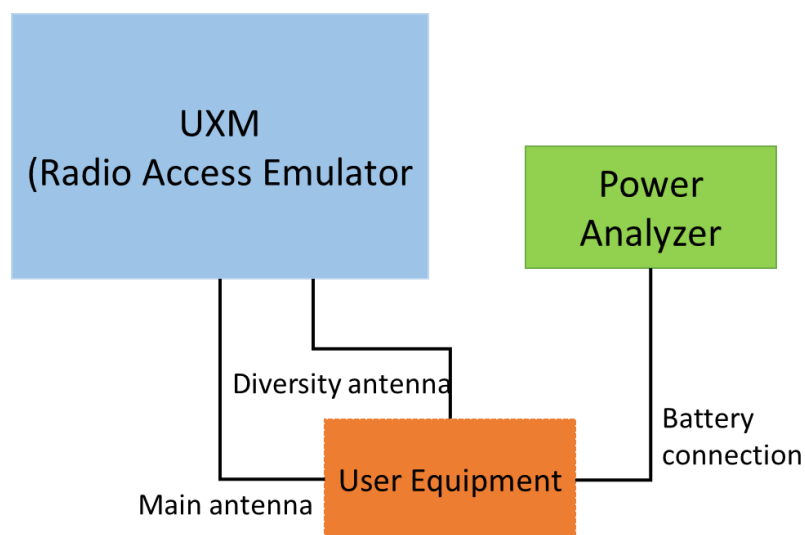


Figure 12 User equipment connectorization



Figure 13 iPhone 7 plus

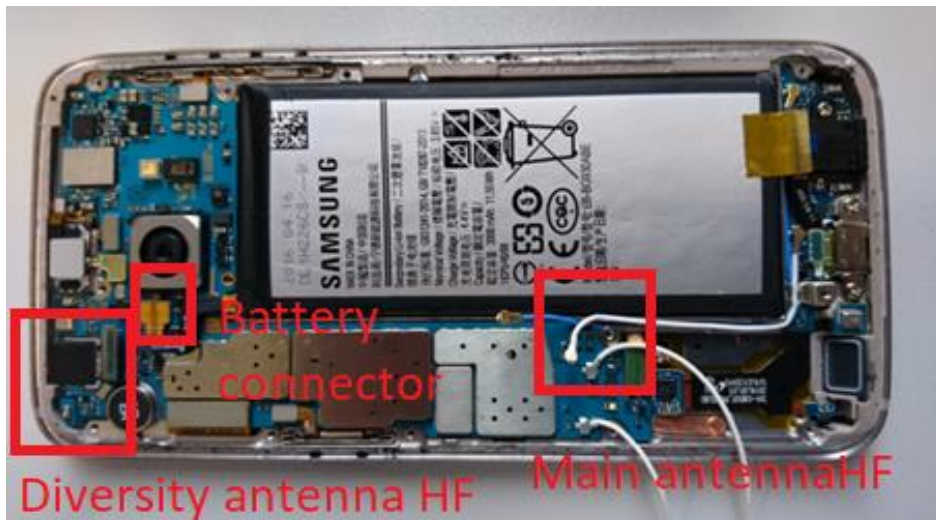


Figure 14 Samsung Galaxy S7

Table 6 shows the status of the commercial devices connected to the TRIANGLE testbed. Samsung Galaxy S4 and Samsung Galaxy S7 are fully integrated (radio and battery are connectorized). We are working on the connectorization of iPhone 7 Plus, the type of the antenna connector has not been identified yet. A potential solution for those devices, which cannot be connectorized via cables, would be to use a contact antenna like the one shown in Figure 15. We are also working in the addition of latest devices models.



Figure 15 Contact antenna

Table 6 Current status of devices integrated into the testbed

Device	Main Ant 1 (HF)	Main Ant 2 (LF)	Diversity Ant 1 (HF)	Diversity Ant 2 (LF)	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	No
Samsung Galaxy S7	Yes	No	Yes	No	Yes
iPhone 7 Plus	No	No	No	No	No

10 Local applications and servers

10.1 DANE (DASH-Aware Network Element)

TNO's extension adds a DANE (DASH-Aware Network Element) in the TRIANGLE testbed. The DANE is a recently standardised network element [2][3] whose aim is to provide network assistance to video streaming clients supporting the MPEG-DASH (or the 3GP-DASH) protocol [3],[4]. Specifically, the DANE supports the SAND protocol [2][3], 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 [3]. In particular, the DANE awaits 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 16 pictures the DANE within the context of the TRIANGLE testbed, including a streaming server offering DASH content and a VR streaming application consuming it.

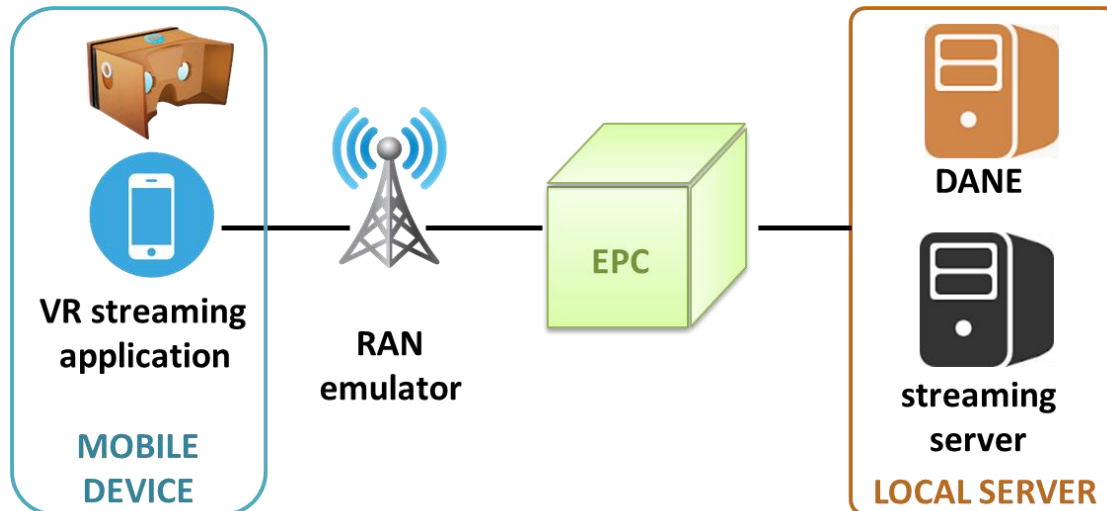


Figure 16 SAND architecture within the TRIANGLE testbed, including DANE, DASH streaming server and DASH VR app

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 17. In fact, for each test scenario, the average bandwidth that the UE is expected to achieve has been previously computed via measurements with iperf.

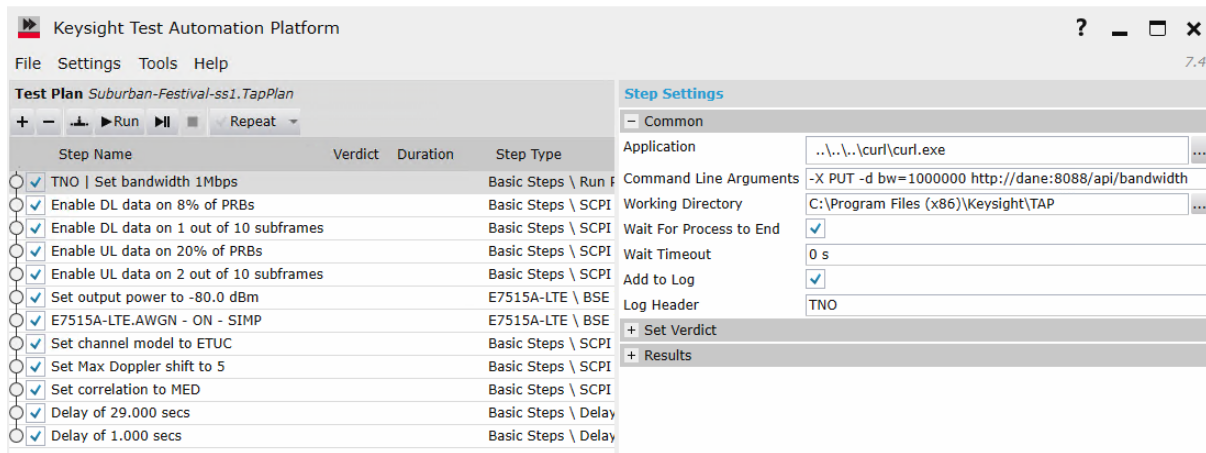


Figure 17 Setting the DANE's bandwidth within a TAP's test scenario



11 Extensions and new features

11.1 GPS emulation

The importance of including an option to generate GPS signals in the 5G testing benchmark framework, is an incipient need in many aspects. 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 18 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.

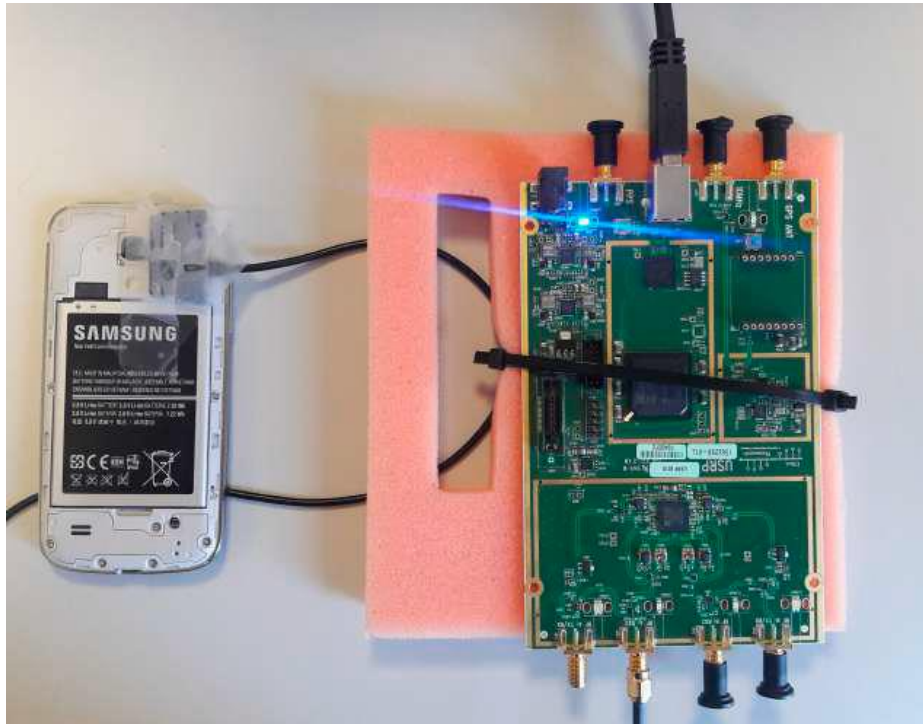


Figure 18 Connecting USRP output to the UE GPS antenna

11.1.1.1 Control of the USRP

11.1.1.1.1 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 others 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.

11.1.1.1.2 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

11.1.2.1 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 for 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
```



11.1.2.1 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.

11.1.2.2 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 19 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>#s_ylw-pushpin</styleUrl>
    <LineString>
      <tessellate>1</tessellate>
      <coordinates>
        -74.04454330351703,40.68992250780924,0 -74.04448049752409,40.68998981829354,0 -74.04365999683444,40.68972312818734,0
        -74.04355748141882,40.68959806666543,0 -74.04355415654128,40.68927640089319,0 -74.04369295015395,40.68887115009155,0 -74.04397768182285,40.68864807428893,0
        -74.04437460585586,40.68869213605716,0 -74.04470506468249,40.68864118439994,0 -74.04539210140229,40.68901631409842,0 -74.04532338224951,40.68924669541931,0
        -74.0451776264553,40.68951278154722,0 -74.04536175416838,40.68980927950763,0 -74.0455346266455,40.69027693804819,0 -74.04582298716409,40.69037221398904,0
        -74.04593396670382,40.69059203985196,0 -74.04600821478076,40.69066103617261,0 -74.0459808084006,40.69073071456939,0 -74.04588926482131,40.69074831902451,0
        -74.04579207446482,40.69079563590495,0 -74.04564704995599,40.69079563394025,0 -74.04547531046278,40.69074243846327,0 -74.04537162718945,40.69054747868183,0
        -74.04539422568363,40.69042873043386,0 -74.04487672296301,40.6898476452449,0
      </coordinates>
    </LineString>
  </Placemark>
</Document>
</kml>
```

Figure 19 Google Maps route and XML file

11.2 VR Applications Testing

Virtual Reality is a 5G use case identified in the Work Package 2. The Release 1 of TRIANGLE testbed did not have the measurement capability to test this type of applications. 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.

This testing solution has not yet been integrated in the Release 2 of the TRIANGLE testbed and has been validated as standalone module. The integration of the module in the TRIANGLE testbed has been planned for Release 3.



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 7 is an extract from the Apps User Experience Test Specification (AUE).

Table 7 – VR Application User Experience Key Performance Indicators

KPI	Definition
<i>Time to load the virtual world</i>	Time elapsed from selecting a scenario (world, experience, etc.) to loading the 3D visual context
<i>Immersion Cut-off</i>	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 20 shows the high level architecture of the VR testing solution implemented in TRIANGLE.

