Noname manuscript No.
(will be inserted by the editor)

Model-based testing of apps in real network scenarios

Laura Panizo · Almudena Díaz · Bruno García

Received: date / Accepted: date

Abstract Traditional testing methods for mobile apps focus on detecting execution errors. However, the evolution of mobile networks towards 5G will require additional support for app developers to also ensure good performance and user-experience. Manual testing in a number of scenarios is not enough to satisfy the expectations of the apps' end users. This paper presents the testing framework developed in the TRIANGLE project[1], which integrates a complete mobile network testbed to test, benchmark and certify mobile apps. In this paper, we focus on a recent extension of the TRIANGLE framework that uses *model-based testing* based on *model checking* to support the automatic generation of user interactions. We introduce the complete testing framework and the basis of the model-based extension. Finally, we use the testing framework to evaluate the performance of the ExoPlayer app in different network scenarios. ExoPlayer is a video streaming app for Android that implements different adaptive streaming protocols.

Keywords Model-based testing, mobile network testbed, model checking

This is a pre-print version of the journal paper version. Panizo, L., Díaz, A. and García, B. Model-based testing of apps in real network scenarios. Int J Softw Tools Technol Transfer 22, 105–114 (2020). https://doi.org/10.1007/s10009-019-00518-2

This work is funded by the European Union Horizon 2020 research and innovation programme, grant agreement No 688712 (TRIANGLE project) and 815178 (5GENESIS project).

L. Panizo, A. Díaz and B. García at Universidad de Málaga, Andalucía Tech, Dept. de Ciencias de la Computación E-mail: [laurapanizo,adz,bgg]@uma.es

[1] https://www.triangle-project.eu/

1 Introduction

The TRIANGLE testbed [6] is devoted to the testing of mobile applications. It is the result of the efforts of the EU H2020 funded TRIANGLE project.

The testbed provides an end-to-end testing platform that emulates a complete cellular network including commercial off-the-shelf mobile devices. The objective of this platform, and the approach adopted for the execution of the tests, is to bring to the application level the traditional certification methodology followed in the domain of mobile protocol stack certification.

The certification of the mobile terminals not only ensures that the terminals work according to standards but also that they are interoperable with the other elements of a mobile network. The certification of apps must also ensure their efficiency in different domains; that the app makes efficient use of network and device resources (transmitted and received data, CPU, memory, etc.), battery consumption, as well as satisfying the required user experience.

Clearly, the complexity of app testing is huge due to the wide variety of orthogonal domains, also requiring the creation of a certification methodology for applications. The certification of mobile equipment is based on the execution of a set of test cases specified primarily by the 3GPP (3rd Generation Partnership Project) consortium. The purpose of the 3GPP is to prepare, approve and maintain globally applicable Technical Specifications and Technical Reports for Mobile Systems to be transposed by the telecommunication standard bodies into standards.

Within the TRIANGLE consortium, the international company DEKRA Testing and Certification

S.A.U.[2], which is a member of different standardization bodies, leads the task of specifying a completed set of test cases for the different domains mentioned before: network resources usage, device resources usage, energy consumption, power consumption and user experience. Mobile app testing is not a new topic. There are several companies [24,19] that provide test services for mobile applications. In addition, there are initiatives coming from the standardization industry. The 3GPP technical report TS 37.901 [1] is the only recommendation from 3GPP that includes the definition of test cases at the application level. This recommendation defines the execution of FTP (File Transport Protocol) tests to measure the IP throughput reached in different network scenarios. The Cellular Telecommunications Industry Association (CTIA) has also specified a set of test cases for determining the expected battery life of smartphones [7]. In this case, the specification only considers a set of the most representative applications: gaming, audio and video streaming, voice call, web browsing, email and SMS messages, and is focused on battery consumption. Finally in [10], The Global System for Mobile Communications Association (GSMA) focuses on the evaluation of the performance of the devices when running a set of reference applications such as web browsing, email and SMS.

Compared with the approaches introduced above, the TRIANGLE testing methodology includes the following novelties:

- It is not restricted to the evaluation of the app's performance in terms of device resources. To this end, the testing environment includes the emulation of a complete mobile network including mobility scenarios and different resource allocation policies.
- It is a standard-oriented approach based on defining sets of applicable test cases.
- It is an open methodology based on publicly available definitions, so it can be adopted by third parties. This also opens the door to the global comparison of the results obtained following the approach.
- The methodology can be applied to any application, thanks to the instrumentation library developed in the TRIANGLE project, which allows the insertion of measurement points into the source code of the application under test.

It is worth mentioning that the TRIANGLE project has contributed to the white paper "Definition of the testing framework for the NGMN 5G pre-commercial networks trials" [20] delivered by the NGMN Alliance, so that the outputs of the project reach the standardization bodies. The TRIANGLE project is also in contact with the General Certification Forum (GCF).

In the context of the TRIANGLE project, the model-based testing (MBT) technique is used to automatically generate the set of app user flows (sequence of user actions) that stimulates the application under test. The use of MBT improves the test coverage ensuring that the features of the apps have been tested extensively and exhaustively.

In this paper, we present the methodology used in [21] to automatically generate the app user flows to achieve the complete automation of the application testing process. This methodology is currently integrated in the TRIANGLE testbed, and it has been used to evaluate the implementation of three different adaptive video streaming protocols included in the ExoPlayer app: MPEG's Dynamic Adaptive Streaming over HTTP(DASH) [12], Apple's HTTP Live Streaming (HLS) [23], and Microsoft Silverlight Smooth Streaming (MSS) [18].