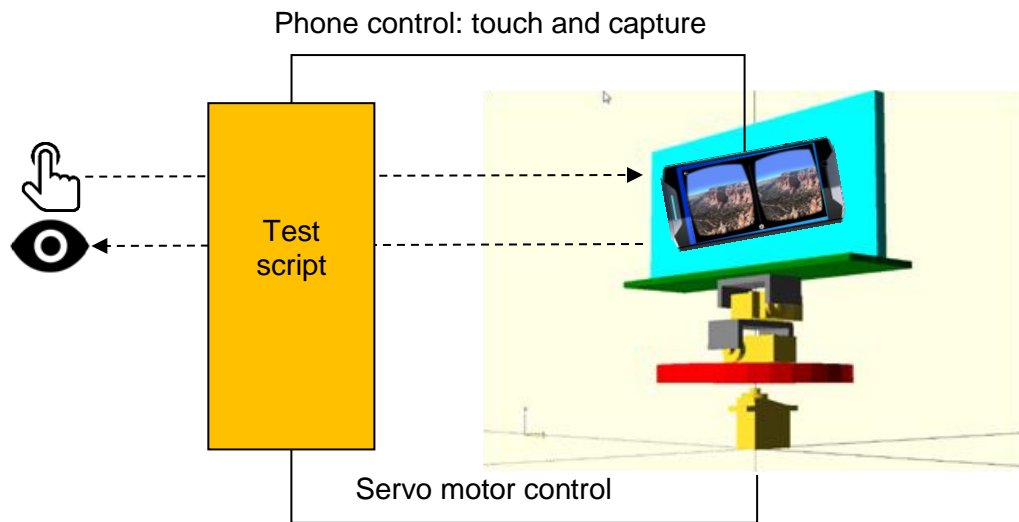


Figure 20 VR test module architecture

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.

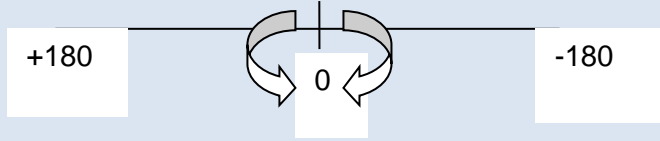
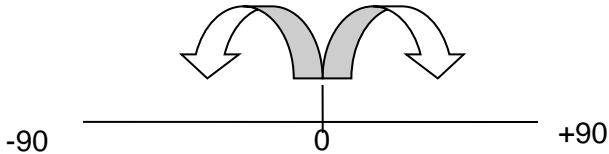
11.2.3 Robotic Arm

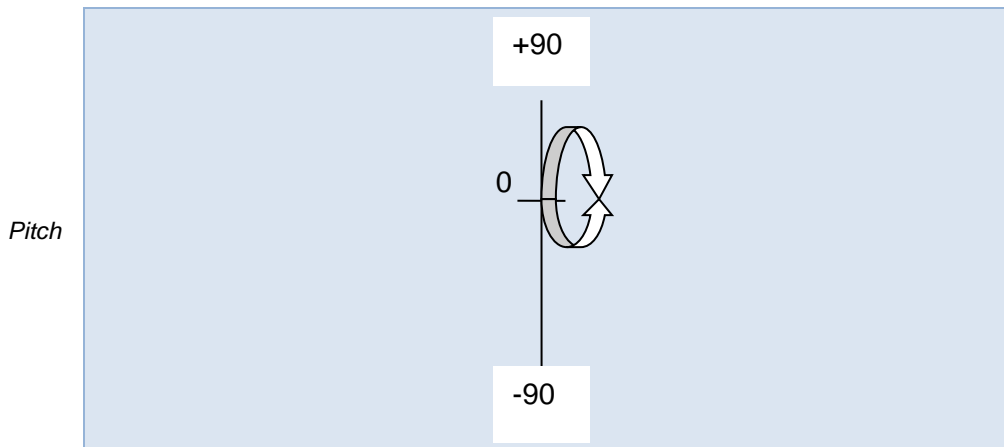
A robotic arm has been designed to support a device phone on top. This enables that the test script can force the phone to move in the three-spatial axis thus discovering the surrounding virtual world.

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 9 shows the position range and reference system used by the servos:

Table 8 – Robotic arm position range and reference

Axis	Range
Yaw	
Roll	



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.

Command 1: 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, ASCII code 27

Command 2: 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.

11.2.4.1 Auxiliary Commands

We have implemented some “utile” functions in order to facilitate further implementation of test scripts.

Position conversion degrees- μ s

We have implemented a function to convert the magnitude of the position from pulse width (μ s) into degrees. This way is much more intuitive for programming purposes.

The function to convert from μ s to degrees is:

$$\text{Angle (degrees)} = (\text{Central position } (\mu\text{s}) - \text{Angle } (\mu\text{s})) / \text{Time to spin } 180^\circ \text{ in } \mu\text{s}$$

The function to convert from degree to us is:

$$\text{Angle } (\mu\text{s}) = \text{Central position } (\mu\text{s}) - (\text{Time to spin } 180^\circ (\mu\text{s}) * \text{Angle } (\mu\text{s})) / 180$$



Table 9 shows the reference values needed in the conversion functions for the servos used in the TRIANGLE testbed:

Table 9 – Robotic arm reference values

Axis	Central Position (μs)	Time to spin 180° (μs)
<i>Yaw</i>	1450	1700
<i>Roll</i>	1550	1650
<i>Pitch</i>	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 servo 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 21 shows how openCV refers to the axis depending on the image orientation.

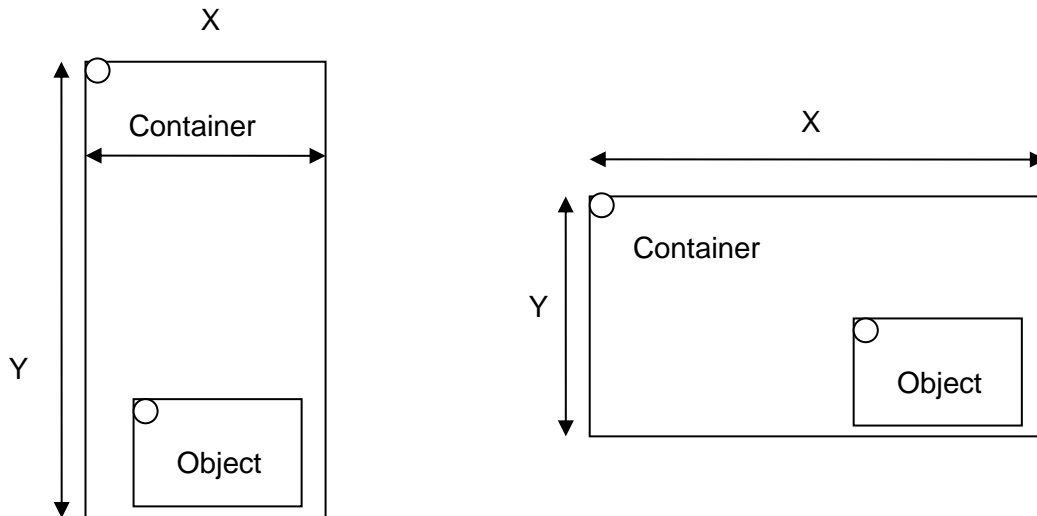


Figure 21 OpenCV system reference

From the comparison result the library provides the coordinates of the maximum matching point (circle inside the object in Figure 21). 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 11.2.6.4). The buffer contains some header bytes, which provides metadata about the captured image (version, size, orientation, etc.). Based on 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”.

11.2.6.1 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 11.2.6.2) 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.

11.2.6.2 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.

11.2.6.3 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.

11.2.6.4 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 22 shows the validation scenario.

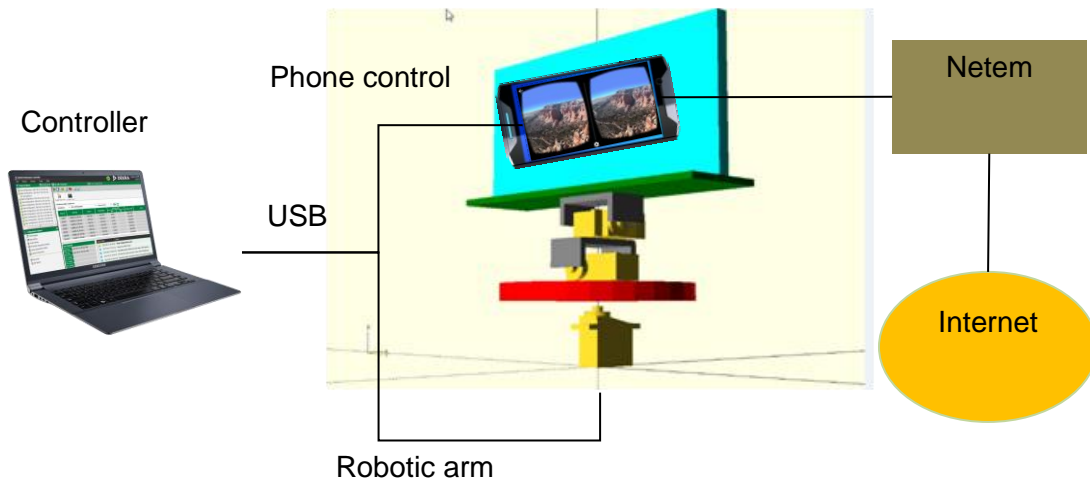


Figure 22 VR test solution validation

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 impairments on that link.

Figure 28 shows a photo taken in the lab during the validation of the module:

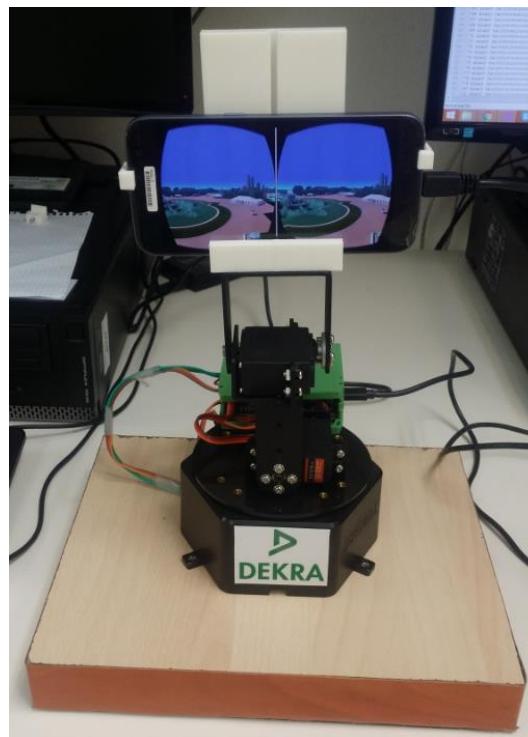


Figure 23 VR validation



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 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 10 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 10 – 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 10 compiles the results of the tests for different network bandwidths.

Table 11 – 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

Model-based testing techniques use a model of the system under test for automatically generating test cases with an adequate coverage. We have adopted a model-based approach to construct app user flows (user interactions) on which we can analyse properties. In this project, the purpose is to evaluate the app features using a set of KPIs.

In [6] [7], [8], we have constructed the tool MVE 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 [10] has supported test cases generation and runtime verification, in particular SPIN model checker [9]. Model checking is a formal technique that exhaustively explore 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, we have extended the previous approach in two different ways. First, we explicitly separate the construction of the app model and the specific requirements 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.

11.3.1 App model

The app under test is modelled using nested state machines [7]. 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 24 shows an example of the modelling language for the Universal Music Player sample app from the Android SDK, whose GUI can be seen in Figure 25.

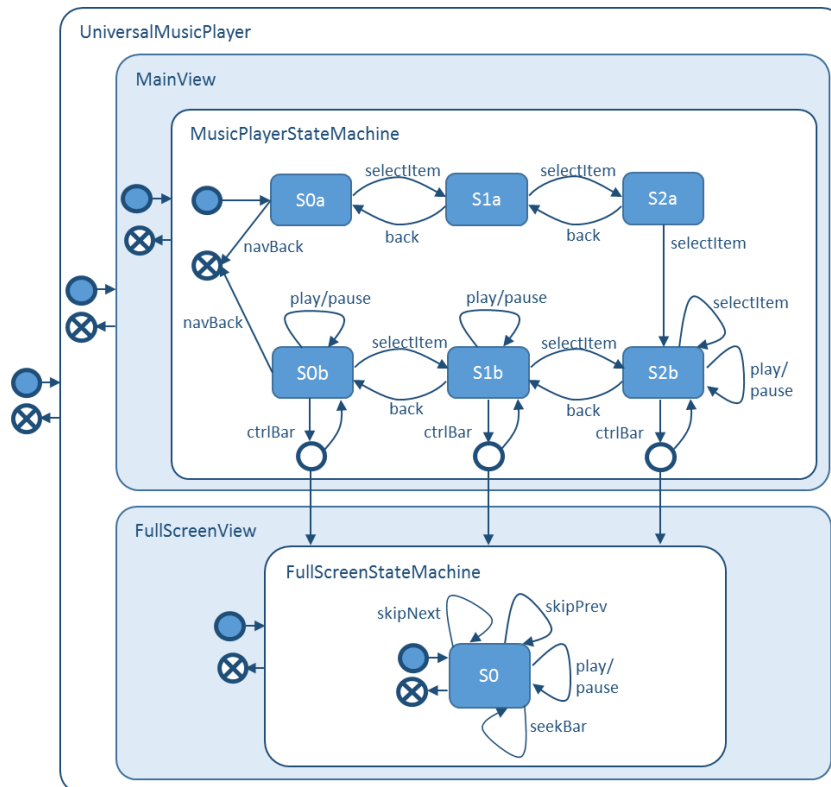


Figure 24 Universal Music Player model

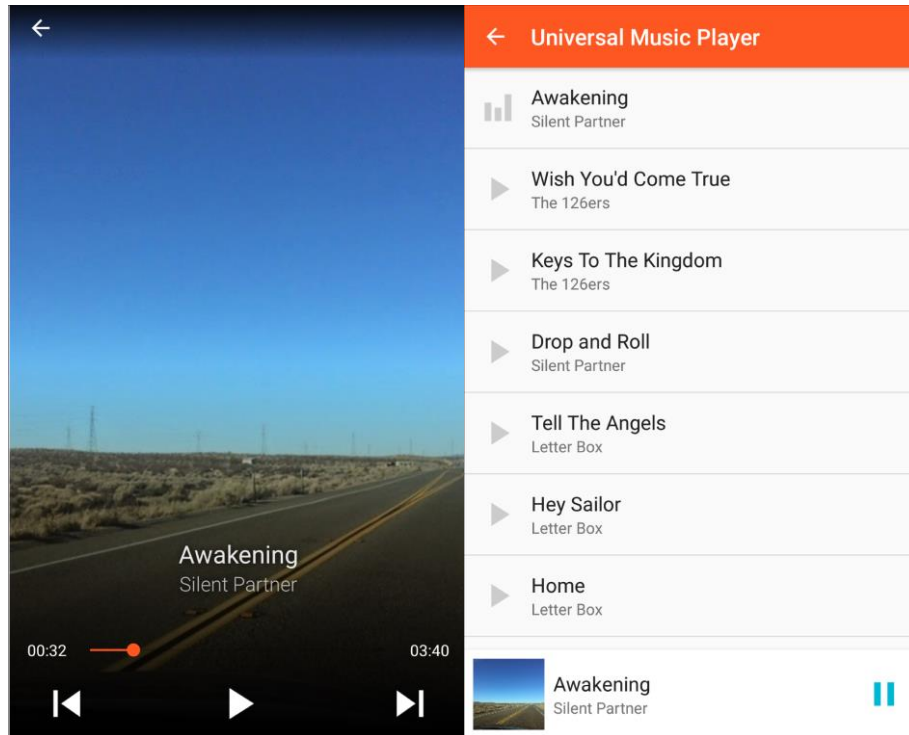


Figure 25 Universal Music Player GUI

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 24 shows the two activities (`MainView` and `FullscreenView`) and the possible user events (Play/Pause, back, etc.) that can happen during the app execution.

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. The app model is provided as an XML file, which is translated into a PROMELA specification [9]. We then used the Spin [9] 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 were then converted into Java programs that performed the flows on an actual Android device, using the UiAutomator API.

Figure 26 shows part of the PROMELA specification generated automatically from the app model in Figure 24. 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`.



```
1 DeviceType devices [ DEVICES ];
2 # define curBackstack devices [ device ]. Backstack
3 # define curState curBackstack . states [ curBackstack . index ]
4
5 proctype device_4107a7166c03af9b (int device ) {
6   do
7     :: curBackstack . index > -1 && curState == St_MainView_MusicPlayerSM_S2 ->
8     // Event : selectItem
9     transition (device , VIEW_MainView , 6);
10    curState = St_MainView_MusicPlayerSM_S2b
11    :: curBackstack . index > -1 && curState == St_MainView_MusicPlayerSM_S2b ->
12    // Event : clickBack
13    transition (device , VIEW_MainView , 7);
14    curState = St_MusicPlayer_MainView_MusicPlayerSM_S1b
15    :: curBackstack . index > -1 && curState == St_FullScreenView_FullScreenSM_init ->
16    pushToBackstack (device , St_FullScreenView_FullScreenPlayerSM_init );
17    transition (device , VIEW_FullScreenView , 0);
18    curState = St_FullScreenPlayerView_FullScreenSM_S0
19  // ...
20  od
21 }
```

Figure 26 Extract of PROMELA specification for test case generation

11.3.2 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 [9]. 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.

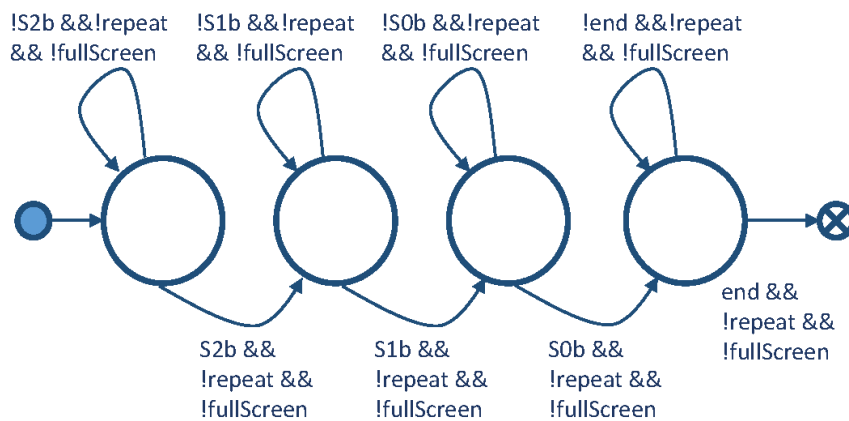


Figure 27 Pruning never claim as automaton

Figure 27 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 26. 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 27, the looping transitions exclude the paths that have repeated transitions or activate the full screen activity.



11.3.3 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 27). 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 24 Universal Music Player model. Table 12 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 12 App user flow generation - Experiments

Max. len.	Never	Flows	Time	Memory	States
10	<i>pr.</i>	18/20	< 1s	9.5MB	1,059
10	<i>no-pr.</i>	18/85	55s	11.1MB	20,787
10	-	-/85	< 1s	10.9MB	20,787
20	<i>pr.</i>	20/22	< 1s	9.6MB	1,645
20	<i>no-pr.</i>	-/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.

11.4 Traffic Impairments

The testbed now support the configuration of traffic impairments on different interfaces of the system. These impairments are provided using netem and a Ssh to provide remote access to the machines involved. The impairments exposed in each of the interfaces are the following:



- Latency, which is defined by the mean, the variance and the correlation. Optionally a probabilistic distribution can also be specified. Currently the testbed supports normal, pareto and pareto normal distributions but it could be expanded with new ones in the future.
- Link errors, defined by their mean percentage and variance.
- Mean duplication of packets.
- Reordering of packets, specified by their link and variance.

Currently the system is designed to allow the previous impairments in the S1 interface and in the SGi but more interfaces can be enabled on demand very easily. The implementation details of the actual SCPI server are provided in deliverable D4.1 and in the internal deliverable “Impairments Plugin Server”.

11.5 Remote PCAP Capabilities

A remote PCAP server has been deployed in the EPC, enabling the capture of traffic in any of the interfaces. Currently the access to that server is done only from the research profile using standard tools such as tshark or wireshark.

In future versions the traffic sniffing will be enabled on demand per each of the interfaces (e.g.: enable capture on S1-MME interface, etc.).



12 Internal test experiment

This section introduces initial test cases designed to validate the investigated KPIs and the automation capabilities of the TRIANGLE test infrastructure by verifying the performance of well-known applications such as YouTube and ExoPlayer. The latter is an open source media player, originally developed by Google with support for DASH [1] adaptive playbacks. The tests will allow to have an initial baseline of the performance of these applications, and to identify new aspects of interest. At a later phase we will use these results as reference for benchmarking evaluation. As a side note, the use of ExoPlayer as test App, given its open source nature, allowed the inclusion of an experimental measurement extraction library in order to compute the App's KPIs.

With respect to the previously defined certification scheme, the developed test cases refer to Quality of Experience, Device Resource Usage, Network Resources Consumption, and Energy Consumption. The particular use cases that the Apps currently compared is CS (Content Distribution Streaming) while the experiments have been run in a UR-PE (Urban Pedestrian) network scenario.

12.1 Testbed setup

The TRIANGLE testbed uses advanced lab instruments to evaluate applications and devices, when accessing services over emulated mobile networks, in realistic but controlled experiments.

To orchestrate the different components, the testbed uses the Test Automation Platform (TAP) from Keysight [12], to coordinate and run the test cases. Each testbed component is controlled through a TAP driver, which serves as bridge between the TAP engine and the actual component interface. Some of these drivers are being created as part of the testbed development. Radio access emulation plays a key role in the TRIANGLE testbed. In the first release of the testbed, the Radio Access Network (RAN) part will be provided by a UXM Wireless Test Set from Keysight [13], a mobile network emulator that provides state of the art test features. From the mobile devices and applications under test point of view, the UXM is perceived as a real network, even replicating complex RF propagation conditions when required, as it incorporates a channel emulator.

The testbed includes a set of reference devices for app testing which have to be physically connected to the testbed. In order to preserve the control over the radio channel conditions emulated by the UXM Wireless Test Set, the radio frequency connection must be conducted through calibrated cabling. Additionally, to accurately analyse the power consumption during the experiments, the devices must be powered directly by the Keysight's N6705B power analyser [14]. In the experiments, the DEKRA performance tool is also used to monitor RAM and CPU usage, as well as to derive QoE measurements for the YouTube sessions.

12.2 Test Configuration

In these initial experiments, we have selected a test configuration reproducing a crowded environment, where the users would only get access to a fraction of the available time frequency resources in the cells, as it could happen in a peak hour in the centre of a big city such as London or Tokyo. The mobile device is connected to a 4G-LTE Rel'12 cell transmitting at Band 3 (1843,5 MHz). Impaired propagation are emulated by setting the EPA5 fading conditions that cause fluctuations in the cell wanted signal power around -70 dBm in average, and an interfering cell which is configured to operate at the same frequency 20 dB below the average but with independently fading conditions. The effect of more distant cells is modelled with a white Gaussian noise interferer which is embedded in the UXM. The cell is configured for a Multiple

Input Multiple Output (MIMO) multi antenna transmission on Transmission Mode TM3. The selected Modulation and Coding Scheme (MCS) is set according to what is periodically reported by the UE via Channel Quality Indication (CQI) reporting messages.

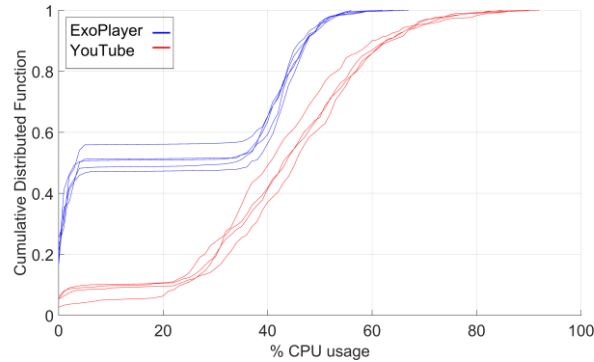


Figure 28 - CDF of the CPU usage [%] for YouTube API and ExoPlayer

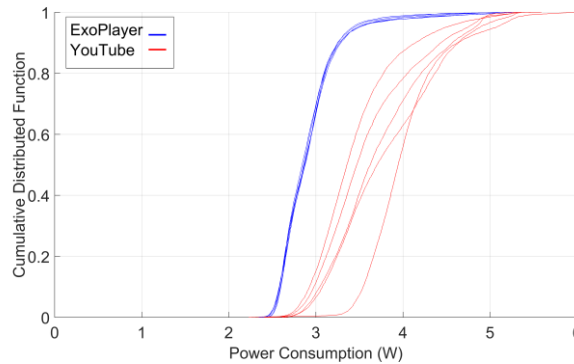


Figure 29 - CDF of the consumed Power [W] for YouTube and ExoPlayer

The measurements have been performed on ExoPlayer and the YouTube API while reproducing the video Big Buck Bunny [15], known to be a commonly used video reference. The video was streamed in two different ways, the native YouTube multi-codec and the DASH standard. Both streams were played from YouTube server, in order to make a fair comparison since the traffic would presumably transverse internet same way.

12.3 Initial Measurements

The measurements cover CPU and RAM usage percentage, video resolution in terms of pixel density, power in Watts, and received amount of data in kB.

Figure 28 depicts the Cumulative Distribution Function (CDF) of the CPU usage of ExoPlayer and YouTube. In both cases, CPU use has been monitored at 1 s intervals. For ExoPlayer, it can be observed that in average 50% of the time the application is almost not using any CPU resources. Interestingly, most of the remaining active time (40%, from a cumulative 55 to 95) involves a usage of 40 to 50% of the CPU capacity. The consumption is though reaching a maximum of roughly 60%.

During the YouTube reproductions, the negligible CPU usage extends only for 10% of the total time, jumping then to 25% and increasing linearly up to a max of 60% CPU for the remaining 10% of the time. The upper limit is higher than ExoPlayer extending to a roughly 80%. We can conclude that YouTube CPU consumption is 30% higher during the low activity half of the time,

and 10% higher during high activity phases. ExoPlayer presents a slightly better performance in terms of CPU. RAM usage has also been analysed, but there are only small differences between the two players, being the RAM usage of ExoPlayer lower than YouTube by 2-5%.