The paper is organized as follows. Section 2 introduces the main components of the TRIANGLE testbed. Section 3 describes the extension of the testbed using model-based testing techniques. Section 4 presents the testing of the ExoPlayer app with the TRIANGLE testbed and the model-based testing extension. Section 5 reviews some related work. Finally, Section 6 summarizes the conclusions and future work.

2 The TRIANGLE Testing Framework

As mentioned earlier, the TRIANGLE testbed [6] is devoted to the testing and benchmarking of mobile applications and devices. Figure 1 shows an overview of the main functional blocks of the testbed architecture. The blue box contains all the hardware components integrated into the testbed. The UXM is a base station emulator also used in state of the art conformance and research testing of mobile devices. This equipment implements the full stack of 2G/3G/4G radio access network protocols.

In order to accurately control the radio conditions configured at the base station emulator, the radio antenna is connected to the mobile device through cables. This is the standardized approach followed in protocol and Radio Frequency testing by 3GPP. In addition, to properly analyze power consumption, the device is powered directly by a power analyzer. Finally, the testbed also integrates a commercial Evolved Packet Core (EPC) that includes the main elements of a standard core network and local application servers such as streaming or VoIP servers.

[2] https://wireless.dekra-product-safety.com/

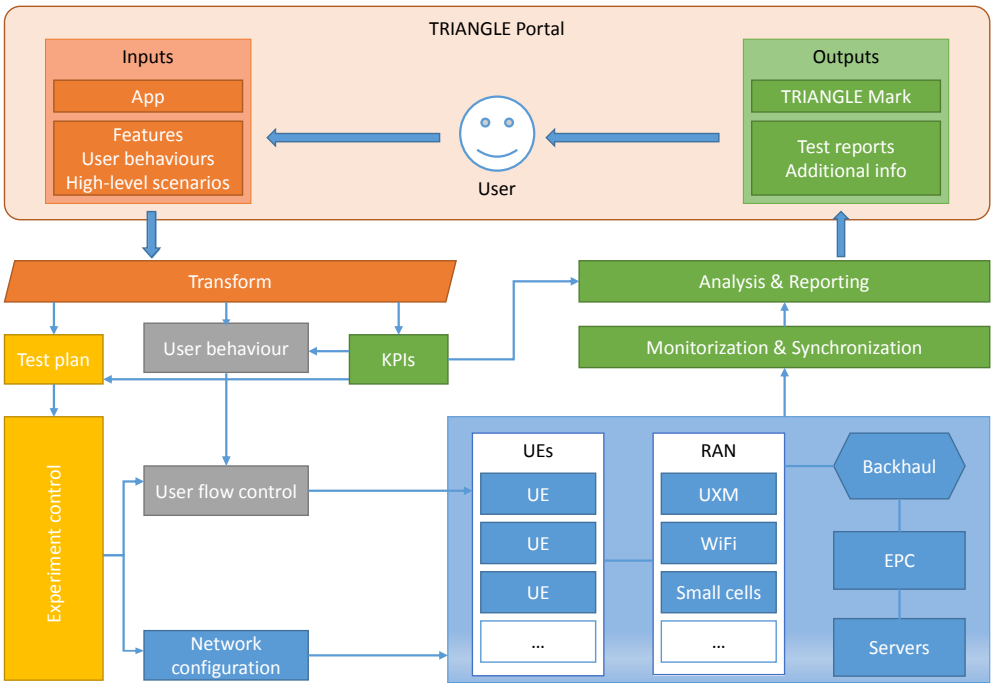

Fig. 1 TRIANGLE testing framework overview

For all intents, the UXM is seen as a base station by the devices under test, with the singularity that all the parameters of the communications stack can be modified. In addition, the UXM includes a channel emulator that enables the configuration and impairing of mobile radio conditions. Based on these features, the TRIANGLE project has developed a set of high level scenarios covering urban, sub-urban and high-speed scenarios. Urban scenarios are classified into pedestrian, driving and internet-café scenarios, while sub-urban scenarios are classified into Festival, Stadium and Shopping Mall. The complete description of the scenarios and the list of test cases are published respectively in [26] and [27]. These scenarios are used to test the applications in a huge variety of network conditions.

On top of the hardware components, the TRIANGLE testbed has built an automation framework that enables the configuration of all the network components, and recording and replaying apps user flows. The purpose of the framework is to enable the autonomous and unattended execution of all the test cases. This automation framework can be accessed by expert users who have the required know-how to configure the low level parameters directly exposed by the components of the network. For those users who are not familiar with the technical details, the testbed provides a Web-based access called the *Portal*, whose main purpose is to prepare and run the tests, and later review the results. It provides an intuitive interface for the definition and execution of the *testing campaigns*, hiding unnecessary complexity. Testing campaigns are based on the execution of the *test cases* specified in the TRIANGLE project for app testing. A test case defines the configuration of the network scenarios, the app user flow (sequence of actions) that will be used to activate the feature under test, and finally the measurements that have to be collected, which are determined by the Key Performance Indicators (KPIs) associated to the features under test.

As commented before, the project has also developed an instrumentation library in order to collect measurements related to the internal performance of the app under test. The app's source code is instrumented with this library to push measurement points into logcat, the Android logging system, and correlate these points with radio and power measurements.

Finally, an integral part of testing apps is automating their execution, i.e., simulating the user's interactions with the app. Quamotion automation tools[3] provide the means to record sequences of user actions, and then replay them on a testbed device. The user has to record and upload, through the Portal, the app user flows specified in each test case. It is at this point where model-based testing comes into action. The first approach in the project was to manually record these app user flows. However, this approach has two main drawbacks. The first one is that it can be a very time consuming task, and the second one is that there may be

[3] http://quamotion.mobi/Mwc

more than one user flow that meets the actions specified in the test case.