In terms of resolution, a relevant QoE parameter, it can be noted that the videos downloaded with ExoPlayer have a resolution of 360p (standard definition) while in YouTube the resolution is 480p (DVD quality). From these preliminary results, it is possible to deduce that ExoPlayer has generally better performance in the domain of Device Resource Consumption, while paying the price in terms of QoE as the resolution is higher in YouTube.

Figure 29 compares the Power consumption of the two Apps. As the video resolution is higher in the YouTube sessions, it seems reasonable to expect higher consumption. However, the 50 to 100% additional power increase measured cannot be justified by the sole difference in video resolution. It is then possible to deduce that YouTube is a more power-hungry App compared to ExoPlayer.

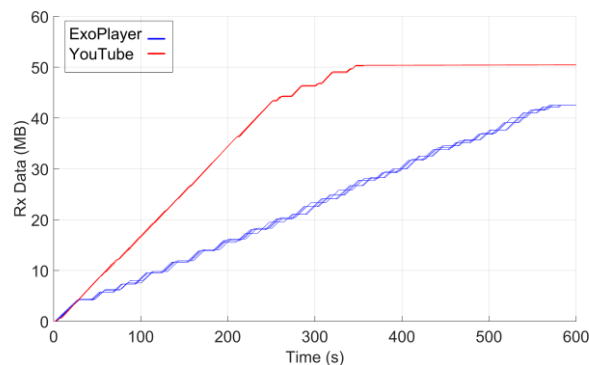


Figure 30 - Cumulative data received by YouTube and ExoPlayer

Figure 30 shows the cumulative amount of data received by the streaming clients. In the sessions analysed the YouTube client shows a more aggressive buffering strategy, where more than 90% of the video has been received at half of the playback time. ExoPlayer buffering grows nearly linearly during the full video playback in steps of approximately 5% of the reproduced video, after an initial buffering of 10% of the content.

In these initial measurements, given the freedom the applications have when adapting the codec that they use, we observe also significant differences in the monitored parameters. The different buffering strategies may also lead to advantages or drawbacks depending on the user behaviour and network conditions. On one hand, if the user decides to interrupt the reproduction, the application would waste part of the buffered data and its associated energy drain. On the other hand, if the network conditions cause temporary outages, e.g., when crossing a tunnel, the larger amount of buffered data will avoid playback interruptions thus increasing the user experience.



13 TRIANGLE testbed Release 3 specifications

Release 3 is expected to be available by 31st of March 2018. The expected new features for this release are grouped as follows:

- **User point of view:** TRIANGLE certification mark (QoE included), extended mobile device coverage.
- **Testbed access:**
 - Portal: usability improvements, results displayed in the portal & extended downloadable results.
- **Capabilities:** iOS extended UE measurement support. Extended support for VR and AR applications, preliminary support for gaming applications. Extended GPS support. Expand reference app. Support new app versions (iOS native and Web/Hybrid), Screenshot feed (performance framerate), Automatic generation of app models from apps without user intervention, provide testbed booking functionality,

These new features translate into the following updates for each one of the elements of the tested architecture.

13.1 Interface and visualization (Portal)

Adding support for the new features:

- Extended downloadable results
- Show TRIANGLE Mark, spider diagram(s)
- Support the execution of certification campaigns
- Add Portal support to booking
 - Add support in the Portal to book the Testbed, view current or future bookings.
- Add Portal support to create the routes for GPS emulation

13.2 Orchestration

The Portal is the user interface for the Testbed and the TRIANGLE test cases. TAP is the engine that runs the test cases. However, there is much more in order to orchestrate test campaigns and collect their results. Orchestration system is continuously evolving.

13.2.1 Webdriver tool

- Support new version:
 - iOS native
 - Web/hybrid
- Record
 - Record xy coordinates
- Remote control (VNC)
 - iOS
- Screenshot feed (performance framerate)



13.3 Measurement and data collection

A framework to compute KPIs and metric is being implemented. The framework is backed by a database where all the intermediate and final results are stored. This database should include information about the test campaign execution to which it belongs, KPIs, MOS, metrics by scenario and domain, and the aggregated metrics.

13.3.1 Certification and TRIANGLE mark

For certification, a set of metrics will be derived out of the computed KPIs. To rate the product using the TRIANGLE mark, the metrics will be compared against a set of reference values.

13.3.2 Instrumentation library for iOS

Create a new instrumentation library for iOS, using the same principles as the one for Android.

13.4 User equipment and accessories

New devices will be integrated into the testbed

13.5 Extensions and new features

Automatic generation of app models from apps without user intervention. Fully integration of the robotic arm.



14 References

- [1] libimobiledevice. "libimobiledevice A cross-platform software protocol library and tools to communicate with iOS® devices natively", online resource <http://www.libimobiledevice.org/>
- [2] "Information Technology — Dynamic adaptive streaming over HTTP (DASH) — Part 5: Server and network assisted DASH (SAND)", ISO/IEC 23009-5:2017
- [3] "Server and network assisted DASH for 3GPP Multimedia Services", ETSI 3GPP SA-170732, June 2017
- [4] "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
- [5] "Transparent end-to-end packet-switched streaming service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH)", ETSI 3GPP TS 26.247
- [6] 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.
- [7] 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.
- [8] 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.
- [9] G. Holzmann. The SPIN Model Checker : Primer and Reference Manual. Addison-Wesley Professional, Sept. 2003
- [10] E. Clarke, O. Grumberg, and D. Peled. Model checking. Mit Press, 1999.
- [11] Ricardo Marco Alaez, Jose M. Alcaraz Calero, Qi Wang, Fatna Belqasmi, May El-Barachi, Mohamad Badra, and Omar Alfandi, "Open Source Based Testbed for Multi-Operator 4G/5G Infrastructures Sharing in Virtual Environments", Wireless Communications and Mobile Computing, 2017
- [12] Keysight Technologies, "KS8400A Test Automation Platform 2017 Developer's System Software – Technical Overview", online resource, <http://literature.cdn.keysight.com/litweb/pdf/5992-1909EN.pdf?id=2796881>
- [13] Keysight Technologies, "E7515A UXM Wireless Test Set Getting Started Guide", online resource , <http://literature.cdn.keysight.com/litweb/pdf/E7515-90001.pdf?id=2459161>
- [14] Keysight Technologies, "N6705 DC Power Analyzer and Source Measurement Unit (SMU) Modules – Product Fact Sheet", online resource <http://literature.cdn.keysight.com/litweb/pdf/5989-8615EN.pdf?id=1981218>
- [15] Blender Institute, "Big Buck Bunny", website <https://peach.blender.org/>



15 Appendix 1: Portal API REST

This Appendix describes the API REST provided by the Portal to access the test campaign information available at the Portal.

15.1 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"
}
```

15.2 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"}
```

15.3 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"
    }
  ]
}
```

15.4 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"
}
```

15.5 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"
}
```



15.6 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",
}
```

15.7 Campaigns

List all the campaigns in the application



16 Appendix 2: TAP plugins. Implementation details.

16.1 Quamotion WebDriver TAP plugin

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.

16.1.1 Quamotion WebDriver TAP plugin instrument

Table 13 describes the settings of the Webdriver instrument implemented in TAP.

Table 13 Quamotion WebDriver instrument settings

Settings	Type	LTE Bands
Base URL	String	Base URL for the WebDriver API calls. Default: "http:localhost:17894/wd/hub"
SessionReadyPollPeriod	Int	When waiting for a session to be ready, polling period. Default: 1000 (ms)
SessionReadyTimeout	Int	When waiting for a session to be ready, timeout. Default: 40000 (ms)

Table 14 describes the methods provided by the instrument to control the features offered by the Webdriver instrument through the TAP plugin.

Table 14 Quamotion WebDriver instrument methods

Method	Description
Open	
Close	
NewSession	<p>Creates a new session for the given app and device, and makes it the active session.</p> <p>Required parameters:</p> <ul style="list-style-type: none">• deviceId: The ID of the device• appId: The ID of the app <p>Optional parameters:</p> <ul style="list-style-type: none">• clearAppSettings: If true, clear the app settings before starting the session• reuseSession: If true and there is an existing session with the same device and app, use that instead of opening a new one <p>Returns:</p> <ul style="list-style-type: none">• Session

*WaitForSessionReady*

Waits a finite amount of time for the given session to be ready. The instrument polls the Webdriver periodically to find out if the session is ready. The polling period is defined by `SessionReadyPollPeriod`, and the number of retries by `SessionReadyMaxRetries`.

Required parameters:

- Session

Returns:

- True if the session is ready; false if the session was not ready before the timeout

GetActiveSession

Returns the currently active session, if any

CloseSession

Closes the given session.

Required parameters:

- session

CloseAllSessions

Close all sessions in the WebDriver server

16.1.2 Quamotion WebDriver TAP plugin test steps

16.1.2.1 Session management

All the user actions performed in a device must be performed in the context of a WebDriver session. A session corresponds to a device and an application on that device.

The first release of the plugin only support one active session at the same time.

16.1.2.1.1 New session

Creates a new session (or reuses an existing one) for the given device and app.

Can be used with children. The session will be automatically closed after the children are executed. Children can get the new session calling `GetSession()`.

Table 15 Quamotion WebDriver New session step settings

Settings	Type	Description
<i>Webdriver</i>	QuamotionWebdriverInstrument	
<i>DeviceId</i>	string	
<i>AppId</i>	string	
<i>AppFilePath</i>	string	Path to the app file (e.g APK for Android). Required if the app is no



		already loaded in Quamotion WebDriver
<i>ClearAppSettings</i>	bool	Close all active sessions before opening a new one
<i>CloseAllSessions</i>	bool	Close all active session before opening a new one
<i>ReuseSession</i>	bool	Reuse an existing session, if possible. The app will not be restarter in that case.

16.1.2.1.2 Close session

Closes the currently active session.

Table 16 Quamotion WebDriver Close session step settings

Settings	Type	Description
<i>Webdriver</i>	QuamotionWebdriverinstrument	

16.1.2.2 User actions

There is one separate test step for each of the supported user actions:

- Tap
- Long press
- Enter text
- Back

All of them have the following settings in common.

Table 17 Quamotion WebDriver User action steps settings

Settings	Type	Description
<i>Webdriver</i>	QuamotionWebdriverInstrument	
<i>Element</i>	string	UI element on which the action will be performed. The interpretation of this setting depends on the FindStrategy setting.
<i>FindStrategy</i>	FindStrategy	Use one of the following strategies when looking for the element: *Marked *Name



		<ul style="list-style-type: none">*Id*XPath*LinkText*PartialLinkText*ClassName*TagName*CssSelector
<i>Wait for element</i>	bool	If disabled, try to perform the action on the element immediately. Otherwise, the presence of the element will be checked periodically before executing the action.
<i>PollPeriod</i>	int	Polling period when checking the presence of the UI element
<i>PollTimeout</i>	int	Timeout (in milliseconds) when checking the presence of the UI element

16.1.2.2.1 Tap

Tap an element.

16.1.2.2.2 Long press

Long press an element.

16.1.2.2.3 Enter text

Enters a text in an element, such as a text field.

Table 18 Quamotion WebDriver Enter text step settings

Setting	Type	Description
<i>Text</i>	string	Text that will be entered in the UI element

16.1.2.2.4 Back

Does the back action on the device, e.g. the back button in an Android device.

This step only has the Webdriver setting.

16.1.2.3 Queries

Query the UI of the app for the presence of a given element, or for the value of a public property of that element.

Table 19 Quamotion WebDriver Query step settings

Setting	Type	Description
<i>Webdriver</i>	QuamotionWebdriverInstrument	



<i>Element</i>	string	UI element on which the action will be performed. The interpretation of this setting depends on the FindStrategy setting.
<i>FindStrategy</i>	FindStrategy	Use one of the following strategies when looking for the element: *Marked *Name *Id *XPath *LinkText *PartialLinkText *ClassName *TagName *CssSelector
<i>QueryProperty</i>	bool?	If enabled, the query will return the value of a public property from the selected UI element
<i>PropertyName</i>	string	Name of a public property from the selected UI element
<i>Wait for element</i>	bool	If disabled, try to perform the action on the element immediately. Otherwise, the presence of the element will be checked periodically before executing the action.
<i>PollTimeout</i>	int	Timeout (in milliseconds) when checking the presence of the UI element
<i>PollMaxRetries</i>	int	Maximum number of polling retries when checking the presence of the UI element
<i>EmitVerdict</i>	bool	If enabled, the step will emit one of the two verdicts configured below. If the element is present (or the expected value matches the actual value of the public property), then 'Verdict if true' will be emitted. Otherwise, 'Verdict if false' will be emitted.
<i>PropertyValue</i>	string	The expected value of the public property of the UI element. The values will be compared as strings.
<i>VerdictIfTrue</i>	Verdict	The verdict to emit if the UI element is present (or the property value matches the expected value).
<i>Verdict if false</i>	Verdict	The verdict to emit if the UI element is not present (or the property value does not match the expected value).