The next section presents the model-based testing approach, which is based on model checking, to automatically generate the app user flows in order to achieve the complete automatization of the app testing process and provide a higher coverage of the app user flows which meet the specifications.

3 Model-based testing

The test cases specified by the TRIANGLE project define the testing of generic app features, such as download content or playback media files. These test cases also specify the measurements that have to be collected during the test in order to compute the Key Performance Indicators (KPIs), which will enable the evaluation of the features' performance. In the simplest case, the app developer provides an app user flow that activates the feature under test. For instance, if the app under test playbacks live streaming, the app user flow has to start and end the playback.

The TRIANGLE testbed integrates *model-based testing* techniques to support the automatic generation of a set of app user flows. Model-based testing [5] covers a set of techniques for automating the generation of test cases based on the formal description of the system under test (SUT). In [22], we presented a preliminary model-based method to guide the generation of app user flows, based on an app model, which describes the possible interactions of the user and the app Graphical User Interface (GUI), and the verification of this model against a property with the model checker Spin [11]. Each counterexample returned by Spin represents an app user flow that activates the feature under test, and can thus be used to compute any given KPI. The property, what we call *app user flow requirements*, is the key to only obtain a small, useful set of app user flows. In the current work, we have used the same modelling language to describe the app, but we have improved the definition of app user flow requirements by means of an xml-based specification language. In the rest of the section, we provide more details about the modelling and the specification language and how the app user flows are obtained in the extended version of the TRIANGLE testing framework.

3.1 Modelling language for apps

The app model is described with a language based on nested state machines [9] that are able to capture the user's interaction with the app, and the intrinsic behaviour of mobile operating systems. An *app state machine* is composed of one or more *activity state machines*, which correspond to the different activities (screens) in the app. In addition, an activity state machine can contain one or more *state machines* with states and edges defining the user's behaviour.

Figure 2 shows a simplified model of the ExoPlayer app, an Android streaming video player that supports different adaptive streaming protocols. The app is divided into two activities: *SampleChooserActivity* shows the list of videos that can be played, and *PlayerActivity* shows the playback controls.

The edges of a state machine represent user actions, such as tapping a button (e.g. *play_pause*), that should be executed when traversing the edge. The initial and final states of a state machine are represented respectively by the closed and bordered circles. Connection states are used to transit between *state machines*. These states are represented with circles marked with a C and two outgoing transitions, one targeted to a state machine, and another to a state of the source state machine (dashed line). When a connection state is reached, the execution continues with the initial state of the state machine referenced. When the target state machine finishes, that is, when it reaches an end state, the execution comes back to the returning state. In Figure 2, it is possible to transit from *SampleChooserActivity_0* to the *PlayerActivity_0* state machine using connection states. When the *PlayerActivity_0* state machine reaches its end state, the execution will come back to a state of *SampleChooserActivity_0*. The specific returning state will depend on the connection state that fired the transition to *PlayerActivity_0*. In addition, observe that when the app is launched (initial state of the *ExoPlayer app state machine*) only the initial state of the *SampleChooserActivity* is accessible. Thus, *PlayerActivity_0* can only be reached by a transition from a connection state. This way, the modelling language describes which activity is enabled when the app is launched, and how the other activities are enabled.

3.2 App user flow requirements specification

In [22], the app user flow requirements define the state machines and states that have to be visited (or not), as well as how many times different events can be fired. An app user flow requirement is defined based on invariants and constraints. The invariants are conditions that must hold in the complete app user flow, while the constraints have to be satisfied at some moment. Figure 3 shows an app user flow represented as an automaton. If an execution path of the app model can go from

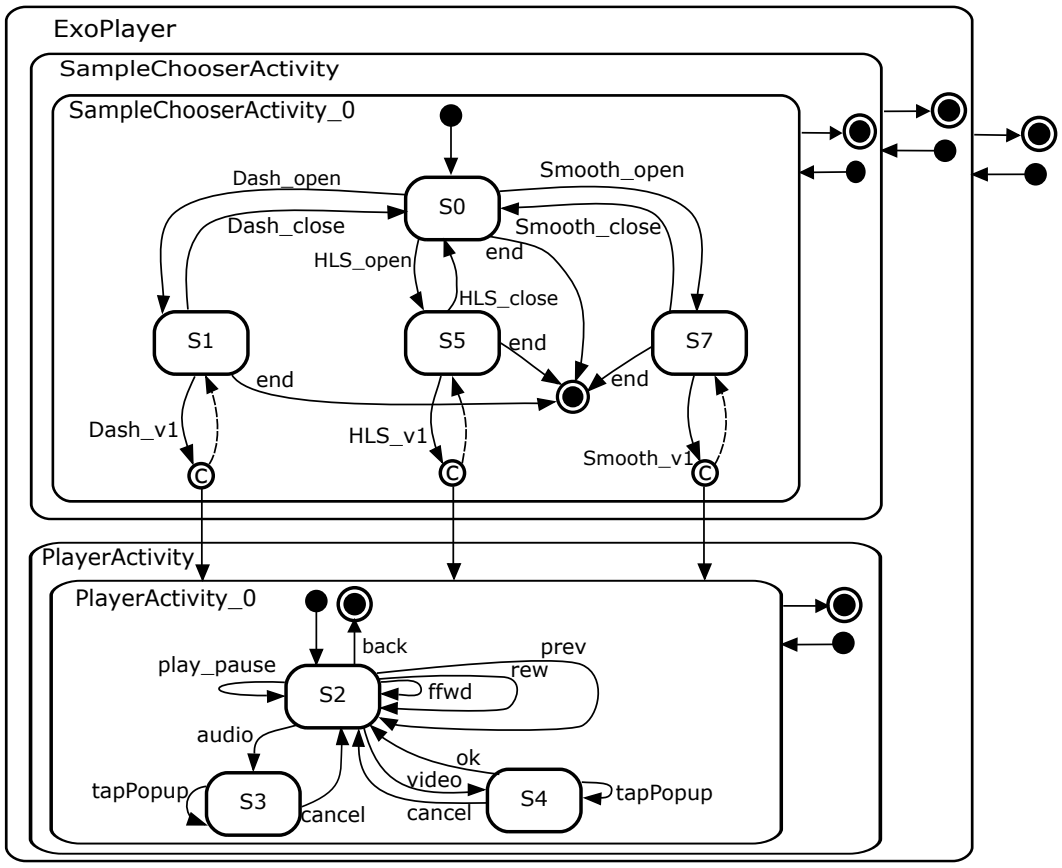

Fig. 2 Exoplayer model with nested state machines

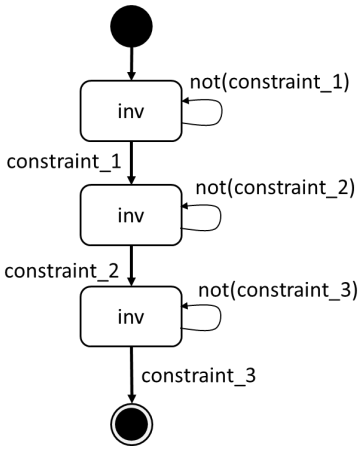

Fig. 3 App user flow requirement described as automaton

the initial to the final states of the requirement without blocking, the path satisfies the requirement and, therefore, is a valid app user flow. The requirement automaton blocks if a state does not satisfy the invariant or if there is no transition enabled.

In [22], the app user flow requirements are directly described with Promela code. More specifically, the app user flow requirements are described as a *never claim*, which is a special Promela process that guides and prunes the exploration of the app.

In previous work [21], we propose an xml-based language to describe the app user flow requirements. This notation is more intuitive for an app developer, and the requirements are automatically transformed into the Promela *never claim*.

In this work, we have extended the app user flow requirements and their xml-based description to support simple temporal constraints. This extension is a step further to fully automate the generation and execution of the test cases. The objective is to automatically produce app user flows that, for example, reproduce a video during a time interval, or wait some seconds before executing the next action (tapping, scrolling, etc.). In [21], the app model can include the time elapsed between a transition and the next one. Moving the temporal information from the app model to the app user flow requirement allows us to reuse the model without modifications to generate app user flows that include different temporal constraints.

Listing 1 shows an example related to the ExoPlayer app model. This example defines an invariant

Listing 1 Example of requirements in XML format

```
<appUserFlowRequirement xmlns="appuserflowrequirement" name
    ="DemoPlayer_never_1">
  <invariants>
    <event name="play_pause" max="2" time ="50"/>
  </invariants>
  <sequence>
    <constraint type="simple">
      <event name="Dash_video_1" min="1" max="1"/>
    </constraint>
    <constraint type="simple">
      <event name="play_pause" min="2" max="2"/>
    </constraint>
    <constraint type="simple">
      <state name="end" view="SampleChooserActivity"
        statemachine="SampleChooserActivity_0" visit="true"
            />
    </constraint>
  </sequence>
</appUserFlowRequirement>
```

that forces at most two play_pause events in the complete app user flow. This way, it is possible to model a play→pause→resume behaviour sequence. In addition, the invariant states that after any play_pause the app user flow will not execute more actions until 50 seconds have passed. The requirement also specifies a sequence of events and the states that have to be visited (relative order). Any other events or states can be fired and visited if the invariant is still satisfied.

3.3 Model-based testing campaigns

To take advantage of this new feature, the app developer has to define a new type of campaign called *model-based*. The inputs for this type of campaign are the app's binaries (in Android the apk file), the app model and the app user flow requirement. In addition, the app developer has to select one of the pre-defined high level network scenarios, and configure the minimum and maximum length of the app user flows generated, and the number of executions of the selected network scenario.

Figure 2 shows a graphical representation of the app model, but the model uploaded is described in an xml format, similar to the app user flow requirements. Although the app model can be manually generated, we have also developed a prototype tool that can automatically extract an app model from the app binaries. The model generator tool executes the app in a device and explores the app GUI, that is, performs different user actions (taps, scroll, etc.) in the different GUI elements to track how the GUI components change and identify, in this way, the different app states. Due to space limitations, we do not provide details about automatic model generation. It is worth mentioning that we have also developed a simple model editor to customize the app model automatically produced by the model generator tool.

The model-based campaign includes a first phase of app user flow generation. The TRIANGLE testbed automatically translates the app model into a Promela specification and the requirements are translated into a *never claim*. Then, Spin executes the synchronous product of the system model and the never claim; that is, after executing a system model statement, it executes a never claim statement. If the requirement automaton is blocked (the invariant is not satisfied or all transitions are disabled), the never claim cannot execute any statement, and the Spin exploration algorithm backtracks to explore a different execution path. If the requirement automaton reaches its final state, the current execution path is a valid app user flow that is recorded in the format of the Quamotion automation tool, in order to automate the app user flow execution on the device. For example, the analysis of the ExoPlayer model (see Figure 2 and Listing 1) produces 86 app user flows with at most 10 user actions. These app user flows can be used to evaluate a given KPI.

If the first phase generates at least one app user flow, the testbed can proceed with the execution of the test in the selected high-level scenario. In each execution of the scenario, the testbed randomly picks one of the app user flows. If there is only one app user flow satisfying the requirements, it will be used in all the executions.