The step emits the following result:

Table 20 Quamotion WebDriver Query step results

Settings	Type	Description
<i>Name</i>	Type	Description
<i>ElementId</i>	string	
<i>FindStrategy</i>	FindStrategy	
<i>ElementPresent</i>	bool	True if the element was found
<i>QueryProperty</i>	bool	
<i>PropertyName</i>	string	
<i>PropertyValue</i>	value	The value of the property PropertyName, if it was requested

16.1.2.4 Replay

The step replays a recorded script that contains the required user actions.

Table 21 Quamotion WebDriver Replay step settings

Settings	Type	Description
<i>Webdriver</i>	QuamotionWebdriverInstrument	
<i>Script</i>	string	Path of the JSON file that contains the recorded user actions.
<i>Continue on Error</i>	bool	Whether or not to continue the execution of the script when an action fails.
<i>Wait for Elements</i>	bool	If disabled, try to perform each action immediately. Otherwise, check periodically the presence of the element on which the action will be performed before executing the action.

*Polling period*

int

Time in milliseconds to wait before checking for the presence of the element

16.2 OML TAP plugin

This plugin allows TAP to send the results reported during an experiment to an OML server (which will store them in a database). This plugin has two main features:

- A `ResultListener` that integrates with TAP way of reporting results
- A test step that sends arbitrary CSV files to the OML server

16.2.1 OML TAP plugin instruments

The `OMLServerInstrument` instrument represents an OML server that collects measurements from experiments.

Table 22 OMLServerInstrument settings

Setting	Type	Description
<i>Uri</i>	string	URI of the OML server. Valid URIs include: * 'hostname:port': for regular OML servers (default port is 3003) * 'file:filename': for local files

Table 23 OMLServerInstrument public properties

Property	Type	Description
<i>Domain</i>	string	Name of the current OML domain, i.e. experiment. Measurements handled by this instrument will be stored in a database with this name.

Table 24 OMLServerInstrument methods

Method	Description	Method
<i>TryParseTcpUri</i>	Returns the host and port parts of a OML TCP URI	TryParseTcpUri



The `ICsInstrument` is an interface that extends `IInstrument`. It is used in conjunction with the step that sends CSV files to OML, in order to know which are the CSV files produced by a tool. This is intended as a thin wrapper around an external tool that is not properly integrated with TAP yet, and thus does not publish its results to `ResultListeners`.

Table 25 ICsInstruments public properties

Property	Type	Description
<i>CsvMeasurementPoints</i>	<code>IEnumerable<CsvMeasurementPoint></code>	A mapping between the measurement points provided by the tool, and the CSV file that contains the data for that measurement point

16.2.2 OML TAP plugin test steps

The “Configure OML” step configures an OML experiment (aka domain) for the given OML instrument for the rest of the test plan execution.

Table 26 OML TAP plugin Configure OML step settings

Setting	Type	Description
<i>CustomName</i>	<code>Enabled<string></code>	If disabled, a random name will be generated for the instrument. If enabled, use this name.

The “Send CSV” sends a CSV file to the OML server. The CSV file can be provided as a file path, or by an implementation of `ICsvInstrument`, which can provide more than one (with the respective measurement point name).

The CSV file will be read whole and its contents sent to the OML server. A file will be considered as one measurement point, and thus will be stored in one table. If the CSV file has headers, those will be used as the names of the columns.

The step will support the following OML types:

- `Int32`



- Int64
- Double
- String

CSV values will be converted to the one that makes more sense. In particular, boolean values will be stored as strings (for now at least).

Table 27 OML TAP plugin Send CSV step settings

Setting	Type	Description
<i>OmlServer</i>	OmlServerInstrument	
<i>CsvSource</i>	CsvSourceEnum	The source of the CSV file (or files) to send: * File * Instrument
<i>CsvInstrument</i>	ICsvInstrument	The CSV files, and the respective measurement point names, will be provided by this instrument
<i>FilePath</i>	string	The path to the CSV file to send
<i>HasHeaders</i>	bool	If enabled, the first non-empty line of the CSV is assumed to define the headers of each column.
<i>Separator</i>	Comma, tab, semicolon	Column separator for the CSV file. Default is comma.
<i>AppName</i>	string	The name of the app sending the CSV data. If empty, and the source is an instrument, its name will be used. Otherwise, "csv2oml" will be used as the default.

*MeasurementPointName*

string

The name of the measurement point for the CSV data

16.2.3 OML TAP plugin result listeners

The OML result listener integrates with TAP to send all the results produced by test steps to an OML server. Each TAP `ResultsTable` object will be mapped to one measurement point of the same name. Since TAP does not distinguish the source of the results, all of them will be mapped to the same OML app and node ID, which can be set in the result listener.

Table 28 OML result listener settings

Setting	Type	Description
<i>OmlServer</i>	OmlServerInstrument	OML server where measurements will be sent
<i>SenderId</i>	string	ID of this node, for OML
<i>Appname</i>	string	Name of this app, for OML

16.3 App instrumentation TAP plugin

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.

16.3.1 App instrumentation TAP plugin steps

16.3.1.1 Parse Measurements

This step can be used for extracting the measurements provided by the instrumentation library, from a previously saved file, or directly from the device by using Logcat. The step is compatible with the measurement points defined in the instrumentation library, so it is able to extract all the measurements without the need for any additional configuration.

Table 29 Settings for the Parse Measurements step

Step	Setting	Description
<i>Parse Measurements</i>	Source Measurement source	Source type from where the measurements will be extracted. The available sources are 'adb' and 'Logcat file'
	Android / adb	When using adb, this setting selects the adb instrument to be used.
	Android / Device ID	Android device ID to use, this setting is only required when multiple devices are connected through adb



File / File path

Path of the file that contains the output received from Logcat.

16.3.1.2 Parse Regex in Logcat

This step can be used for generating custom results by reading specifically formatted messages from Logcat. The step looks for messages that match a regular expression defined by the user, and reports each of the matched groups in a new TAP result. The name and columns of this result are also defined by the user, with every matched group content being assigned to one of the columns in order.

Table 30 Settings for the Parse Regex in Logcat step

Step	Setting	Description
Parse Regex in Logcat	DUT / adb	Adb instrument that will be used.
	DUT / Device ID	Android device ID to use, this setting is only required when multiple devices are connected through adb
	Logcat filter / Filter by tag	Configures the step for listening to specific tags in the messages.
	Logcat filter / Tag	Tag filter to use
	Logcat filter / Priority	Priority of the messages.
	Results / Result name	Name of the results that will be generated by the step.
	Results / Regex	Regular expression that will be used for generating the results. The regular expression must contain one capture group for each of the columns in the results generated.
	Results / Column names	Comma separated list of column names for the generated result.

16.4 Android TAP plugin

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.

16.4.1 Android TAP plugin instruments

The plugin defines a new TAP instrument (AdbInstrument) that acts as an interface between the steps and the device. This instrument makes use of an ADB server, which can be running on the same machine as TAP or on a remote machine accessible through a network connection.

Table 31 Android TAP plugin instruments

Instrument	Setting	Description
Adb Instrument	adb / Executable	Location on the local machine of the adb executable file.



Remote / Remote adb	Specifies whether to connect to an adb server running on the local machine (default) or on a remote machine as configured in the settings below.
Remote / Host	Address of the remote server host
Remote / Port	Port where the remote adb server is listening for connections.

16.4.2 Android TAP plugin steps

There are some settings that are shared among the steps:

Table 32 Settings shared by the Android steps

Step	Setting	Description
All steps	adb / adb	The adb instrument targeted by the step
	Device / Device ID	Optional. If more than one device is connected to the adb server, the commands are sent to the device specified in this setting.
Adb Command and Activity Manager	Advanced / Number of retries	Number of times in which the action is retried before emitting a 'Fail' verdict.
	Advanced / Retry Wait	Period to wait between two consecutive retries.
	Advanced / Timeout	Time to wait before aborting the execution of the action.

16.4.2.1 Adb Command

The Adb command step can be used for sending a set of predefined commands to the device through adb, or for sending a custom command by specifying the arguments that need to be sent through adb. The supported commands, and their settings are:

- Custom: Sends the specified arguments through adb. This command can be used to perform any action supported by adb, even if it is not supported by any other step.

Table 33 Settings for the Custom command on the Adb Command step

Step	Setting	Description
Adb Command	Command Arguments	Arguments to send through adb, in the same format as in the command line.

- Push, Pull: Sends or retrieves a file from/to the device.

Table 34 Settings for the Custom command on the Adb Command step

Step	Setting	Description
Adb Command	Command / Local file	Path of the file to be sent or saved in the local machine
	Command Remote file	Path of the file to be retrieved / saved on the device



- Install: Installs the selected package on the device.

Table 35 Settings for the Custom command on the Adb command step

Step	Setting	Description
Adb Command	Command / Local file	Path of the apk package that needs to be installed
	Command / Install options	Options for the adb install command

- Uninstall: Uninstalls the selected package from the device:

Table 36 Settings for the Custom command on the Adb Command step

Step	Setting	Description
Adb Command	Command / Package name	Name of the package that needs to be uninstalled.

- Reboot: Restarts the device. This command does not need additional settings.

16.4.2.2 Activity Manager

This step can be used to send commands to a device's Activity Manager. The Activity Manager can perform actions such as starting and stopping applications, or broadcasting intents.

All the commands supported by this step can be sent using the Custom command on the adb command step, but this step can be used to easily perform the most common actions of the Activity Manager. The supported commands are:

- Start: Starts an activity
- Start Service: Starts the service specified
- Force Stop: Stops all the processes associated with the specified package
- Broadcast: Issues a broadcast intent.

The following table details the available settings for the Start, Start Service and Broadcast commands:

Table 37 Settings for the Start, StartService and Broadcast commands on Activity Manager

Step	Setting	Description
Activity Manager	Intent / Component	Specifies the component name, including the package name prefix, to create an explicit intent
	Intent / Action	Specifies the intent action
	Intent / Data URI	Specifies the intent's data URI
	Intent / MIME Type	Specifies the intent's MIME type
	Intent / Selector	Used to specify the intent
	Intent / Category	Specifies an intent category



Intent / Selector	Used to specify the intent resolution by using the values on the Data URI and MIME Type settings
Intent / Flags	Extra flags that control the execution of the intent
Intent / Intent Extras	Represents a list of intent extras that are to be used by the intent. These extras act as additional parameters that are sent to the activity that performs the actions specified in the intent.

Only one setting needs to be specified for the Force Stop command:

Table 38 Settings for the Force Stop command on Activity Manager

Step	Setting	Description
<i>Activity Manager</i>	Package / Package	Specifies the package name

16.4.2.3 Adb Airplane Mode

The Airplane Mode step can be used to enable and/or disable this mode on the device.

Table 39 Settings for Adb Airplane Mode step

Step	Setting	Description
<i>Adb Airplane Mode</i>	Procedure / Procedure	Specifies the type of procedure to perform: Enable, Disable or Attach. The Attach procedure forces the device to attach to the cell by enabling and then disabling Airplane Mode.

16.4.2.4 Logcat

Logcat is a command line tool that dumps a log of system and application messages from the device. These messages are commonly generated by invoking the methods on the Log class from an Android application.

The Logcat step can be used for retrieving the messages from Logcat. These messages can be saved to a file, either on the local machine or on the device, or displayed in TAP. The user can also configure the step so that the messages are saved in the background, and later retrieved by the *Retrieve Background Logcat* step.

Table 40 Settings for the Logcat step

Step	Setting	Description
<i>Logcat</i>	Filter / Filter tags	List of specific tags to be filtered, and their priorities
	Filter / Default filter	If enabled, causes the step to filter all tags not explicitly filtered in <i>Filter tags</i>
	Filter / Buffers	Specific Logcat buffers to listen
	Execution / Execution Mode	Execution mode of the step. In <i>Instant</i> mode, the step retrieves the current contents in Logcat. In <i>Continuous</i> mode the step continue running in the background, retrieving all the content sent to Logcat



	until it is stopped by a <i>Retrieve Background Logcat</i> step.
Output / Format	Format of the saved Logcat messages.
Output / Target	Where the output will be saved or displayed.

16.4.2.5 Clear Logcat

This step can be used to clear the Logcat output from an Android device. Use of this step ensures that the Logcat output retrieved by the Logcat step does not contain messages emitted before the test plan execution. This step does not require additional settings.

16.4.2.6 Retrieve Background Logcat

The Retrieve Background Logcat step can be used to stop the background process generated by the *Continuous* mode of the Logcat step, retrieving the logcat contents from it.

Table 41 Settings for the Retrieve Background Logcat step

Step	Setting	Description
Logcat	Logcat / Background Logcat	Background Logcat process to retrieve, as returned by a previous Logcat step.
	Logcat / Delete Device Files	If enabled, the step will remove the temporal files saved on the device once the contents have been retrieved.
	Output / Local File	Path of the file to be saved with the contents of Logcat.

16.5 iOS TAP plugin

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.

This plugin will make use of the libimobiledevice library [1]. This library provides tools that let a user communicate with services of iOS using simple commands, similar to Android ADB. These commands can remotely restart a device and save its syslog, among other actions. To capture traffic, a virtual interface of the device will be created and tshark software will capture the traffic that passes through it. To achieve the actions aforementioned, the iOS device must be connected to a MAC OS machine. The plugin will handle the connection to the remote MAC OS machine using SSH protocol. Plink software will be used at the local machine to execute libimobiledevice commands in the remote machine. These commands will run certain processes at the iOS device.