4 Case Study: ExoPlayer

This section presents a case study to show the complete process of evaluating the performance of (or benchmarking) a mobile app with the extended TRIANGLE testbed. The application under test is the ExoPlayer app, an open source video streaming player that implements different adaptive video streaming protocols over HTTP. In this section, we evaluate video on demand streaming in different high level network scenarios.

4.1 High level scenarios

- Ideal conditions.
- Internet Café at busy hours: this scenario emulates a Café in a densely populated city at rush hour. Users are mostly static, and connect to the Internet with multiple radio access technologies, specially WiFi and cellular networks. The cellular network in dense urban areas is characterized by a combination of macro and small cell deployments, with a reduced inter-site distance. In addition, since the scenario

emulates rush hour, the network is heavy loaded and the network resources are scarce.
- Driving in traffic jam: this scenario emulates a traffic jam in an urban area. Thus, there is a high density of vehicles with a very low speed. In this case, users connect to the Internet with the on board entertainment system and make an intensive use of the network.
- High speed train with direct passenger connection: this scenario emulates a high speed train (350 km/h) crossing a rural area with sparse macro cells. Each user is dealt with individually by the network. Thus, the massive changes of cells where the users are connected (handover) directly impact the network performance and the quality of experience.

4.2 Adaptive video streaming protocols

In general, adaptive bitrate streaming protocols are based on encoding the source video (live or on demand) and other data (e.g. subtitles or metadata) into file segments called fragments. A fragment stores between 2 and 10 seconds of video, depending on the protocol. The same content may be available in different fragments with different bitrate, video resolution or codecs. Thus, different fragments contain the same video source with different features. The fragments are available in a HTTP server, and the clients have to request the fragments and reproduce them contiguously as they are downloaded. Since the streaming protocol is adaptive, the client can compute, for instance, the available bandwidth and request fragments with a bitrate that is better suited for the current network conditions. The client can take into account other parameters when requesting the fragments, such as the CPU load or the maximum resolution it can playback. We have tested the implementation of the following adaptive streaming protocols included in ExoPlayer app:

HTTP Live Streaming [23] (also known as HLS) is Apple's HTTP-based media streaming protocol. The media source is segmented into MPEG-2 fragments. The list of available streams, encoded at different bit rates, is sent to the client using a playlist file in M3U8 [23] format, which includes the URL of the fragments and other metadata as comments. The playlist is downloaded every time a fragment is played; thus, in order to reduce the number of playlists downloaded, the fragment's recommended length is 10 seconds.

Smooth Streaming [18] is an IIS Media Services extension that enables adaptive streaming of media to clients over HTTP. This protocol segments the media into a sequence of MPEG-4 fragments. Smooth Streaming does not send the playlist to a separate file. Instead, it embeds on each fragment information in order for the client to construct the URL of the next fragments. Thus, the recommended fragment size is between 2 and 4 seconds, smaller than in HLS.

Dynamic Adaptive Streaming over HTTP [12], also known as DASH or MPEG-DASH, is an open standard. It is the most complete but also complex adaptive streaming protocol; thus it is difficult to implement and different implementations can be incompatible. It combines features from HLS and Smooth; for instance, it supports repeated download of playlist files but also generation of URLs by the client using a template. In addition, multiple fragments file formats are supported. The recommended fragment size is around 2 seconds. Thus, Smooth and DASH adapt faster to channel changes.

4.3 Evaluation

Testing or benchmarking an app in the TRIANGLE testbed is a simple and automatic task that only requires the following steps:

1. Instrumentation of the source code. The TRIANGLE testbed does not require the app source code, just the binaries. However, the app must include *measurement points* in order to correctly compute the measurements and the KPIs. For example, the ExoPlayer app includes measurement points to detect the playback start, pause and end.

2. App model extraction. Given the app's instrumented binaries, the following step is the generation of the app model. The model is the key to automatically generate the app user flows used during the test. Thus, the model has to describe the user's interactions with the app. In order to ease this task, we have developed prototype tools for generating and editing the app model. The model generator executes the app in a real device and automatically commands different user's actions (click, scroll, etc.). It explores with a depth-first search strategy how the user's actions modify the user interface components, and produces the app model based on nested state machines, as described in [9]. The model generator can be configured to explore just some types of user interface components, a restricted set of user actions, or until a given depth is reached. This way, we can bind the time and memory required to produce the model as well as its accuracy.

3. Definition and execution of a model-based testing campaign. In the next stage, the app developer interacts with the TRIANGLE web portal in order to upload the app binaries and configure the testing campaign. For each campaign, the app developer first selects the device and the high level network scenario. In addition, if the campaign is model-based, it is also necessary to upload the app model and the app user flow requirement files, as well as to define the minimum and maximum length of the app user flows and the number of tests executed in the campaign.

To evaluate the 3 adaptive streaming protocols, we have defined 3 different app user flow requirements, each one pursuing the video playback with a different protocol. SPIN analyses the ExoPlayer model, shown in Figure 2, in less than 2.5 seconds, and produces a pool of 86 app user flows of length 10 (at most) per requirement.

In total, we have created 12 model-based campaigns. The evaluation of each streaming protocol consists of 4 campaigns, each one with a different high-level scenario. Since ExoPlayer is a content distribution app, each campaign consists of two different test cases: 1) *non interactive playback* and 2) *play pause*, and each test case is executed 5 times using different app user flows.

4. Interpretation of the campaign results. When a model-based testing campaign finishes, the portal provides a complete report with the computed KPIs and the TRIANGLE mark, which currently is the average of three domains: device resource usage, energy consumption and user experience.

Figures 4 to 7 show the mark obtained in each domain for each protocol and scenario and the final TRIANGLE mark. These values are normalized into a standard 1-to-5 scale, as typically used in MOS (Mean Opinion Score) [14]. The basis of the mark per domain and the TRIANGLE mark computation is presented in [8].

The metrics provided by TRIANGLE can be useful to make decisions regarding which protocol implementation to use. For instance, the results show that DASH has the highest TRIANGLE mark in 3 of the 4 scenarios, which is always above 4. Regarding the user experience domain, DASH obtains a mark above 4 in all scenarios with small variations, while the other two protocols present marks varying over a larger range of values. In some scenarios the HLS and Smooth protocols present a better or similar performance but in others they are clearly worse. Thus, it seems that DASH adapts better to a wide range of network conditions.

Finally, it is worth mentioning that our objective in this paper is not to state which is the best adaptive streaming protocol; we just show how the TRIANGLE testbed can help app developers to evaluate their apps in a systematic way, taking into account a wide variety of user interactions with the app, and different network scenarios to detect under-performance issues.

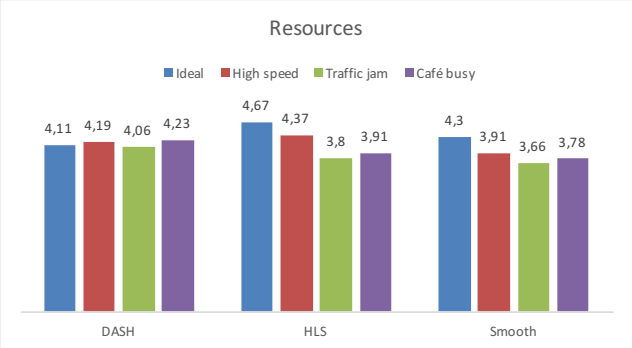

Fig. 4 Resource usage in different network scenarios

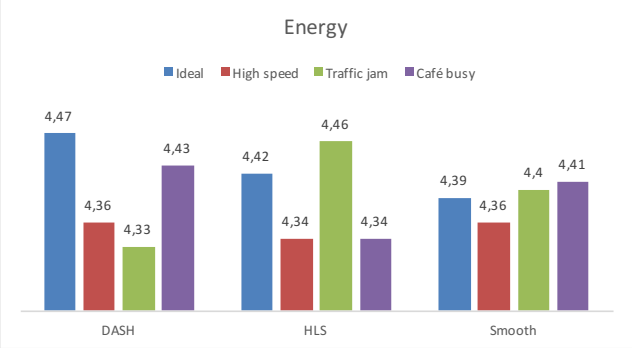

Fig. 5 Energy consumption in different network scenarios

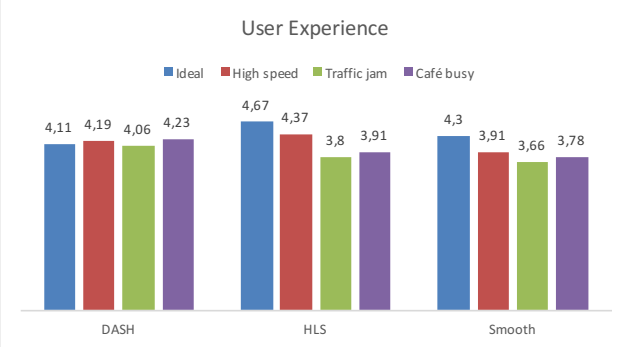

Fig. 6 User experience in different network scenarios

5 Related Work

The automatic testing of mobile applications is a very interesting topic that has been addressed with very different approaches, such as model-based testing [3, 4, 29] or code analysis [15]. In addition, testing can be applied

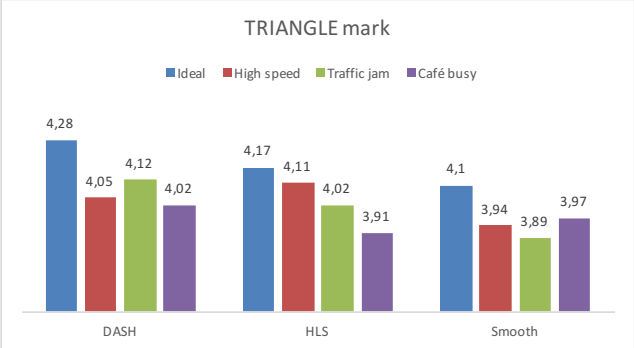

Fig. 7 TRIANGLE mark in different network scenarios

to verify functional properties [29,3] or non-functional properties, such as performance evaluation [17,25].

In [29], the authors present an approach to automatically generate mobile app GUI models based on grey-box testing. First, the app source code is statically analyzed to determine the user events supported by the GUI. Then, this information is used to generate the app model by performing these events on the running app, using a customized version of depth-first search algorithm that takes advantage of the back button (present in most Android devices) to backtrack faster when possible. The tool ORBIT implements this approach.

MobiGuitar [3] is a framework for automatically testing GUI mobile apps. The GUI model extraction is based on a breadth-first traversal algorithm. The resulting model is a single state machine whose transitions represent the different user events. To reduce the number of states explored during the GUI traversal, two states are considered equivalent if the IDs and type properties of the GUI objects are equal. Test cases are generated using the model and the adequacy criterion. In addition, the authors propose a pairwise edge coverage criterion to reduce the number of generated test cases. Finally, MobiGuitar transforms the sequence of user events into JUnit [16] test cases that are executed.