Table 42 shows the instrument provided by the first release of the iOS TAP plugin. A single instrument called *iOSInstrument* will offer the basic properties and methods to send the libimobiledevice commands to the iOS device. The instrument will handle the connection to the remote MAC OS machine, and thus the user has to provide the host and SSH port where the remote machine is running and the path to the plink executable in the local machine. The paths



to the libimobiledevice library and to tshark executable will also be provided. The instrument verifies if the processes launched are actually started by querying its PID in the remote machine and setting the steps verdict accordingly.

Table 42 - iOS TAP plugin instrument

Instrument	Setting	Description
<i>(iOSInstrument)</i>		Handles the connection to the remote MAC OS machine to communicate with the iOS device using libimobiledevice commands.
	Plink executable	Path to the plink executable in the local file system that will be used to interact with the MAC OS machine where the iOS device will be connected to.
	Host	Host name or IP direction of the MAC OS machine.
	Port	Port of the MAC OS machine.
	User	User of the MAC OS machine.
	Pass	Password of the MAC OS machine.
	Libimobiledevice executable	Path to the libimobiledevice directory in the remote MAC OS file system that will contain the tools used to interact with the iOS device.
	Tshark executable	Path to the tshark executable in the remote file system that will be used to capture traffic in the iOS device

Table 43 shows the steps provided by the TAP plugin. iOSRestartStep performs the restart of the iOS device, which can be used to force the attach of the device to a base station. iOSStartLoggingStep starts saving iOS syslog in a file in the remote MAC OS machine and iOSStopLoggingStep stops saving it. This way, one can save only the syslog provided by iOS in a given time interval. iOSStartTrafficCaptureStep and iOSStopTrafficCaptureStep work in a similar way to save network traffic received and sent by any interface of the iOS device. Finally, iOSLaunchAppStep tries to launch an app installed in the device.

Table 43 - iOS TAP plugin test steps

Test step	Setting	Description
<i>Restart Device</i> <i>(iOSRestartStep)</i>		Restarts an iOS device.
	iOS / iOS	iOSInstrument which will handle the communication.
	Device / Device ID	UDID of the targeted iOS device.
<i>Start Logging</i> <i>(iOSStartLoggingStep)</i>		Starts saving the iOS syslog in a file in the remote machine.
	iOS / iOS	iOSInstrument which will handle the communication.
	Device / Device ID	UDID of the targeted iOS device.



<i>Stop Logging</i> (<i>iOSStopLoggingStep</i>)	Options / Logfile Name	Path of the file to save the syslog in the remote machine.
		Stops saving the iOS syslog in the remote machine.
	iOS / iOS	iOSInstrument which will handle the communication.
<i>Start Traffic Capture</i> (<i>iOSStartTrafficCaptureStep</i>)	Device / Device ID	UDID of the targeted iOS device.
		Starts saving iOS network traffic in a pcap file in the remote machine.
	iOS / iOS	iOSInstrument which will handle the communication.
<i>Stop Traffic Capture</i> (<i>iOSStopTrafficCaptureStep</i>)	Device / Device ID	UDID of the targeted iOS device.
	Options / Capture file name	Path to the pcap file to save captured traffic in the remote machine.
		Stops saving iOS network traffic.
<i>Launch App</i> (<i>iOSLaunchAppStep</i>)	iOS / iOS	iOSInstrument which will handle the communication.
	Device / Device ID	UDID of the targeted iOS device.
		Starts an app in the iOS device.
	iOS / iOS	iOSInstrument which will handle the communication.
	Device / Device ID	UDID of the targeted iOS device.
	Options / Bundle id	Bundle id of the app that will be started.
	Options / Attempts	The number of attempts to launch the app

Figure 31 shows the main classes that will be part of the implementation of this plugin for iOS.

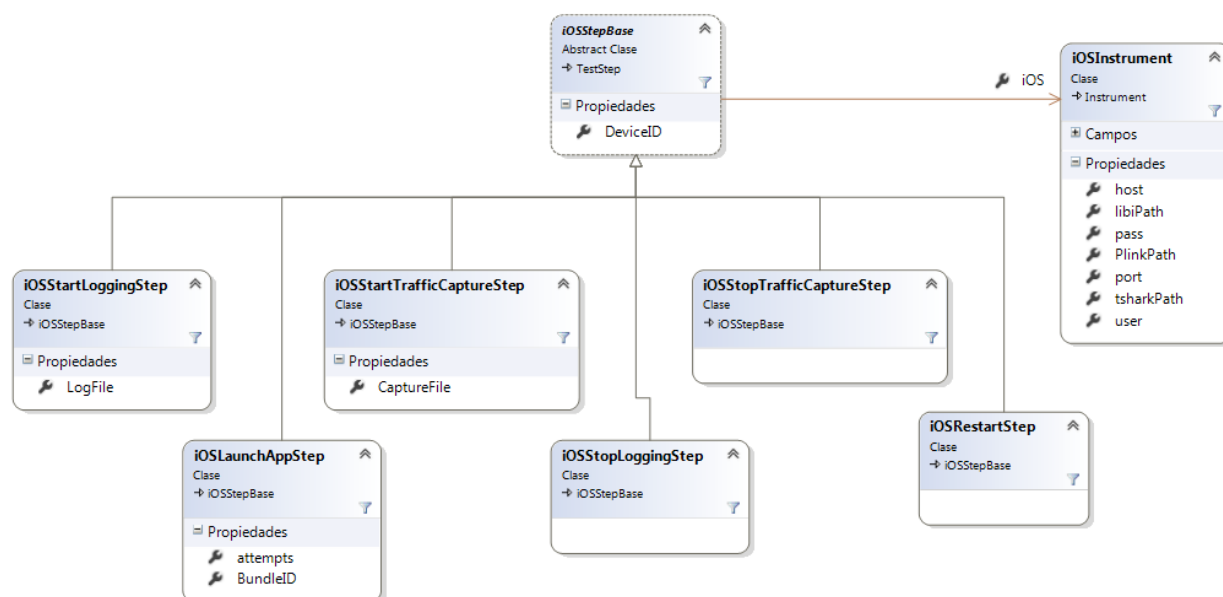


Figure 31 Main classes of TAP plugin for iOS

16.6 Impairments TAP plugin

This plugin allows TAP to control an impairments SCPI server that can set impairments on remote hosts using SSH and netem.

16.6.1 Impairments TAP plugin instruments

The instrument provided by this plugin is able to control the TRIANGLE Impairments System running on a remote host by using the SCPI interface. This instrument can be configured so that all the impairments are automatically disabled after a test plan has been completed.

16.6.2 Impairments TAP plugin steps

16.6.2.1 Set Link Impairments

This step can be used to configure the impairments applied on a specific link. The supported impairments are latency, link errors, packet duplication and packet reordering.

Table 44 Settings for the Set Link Impairments step

Step	Setting	Description
Set Link Impairments	Common / Alias	Alias of the link that will be configured
	Common / Impairments System	Impairments system that will be targeted by the step
	Latency / Latency Enabled	Enable or disable the latency impairment on the link.
	Latency / Mean Latency	Mean latency in milliseconds
	Latency / Variance	Latency variance



Latency Correlation	/	Correlation percentage of the latency
Latency Distribution	/	Statistic distribution of the latency. The supported distributions are Normal, Pareto and Pareto-Normal.
Link Errors / Link Errors Enabled		Enable or disable the Link Errors impairment on the link.
Link Errors / Mean		Mean percentage of link errors.
Link Errors Correlation	/	Correlation percentage of the link errors.
Duplication Duplication Enabled	/	Enable or disable the Duplication impairment on the link.
Duplication Percentage	/	Percentage of duplicated packets on the link
Reordering Reordering Enabled	/	Enable or disable the Reordering impairment on the link.
Reordering Percentage	/	Percentage of reordered packages in the link.
Reordering Correlation	/	Correlation percentage of reordering.

16.6.2.2 Reset impairments

The reset impairments step can be used to disable all the configured impairments on every link controlled by the impairments system.

Table 45 Settings for the Reset Impairments step

Step	Setting	Description
<i>Reset Impairments</i>	Common Impairments System	Impairments system that will be targeted by the step

16.7 RF Switch TAP plugin

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.

16.7.1 RF Switch instruments

The plugin provides an instrument for controlling an 11713C attenuator/switch driver through the SCPI interface. The instrument is able to open/close the connections on two Banks (with two L7104A switches each, one for RX and the other for TX). Each switch provides four possible paths. Using this configuration, the instrument is able to handle the connection of a maximum of eight devices.



16.7.2 RF Switch steps

16.7.2.1 Close all paths

This step can be used for closing all the paths controlled by the switch driver, so that all the devices are disconnected.

Table 46 Settings for the Retrieve Background Logcat step

Step	Setting	Description
<i>Close all paths</i>	Common / RF Switch	11713C attenuator/switch driver that will be targeted by the step.

16.7.2.2 Open path

The Open Path step can be used to open matching paths on both L7104A switches of the selected bank. This will connect the TX and RX RF ports of one of the devices connected to the TRIANGLE testbed, closing all other connections so that only one device is used at any given time.

The step can also be used to configure the attenuation level used on the connection.

Table 47 Settings for the Open Path step

Step	Setting	Description
<i>Open path</i>	Common / RF Switch	11713C attenuator/switch driver that will be targeted by the step.
	Settings / Bank	Bank where the device to be used is connected.
	Settings / Path	
	Settings / Sett Attenuation	Attenuation level of the connection.

16.7.2.3 Connector switching

The Connector Switching can be used to configure the connections on the rear panel of the 11713C attenuator/switch driver. This can be used to change the position of a coaxial switch connected to the rear panel.

Table 48 Settings for the Connector switching step

Step	Setting	Description
<i>Connector Switching</i>	Common / RF Switch	11713C attenuator/switch driver that will be targeted by the step.
	Bank 1 / S9 A\B	Enable or disable switching on the S9 connector on Bank 1
	Bank 1 / S0 A\B	Enable or disable switching on the S0 connector on Bank 1
	Bank 2 / S9 A\B	Enable or disable switching on the S9 connector on Bank 2



Bank 2 / S0 A\B

Enable or disable switching on the S0 connector on Bank 2

16.8 GPS emulation TAP plugin

GPS emulation is a new feature of the testbed introduced in Section 11. This section describes the TAP plugin implemented for its control. The TAP plugin is based on a REST API also described in this section.

16.8.1 REST service

16.8.1.1 Resource definition

For the control of the emulator a RESTful (Representational state transfer) web service has been implemented. This service runs on the emulator system and allows the user to send petitions that are transformed in actions over the USRP (Universal Software Radio Peripheral). The user only needs to configure all the parameters required for the generation of the signal, which would be the resource of the service.

Specifically in this service, the resource will consist on a set of different types of data items, each one corresponding with a parameter of the emulator configuration. The exact structure of this resource is shown in the next piece of code. This structure covers all the possible fields that have been implemented so far.

```
list_parameters=[
{
  'id':0,
  'kmlfile':'statuePATH.kml',
  'outputCSV':'UMfile',
  'sPeriod':0,
  'speed':2,
  'numbites':16,
  'efemerides':'efemerides2903.17n',
  'simulationFile':'gpssim.bin',
  'sampleRate':2500000, #hz
  'gain':'0',
  'duration':'0',
  'txFreq':'',
  'state':'WAITING'
}
```

Not only one structure has been created but a set of them, which allows the user to save different configurations as long as the service is running.

16.8.1.2 Methods definition

Once the resource has been defined, the next step is to implement all the methods that will act over these resources.



The root URL to access to the resources of this service is: <http://hostname/GPSsimulation>, in which “hostname” would be the IP address of the server where the service is running. This service uses the HTTP language basic methods: GET, POST, PUT and DELETE.

These are all the URLs to access the resources implemented in the service:

<http://hostname/GPSsimulation>

http://hostname/GPSsimulation/command_id

http://hostname/GPSsimulation/command_id/start_simulation

http://hostname/GPSsimulation/command_id/start_static_simulation

Using the GET method and the URL <http://hostname/GPSsimulation> all the resources stored in the server are listed.

The “command_id” field is the resource’s identification, it can be manually or automatically defined, and enables access to a specific resource.

The last two URLs have the same functionalities, the only difference is the emulation mode: static or dynamic. The first one starts the emulation of the KML file contained in the resource, and the second one extracts the first coordinate and then generates the appropriate signal.

Table 49 illustrates all the rest of the functionalities the service can do, for each URL several methods can be applied.

Table 49 – GPS API Rest

	/command_id	/command_id/start_simulation /command_id/start_static_simulation
<i>GET</i>	Shows a specific resource	Shows the emulation state
<i>POST</i>	Creates a new resource	Starts a new emulation
<i>PUT</i>	Updates a resource	Not implemented
<i>DELETE</i>	Removes a resource	Stops an emulation currently running

16.8.1.3 TAP plugin

In order to integrate and control the emulator in the TAP platform, an Instrument called USRP and four test steps have been implemented: PrepareSimulation, StartSimulation, SettleGPS, StopSimulation, and GetState.

The instrument has as configurable parameters the IP direction and the port of the server. It contains the necessary functions to allow the communication with the server.