Baek and Bae [4] propose a GUI testing framework that generate app models using a multi-level abstraction criterion. This criterion compares two states at different levels to determine if a state is new, existing or terminated. To produce fine grain models it compares at five levels: package, activity, widget, widget executability, and content. Moreover, the model generation process can be configured to consider only the first levels to produce coarser models. The GUI is explored using a breadth-first search strategy, and a graph of the GUI (the model) and the test cases are obtained simultaneously.

Our proposal has many commonalities with these approaches. Our app user flows are generated using an app model, which describes the user interaction with the app GUI. However, the main difference with the other papers is the aim of the testing process. The other proposals generate test cases to detect GUI bugs or functional errors. Our objective is to generate test cases that support the verification and evaluation of extra functional properties under different network scenarios.

Testing for app performance has been addressed with different techniques. PerfChecker [15] is a static code analyzer that detects common performance bug patterns. These patterns, which were defined based on the analysis of eight large-scale Android apps, can produce low responsiveness of the GUI, energy leaks and high memory consumption. Although these patterns are valuable, they do not consider external factors, such as the connectivity/network. In [28], the authors test poor responsiveness in Android apps using a model of the app GUI. Test cases are executed to cover the possible GUI states and transitions. Then, the tests are re-executed in a modified environment that artificially injects long delays in problematic operations, mainly operations which require network, mass storage or data base access. However, it is not clear how the delays are selected to simulate, for instance, different network scenarios.

Other approaches focus on testing or evaluating the apps' performance taking into account the different network scenarios [17,25]. These proposals usually include network emulators or simulators where the tests or experiments are executed, but they do not provide much information about how the tests are selected or generated. Finally, we recommend reading the work presented in [30,2] that discuss and compare other approaches.

6 Conclusions

In this paper, we have presented the TRIANGLE testbed, which provides an end-to-end mobile network environment that enables app developers to thoroughly test their applications in real network scenarios, including radio conditions, which are not under their control when running tests in live mobile deployments. One of the most important features of the testbed is that the network scenarios emulate realistic conditions, are totally repeatable and the experiments are reproducible. This ensures the validity of the test results.

The previous version of the testbed is able to rigorously test the application in numerous combinations of radio and network configurations. The integration of model-based testing techniques pursues this same accuracy to generate app user flows to stimulate applications

under test in different network scenarios. In particular, the integration of model-based testing techniques improves the usability and flexibility of the testbed in different ways:

- The testbed automatically produces a pool of app user flows that can activate the app features in different ways, improving test coverage.
- App user flows satisfying different requirements are generated with the same app model. If the testbed is extended with new test cases, the app model does not change, and only new requirements have to be defined.
- The app user flow requirements are described in a simple language that abstracts from the automation format of the app user flows, which is transparent to the app developer.

As future work, we would like to reduce human intervention during the manual definition of the app user flow requirement. A possible solution is to include in the TRIANGLE portal a library of generic app user flow requirements for each app use case (content distribution, gaming, etc.) and ask the app developer for an app model with specific annotations that allow us to automatically adapt these generic requirements to the app model. To facilitate the analysis of the results of the MBT campaigns, we would like to include a graphical comparison of the results obtained in the different network scenarios.

In addition, we want to extend the model-based approach to other mobile operating systems. The current app modelling language suits Android apps perfectly(e.g. the backstack mechanism), but we must study if the language can capture others, such as iOS apps.

Finally, the testbed and the methodology presented in this paper will be extended in the 5GENESIS project [13]. The objective of the 5GENESIS project is to establish a complete, open, evolved and distributed experimentation facility across Europe for assessing the added-value of 5G and validating the associated KPIs. In this context, we plan to use and extend the testbed and the case-based methodology. The fifth generation of mobile technology (5G) will rely on technologies such as Software Defined Networks (SDNs) and Virtual Network Functions (VNF), whose aim is to deploy very flexible networks based on software components that can be deployed in infrastructures with different locations and resources that have a direct impact in the KPIs. We think that model-based testing is a good approach to generate different VNF chaining and deployment options, similarly to the app user flows generated in this work. To this end, we have to define the interaction model between the VNFs and their *environment*, which is mainly the management and orchestration (MANO) entity that is in charge of dynamically deploying the VNFs in the network.

References

1. 3GPP: TR37.901:User Equipment (UE) application layer data throughput (Rel. 15). Tech. rep. (2018)
2. Amalfitano, D., Amatucci, N., Memon, A.M., Tramontana, P., Fasolino, A.R.: A general framework for comparing automatic testing techniques of Android mobile apps. Journal of Systems and Software **125**, 322 – 343 (2017). DOI 10.1016/j.jss.2016.12.017
3. Amalfitano, D., Fasolino, A.R., Tramontana, P., Ta, B.D., Memon, A.M.: MobiGUITAR: Automated Model-Based Testing of Mobile Apps. IEEE Software **32**(5), 53–59 (2015). DOI 10.1109/MS.2014.55
4. Baek, Y.M., Bae, D.H.: Automated Model-based Android GUI Testing Using Multi-level GUI Comparison Criteria. In: Proc. of the 31st IEEE/ACM Int. Conference on Automated Software Engineering, ASE 2016, pp. 238–249. ACM (2016). DOI 10.1145/2970276.2970313
5. Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A.: Model-Based Testing of Reactive Systems: Advanced Lectures. Springer-Verlag, Berlin, Heidelberg (2005)
6. Cattoni, A.F., Corrales-Madueño, G., Dieudonne, M., Merino, P., Díaz-Zayas, A., Salmerón, A., Carlier, F., Saint-Germain, B., Morris, D., Figueiredo, R., Caffrey, J., Baños, J., Cárdenas, C., Roche, N., Moore, A.: An end-to-end testing ecosystem for 5G. In: European Conference on Networks and Communications (EuCNC 2016), pp. 307–312 (2016). DOI 10.1109/EuCNC.2016.7561053
7. Cellular Telecommunications Industry Association (CTIA): Battery Life Test Plan v.1.2. Tech. rep. (2018). URL https://api.ctia.org/wp-content/uploads/2018/04/CTIA-Battery-Life-Test-Plan-Version-1.2.pdf
8. Díaz, A., Panizo, L., Baños, J., Cárdenas, C., Dieudonne, M.: QoE Evaluation: The TRIANGLE Testbed Approach. Wireless Communications and Mobile Computing p. 12 (2018). DOI 10.1155/2018/6202854
9. Espada, A.R., Gallardo, M.M., Salmerón, A., Merino, P.: Performance Analysis of Spotify® for Android with Model Based Testing. Mobile Information Systems **2017**, 14 (2017). DOI 10.1155/2017/2012696
10. Global System for Mobile Communications Association (GSMA): Smartphone Performance Test Case Guideline v3.0. Tech. rep. (2017). URL https://www.gsma.com/newsroom/wp-content/uploads//TS.29_v3.0.pdf
11. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional (2003)
12. ISO/IEC: Information technology — Dynamic adaptive streaming over HTTP (DASH). http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip (2014)
13. Koumaras, H., Tsolkas, D., Gardikis, G., Merino-Gómez, P., Frascolla, V., Triantafyllopoulou, D., Emmelmann, M., Koumaras, V., Garcia-Osma, M.L., Munaretto, D., Atxutegi, E., de Puga, J.S., Alay, O., Brunstrom, A., Bosneag, A.M.C.: 5GENESIS: The Genesis of a flexible 5G Facility. In: IEEE Int. Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD-2018) (2018)

14. Kozamernik, F., Steinmann, V., Sunna, P., Wyckens, E.: SAMVIQ—A New EBU Methodology for Video Quality Evaluations in Multimedia. SMPTE Motion Imaging Journal **114**(4), 152–160 (2005). DOI 10.5594/J11535
15. Liu, Y., Xu, C., Cheung, S.: Characterizing and detecting performance bugs for smartphone applications. In: Proc. of the 36th Inte. Conference on Software Engineering, ICSE 2014, pp. 1013–1024. ACM (2014). DOI 10.1145/2568225.2568229
16. Massol, V., Husted, T.: JUnit in Action. Manning Publications Co., Greenwich, CT, USA (2003)
17. Mehmood, M.A., Wundsam, A., Uhlig, S., Levin, D., Sarrar, N., Feldmann, A.: QoE-Lab: Towards Evaluating Quality of Experience for Future Internet Conditions. In: T. Korakis, H. Li, P. Tran-Gia, H.S. Park (eds.) Testbeds and Research Infrastructure. Development of Networks and Communities, pp. 286–301. Springer Berlin Heidelberg (2012)
18. Microsoft Corporation: Smooth streaming protocol. https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SSTR/[MS-SSTR].pdf (2009)
19. NeoLoad by Neotys: The Load Testing Platform Accelerating DevOps. http://www.neotys.com/. Last accessed: 10/26/2018
20. NGMN Alliance: Definition of the Testing Framework for the NGMN 5G Pre-Commecial Networks Trails v1.0. Tech. rep. (2018). URL https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2018/180220_NGMN_PreCommTrials_Framework_definition_v1_0.pdf
21. Panizo, L., Díaz, A., García, B.: An Extension of TRIANGLE Testbed with Model-Based Testing. In: M.d.M. Gallardo, P. Merino (eds.) Proc. of the 25th Int. Symposium on Model Checking Software (SPIN2018), pp. 190–195. Springer International Publishing (2018)
22. Panizo, L., Salmerón, A., Gallardo, M.M., Merino, P.: Guided Test Case Generation for Mobile Apps in the TRIANGLE Project: Work in Progress. In: Proc. of the 24th International SPIN Symposium on Model Checking of Software, pp. 192–195. ACM (2017). DOI 10.1145/3092282.3092298
23. Pantos, R., May, W.: HTTP Live Streaming. RFC 8216 (2017). DOI 10.17487/RFC8216
24. Perfecto: The Cloud-based Platform for Continuous Testing in a DevOps Environment. https://www.perfecto.io/. Last accessed: 10/26/2018
25. Solera, M., Toril, M., Palomo, I., Gomez, G., Poncela, J.: A Testbed for Evaluating Video Streaming Services in LTE. Wireless Personal Communications **98**(3), 2753–2773 (2018). DOI 10.1007/s11277-017-4999-0
26. TRIANGLE project consortium: Deliverable D2.1: Initial report on the testing scenarios, requirements and use cases. Public (2016)
27. TRIANGLE project consortium: Deliverable D2.6: Final Test Scenario and Test Specifications. Public (2018)
28. Yang, S., Yan, D., Rountev, A.: Testing for poor responsiveness in Android applications. In: Proc.of the 1st Int. Workshop on the Engineering of Mobile-Enabled Systems (MOBS), pp. 1–6 (2013). DOI 10.1109/MOBS.2013.6614215
29. Yang, W., Prasad, M.R., Xie, T.: A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications. In: V. Cortellessa, D. Varró (eds.) Proc. of the 16th Int. Conference on Fundamental Approaches to Software Engineering (FASE 2013), pp. 250–265. Springer Berlin Heidelberg (2013). DOI 10.1007/978-3-642-37057-1_19
30. Zein, S., Salleh, N., Grundy, J.: A Systematic Mapping Study of Mobile Application Testing Techniques. Journal of Systems and Software **117**, 334–356 (2016). DOI 10.1016/j.jss.2016.03.065