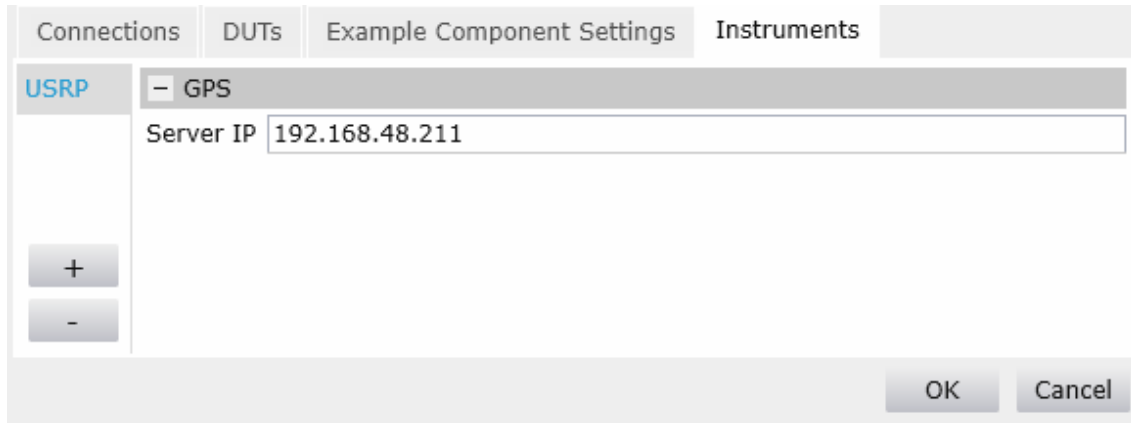


Figure 32 URSP instrument

The PrepareEmulation step creates the necessary files to start an emulation for both cases, either a static one or an emulation of a complete route. The parameters needed to configure any emulation must be setting here. Besides the selection of the route itself, it allows the configuration of the speed, and the signal gain. The KML file and the speed are always required. The default value for the gain is 0 dB, which in most cases should be enough to obtain a good signal.

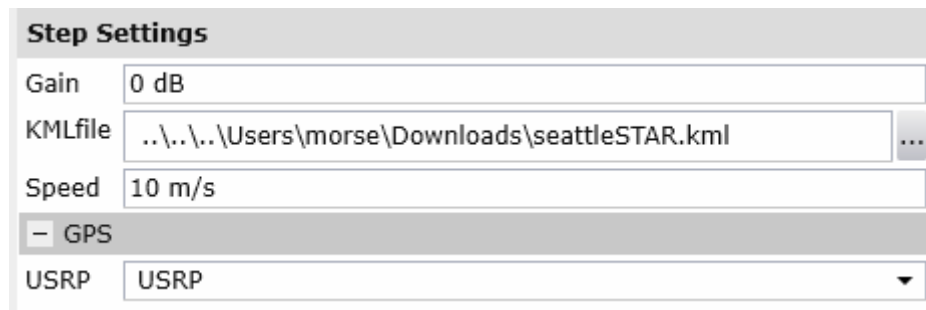


Figure 33 PrepareEmulation step

The StartEmulation step starts an emulation of a particular route previously uploaded by the user, the only parameter that can be specified here is the duration of the emulation. If this parameter is not specified, the default value used is the time that it takes to go over the route at the selected speed. This step must be executed after the Preparing step has finished.



Figure 34 StartEmulation step

The SettleGPS step performs the static simulation of the route's first position with the aim of settling the GPS signal in the receiver device. As in the previous step, the only configurable parameter is the duration of the emulation. If no value is selected the default duration is the necessary until the mobile phone has reached a fixed signal.



The StopSimulation step stops the signal emulation. It acts on either the static or the dynamic emulation. This step does not have any parameter to configure as they are not necessary to accomplish the function requested.

The GetState step provides the state of the current emulation signal. There are eight possible states implemented: WAITING, IN PROCESS, SIMULATION FILE CREATED, RUNNING, FIXED POSITION, FINISHED, STOPPED and ERROR.

The idle state of any emulation process is "WAITING". Once the service received the order to start a new emulation, it starts the creation of the simulation file using the software GPS-SDR-SIM and the state would change to "IN PROCESS" until the creation of the file ends, and then the state changes to "SIMULATION FILE CREATED".

If there are no problems during the signal emulation, the next state would be "RUNNING", and the USRP would be transmitting the signal to the device. Once the emulation ends, the state will change to "FINISHED".

If the emulation is interrupted on purpose, the state will show this as "STOPPED". Also, if there is some error during any part of the emulation the state would inform about it.

Additionally the state "FIXED POSITION" indicates when the mobile has reached a fixed signal and has a stable position. This is used to stop sending a static emulation signal and start the emulation of the complete route.

16.9 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.

This plugin adds two new test steps: "Dynamic Sequence" and "Break Event". These test steps allow to finely control the test execution flow by indirectly introducing the concept of master and slave sequences. The core idea is to classify the sequence containing the application flow as master, and the sequence containing the network scenarios as slave. The slave sequence will run in a loop while the master sequence is running, and to exit when the master is finished.

This is achieved by inserting all network scenarios test steps under a Dynamic Sequence test step, and they will loop until the Break event test step is reached. Naturally, the Break Event test step is located at the very end of the master loop containing the application flow, thus making the slave sequence (containing the network scenarios) loop until the application flow is complete.

The Dynamic Sequence test step additionally contains the feature to start at random child step, to guarantee even distribution of sub scenarios' execution.

This logic is illustrated on Figure 35: the 4 sub scenarios A, B, C, D loop until the Trigger break event contained the Parallel 1 sequence is reached. The initial subscenario is also random.

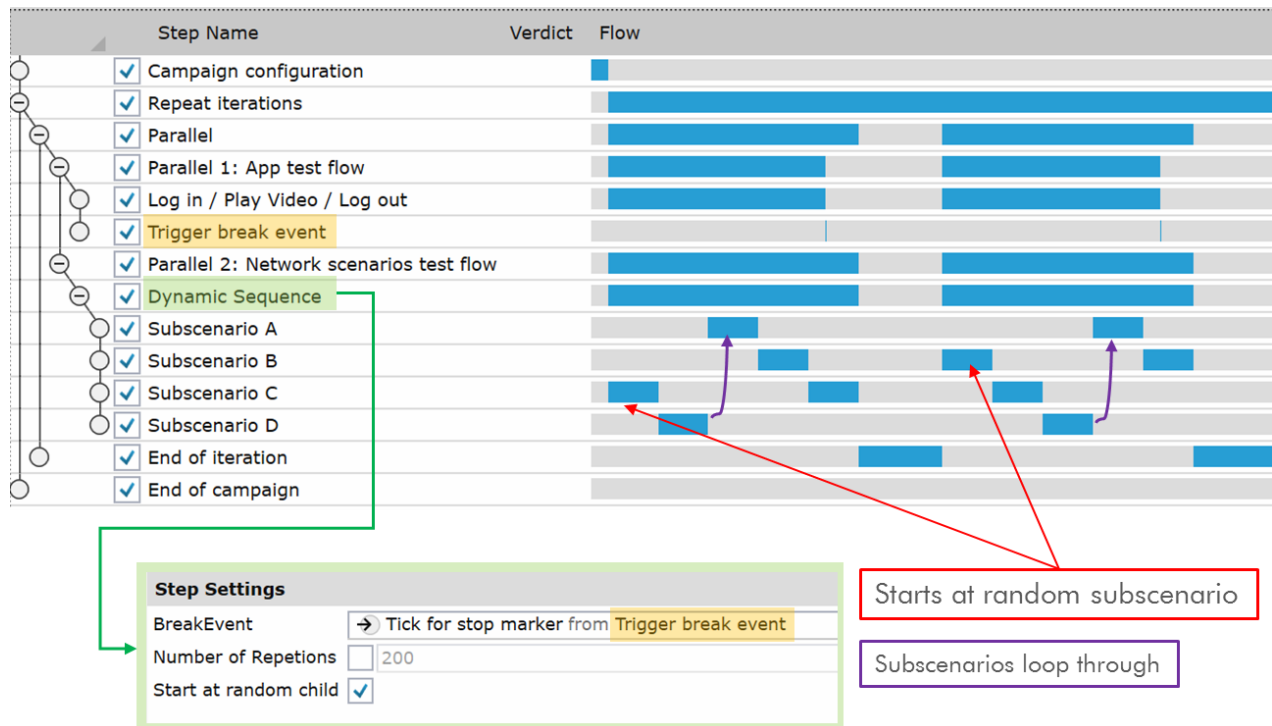


Figure 35 Dynamic Sequence to enable master & slave sequences

16.10 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.

16.11 KPIs calculation plugin

Finally, a KPI calculation plugin has been created. 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. For instance, the plugin gets pointed to a test result, and it detects that it is a result from a "User Experience with Reference Apps" domain, "Social Networking" use case, test case "001" (which is Picture Posting), and will expect to find measurements corresponding to the timing of posting pictures on a reference social networking application. The plugin will then parse the measurements and calculate the KPIs as described in the DRA (Mobile devices User Experience with reference apps) Test Specification, and pass these KPIs into a database to be aggregated further.

16.12 Plugins to reach TAP server

As described in section 4.2.3, when TAP is executed as a server, it can be reached remotely to run test cases.

There are two means to reach the TAP server:

- TCP remote



- A TCP connection is made to the machine hosting TAP, requesting the TCs to be executed
- REST API
 - An API to communicate directly to the TAP server is used instead

16.13 DEKRA TAP plugin

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

16.13.1 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 50 – DEKRA Tool RC Server channel

Item	Setting
<i>LAN IP Address</i>	IP address of the Performance Tool
<i>Protocol</i>	TCP
<i>Port</i>	11500
<i>End of Sentence</i>	'\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.

16.13.2 Instrument

Table 51 shows the operations implemented for TAP Instrument management.

Table 51 – DEKRA Tool TAP Instrument

Operation	Description
<i>Open</i>	Creates the socket with the DEKRA Tool RC Server
<i>Close</i>	Closes the socket with the DEKRA Tool RC Server
<i>Settings</i>	DEKRA Tool RC Server settings: IP address, port, operation mode, and project/session name
<i>End of Sentence</i>	'\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 36 DEKRA Tool TAP Instrument

16.13.3 DUT (TACS4-Agent)

Table 7 shows the operations implemented for TAP DUT management.

Table 52 – DEKRA Tool TAP DUT

Operation	Description
<i>Definition</i>	Defines the Agents which participate in the test. Adding a DUT does not necessarily implies its usage.
<i>Configuration</i>	Defines the configuration of the TACS4-Agent: IP address, Port, etc.
<i>Open/Close</i>	There is no method for open/close. TAP by default initiates the DUT before the Instrument.
<i>Add Agent</i>	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	<input type="text"/>
	Comment	<input type="text"/>
	- 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	<input checked="" type="checkbox"/>
	- Cellular Parameters	
Capture Cellular Parameters	<input type="checkbox"/>	
- GPS Parameters		
Capture GPS Parameters	<input type="checkbox"/>	

Figure 37 DEKRA Tool TAP DUT

16.13.4 Test Step

Table 53 shows the operations implemented for TAP Test Steps management.

Table 53 – DEKRA Tool TAP Test Steps

Operation	Description
Configuration	<p>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;</p> <ul style="list-style-type: none"> Video Id Timeout
Execution	<p>The test steps have basically three methods and a constructor.</p> <p>The constructor initializes the input parameters with default values.</p> <p>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.</p> <p>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.</p>

- YouTube Profile Setting	
Profile Name	YouTube1
Video ID	9ZfN87gSjvI
Live YouTube Video	<input type="checkbox"/>
- YouTube Playback Setting	
Timeout	absolute
Absolute Timeout	60 s



Figure 38 DEKRA Tool TAP Test Step (YouTube test)

16.13.5 Run/Stop Test

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



Figure 39 DEKRA Tool TAP Run/Stop Test

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

16.13.6 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 54 shows an extract of the RC server specification which has been used to implement the DEKRA TAP plugin get results procedures.

Table 54 – DEKRA Tool RC Server Get Results

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

16.13.7 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).

16.14 VELOX plugin

The testbed includes an over-the-top-content enabler, the RedZinc VPS Engine or Velox. Velox can be used by third-party applications to configure traffic prioritization to request a specific quality of service (QoS).



RedZinc provides the Velox engine. This allows the testbed to offer a service with specific bandwidth request API on radio network.

There is an on-demand service which can be activated and deactivated in real-time or “on-demand”. Velox is accessible via the TAP plugin or directly via its API.

The VELOX TAP plugin was created to work as a simple VELOX API client, with the 3 steps described in the following table.

Table 55 - VELOX TAP Steps

Test step	Setting	Description
<i>VELOX Access Setup</i>		Configures the access details of the VELOX Multi-domain Orchestrator
	VELOX Address	IP of the VELOX Server
	VELOX Port	Port where the VELOX API is listening
	VELOX API Key	VELOX API Key for service activation
<i>Create VELOX QoS Slice Session</i>		Creates a VELOX QoS based Virtual network Path Slice session
	Source Address	IP address of the origin end-point (IPv4 or IPv6)
	Destination Address	IP address of the destination end-point (IPv4 or IPv6)
	VELOX Service Slice	Pre-packaged Bandwidth Slice Sizes
<i>Stop VELOX QoS Slice Session</i>	VELOX Session ID	ID generated by VELOX upon slice creation for use in stop step
		Stops a VELOX QoS based Virtual network Path Slice session
	VELOX Session ID	Session ID of the Slice previously created, obtained from the creation step

All necessary information for the VELOX Access Setup is to be provided to experimenters by the testbed manager.

Experimenters themselves must provide the source and destination IPs for the slice creation (ipv4 and ipv6 supported) and must choose one of the pre-packaged VELOX Slice Services from the pool of 1,2,5,10,25,50 Mbps.

This plugin allows experimenters to control when QoS based bandwidth slices are activated/deactivated for specific end-points at specific times in the tests without requiring software changes on their applications. To this end, no specific ties between steps were established so that experimenters are free to place them anywhere in the test, with the caveat that the setup step should always be the first and that stop should be after create, otherwise the test steps will fail.



Test Plan *Untitled **

+ - [Up Arrow] ▶Run ▶|| ■ ✓ Repeat ▾

	Step Name	Verdict	Bin	Duration	Step Type
<input checked="" type="checkbox"/>	VELOX Access Setup				VELOX \ VELOX Access Setup
<input checked="" type="checkbox"/>	Create VELOX QoS Slice Session				VELOX \ Create VELOX QoS Slice Session
<input checked="" type="checkbox"/>	Stop VELOX QoS Slice Session				VELOX \ Stop VELOX QoS Slice Session

Figure 40 - Expected VELOX Step Order



17 Appendix 3: Android Instrumentation Library Usage

This appendix describes the particular implementation and usage details of the Android Instrumentation Library. The same library can be used from Unity apps, as described later in this section.

17.1 Including in an Android project

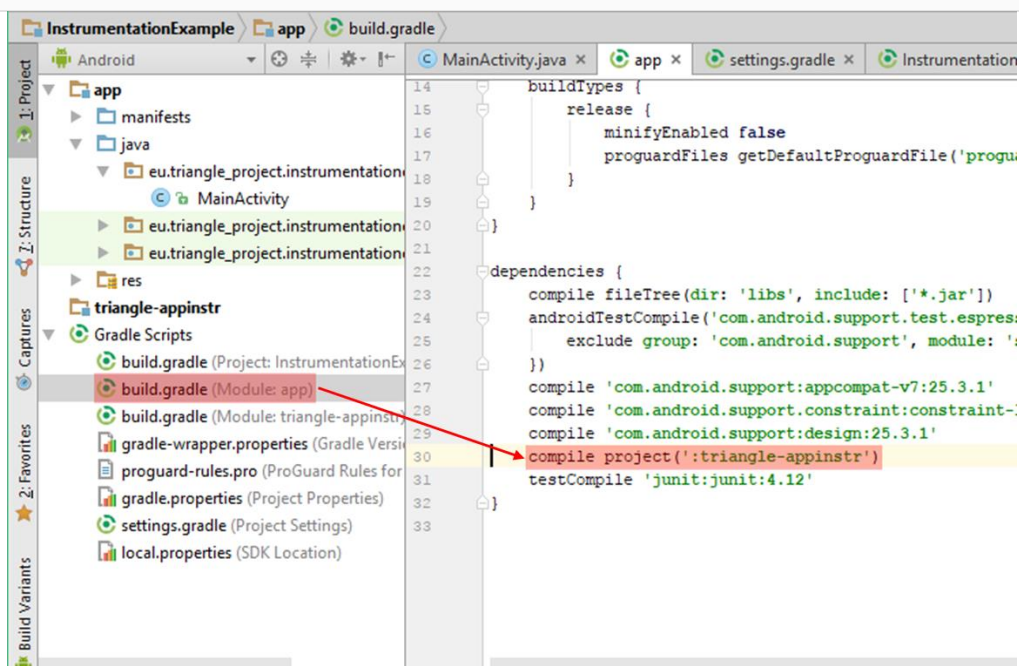
The library is distributed as an [AAR library file](#). To include the library in your own Android Studio project, follow the [instructions](#) in the Android Developers site:

1. Download the AAR file
2. Open the app project in Android Studio
3. Import the file as a new module:
 1. Click **File > New > New Module**
 2. Click **Import .JAR/.AAR Package** then click **Next**
 3. Enter the location of the AAR file
 4. Name the subproject “TRIANGLE-appinstr”
 5. Click **Finish**
4. Make sure the library is listed at the top of your settings.gradle file:

```
include ':app', ':TRIANGLE-appinstr'
```

5. Open the app module's build.gradle file and add a new line to the dependencies block:

```
dependencies {  
    compile project(':TRIANGLE-appinstr')  
}
```



6. Finally, click **Sync Project with Gradle Files**.



Please, see the Android Studio project provided with the Android Library as an example of how to include it in an Android project.

Other ways of integrating the Library in an Android app may work, but are not supported.

17.2 Usage

The library classes are organized into packages according to the use cases. Each package contains one class per feature of that use case. Each class contains one method per measurement used to compute the KPIs relevant to that feature.

The base package for the Android Library hierarchy is `eu.TRIANGLE_project.appinstr`.

The package/class/method organization is as follows:

- Base package: `eu.TRIANGLE_project.appinstr`
 - Use case package: `<use_case>`
 - Feature class: `<feature>Measurements`
 - Measurement method: `<measurement>`

The measurement methods are all static. There is no initialization involved in the usage of the Library. Therefore, they can be called from anywhere in the code of an Android application.

17.2.1 Custom measurements

To provide custom measurements, the Android instrumentation library provides a `CustomMeasurement` class in the `eu.TRIANGLE_project.appinstr.custom` package, with a set of overloaded methods called `custom`. These methods have two or three arguments: the feature and measurement names, and optionally one measurement argument.

17.3 Example

This section uses fictional names for features and measurements, to show an example of the use of the Android Library.

Let “social media” be a use case with the following features: “post picture” and “post video”. For each of these features, there are four measurements: “picture/video size”, “picture/video upload start”, “picture/video upload end”, and “upload success”. The instrumentation library will have two classes with four methods each, as follows:

- `eu.TRIANGLE_project.appinstr.socialmedia`
 - `public static class PostPictureMeasurements`
 - `public static void pictureSize(int size)`
 - `public static void pictureUploadStart()`
 - `public static void pictureUploadEnd()`
 - `public static void uploadSuccess(boolean success)`
 - `public static class PostVideoMeasurements`
 - `public static void videoSize(int size)`
 - `public static void videoUploadStart()`
 - `public static void videoUploadEnd()`
 - `public static void uploadSuccess(boolean success)`



Let us assume that a particular app performs a picture upload inside an uploadPicture method. The measurement methods could be called inside that method, where the required values are known. In particular, those related to events (i.e. “picture upload start/end”) should be placed as close to the actual place where that event happens, to provide precise measurements.

```
import eu.TRIANGLE_project.appinstr.socialmedia.PostPictureMeasurements;

public class MediaUploader {
    public void uploadPicture(Picture picture) {
        PostPictureMeasurements.pictureSize(picture.getSize());

        PostPictureMeasurements.pictureUploadStart();
        // Perform actual upload
        boolean uploadResult = doUploadPicture(picture);
        PostPictureMeasurements.pictureUploadEnd();

        PostPictureMeasurements.uploadSuccess(uploadResult);
    }
}
```

In addition, the app developer may want to provide a custom measurement with some metadata from the picture. The following snippet adds a couple of custom measurements to the previous one:

```
import eu.TRIANGLE_project.appinstr.socialmedia.PostPictureMeasurements;
import eu.TRIANGLE_project.appinstr.custom.CustomMeasurements;

public class MediaUploader {
    public void uploadPicture(Picture picture) {
        PostPictureMeasurements.pictureSize(picture.getSize());
        CustomMeasurements.custom("picture_metadata", "exif",
            picture.getExif());
        CustomMeasurements.custom("picture_metadata", "md5",
            picture.getMd5());

        PostPictureMeasurements.pictureUploadStart();
        // Perform actual upload
        boolean uploadResult = doUploadPicture(picture);
        PostPictureMeasurements.pictureUploadEnd();

        PostPictureMeasurements.uploadSuccess(uploadResult);
    }
}
```

An Android Studio project is provided to the experimenters with a minimal example of how to log custom measurements from an Android application. This application has been generated from one of the Android Studio project templates. When FAB (floating action button) is pressed on the main and only activity, four custom measurements are logged.

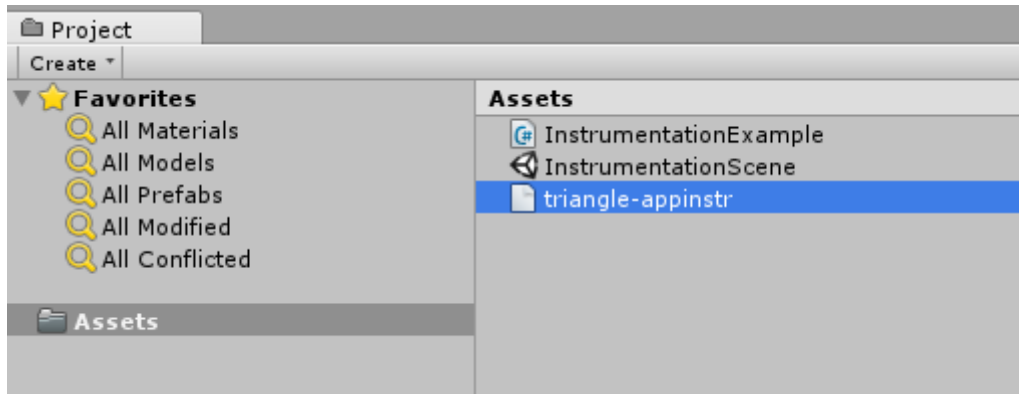
17.4 Using the Android Instrumentation Library from Unity

The Android Instrumentation library can be used in Unity apps for Android.



17.4.1 Including in a Unity project

Copy the AAR library file to a subfolder of the project's **Assets** folder (At the time of writing, drag-and-drop is not supported for AAR libraries). The library will be automatically detected.



It is important to note that the library cannot be used while running the application inside the Unity editor, since an Android environment is required. In the editor, the following exception will be thrown:

```
Exception: JNI: Init'd AndroidJavaClass with null ptr!  
UnityEngine.AndroidJavaClass..ctor (IntPtr jclass) (at C:/buildslave/unity/build/Runtime/Export/AndroidJavaImpl.cs:562)
```

The instrumentation library will work while using the Android emulator or a device.

If your Unity project targets other platforms besides Android, you can use [platform dependent compilation](#).

17.4.2 Usage

In order to use the measurement methods of the instrumentation library, you must first obtain a reference to the required library class using `AndroidJavaClass`. For instance, the following instantiation will return a reference to the `PostPictureMeasurement` class:

```
AndroidJavaClass postPictureMeasurements = new AndroidJavaClass(  
    "eu.TRIANGLE_project.appinstr.socialmedia.PostPictureMeasurements");
```

Using the `CallStatic` method on the `AndroidJavaClass`, the measurement methods of the instrumentation library can be called from the code of the Unity application. For instance, the `pictureSize` method can be called as:

```
postPictureMeasurements.CallStatic("pictureSize", new object[] {  
    picture.GetSize() });
```

It is important to note that you must make sure that the arguments to the method are cast to the appropriate type. For instance, if you want to call a method that has a double argument, but you have a float variable, you must explicitly cast that variable to double.

The included Unity project shows a minimal example of how to log custom measurements from a Unity application.

17.5 Retrieving messages

This information is internal and subject to change.

Measurements are written to Android's logcat with the `TRIANGLEInstr` tag and `INFO` priority.



17.6 Measurements format

This information is internal and subject to change.

Internally, the measurements library uses messages with a textual representation to pass the measurements from the application to the testbed orchestration. This section describes the current format of these messages.

17.6.1 Message format

The messages produced from an instrumentation library should have the following format:

```
<timestamp>\t<use_case>\t<feature>\t<measurement>[\t<value>]
```

Note that \t denotes a tab character, and that the part surrounded by square brackets is optional. The variables used in the format are:

- <timestamp>: the timestamp from the message
 - The timestamp is in the UTC timezone
 - The timestamp is formatted using the following ISO 8601 representation: <date>T<time>, where
 - <date>=YYYY-MM-DD, with YYYY, MM, DD as zero-padded year, month and day, respectively
 - <time>=hh:mm:ss.sss, with hh, mm, ss, sss as zero-padded hour, minute, second, and millisecond, respectively
- <use_case>: the id of the use case
- <feature>: the id of the feature
- <measurement>: the id of the measurement
- <value>: the actual value or values reported for the measurement
 - The actual format of the value depends on the type used in the measurement

17.6.1.1 Value format

A measurement may have zero or more arguments, which together form the value of the measurement. If a measurement has more than one argument, then <value> contains a tab-separated list of each of the argument values.

The format of each argument depends on its type. The following types are supported as arguments of a measurement:

- boolean: true and false values are formatted as true and false, respectively
- int: integers are formatted using base 10
- double: floating point values are formatted using an IEEE 754 compatible format
- string: strings are placed between double quotes ("), e.g. "Hello, World!"
 - quotes inside strings must be escaped with \, e.g. "Hello, \"World\"!"
 - new-line characters should be escaped as well, as \n and \r, e.g. "line1\nline2"