GlobalLogic®

A Hitachi Group Company

Automation of Mobile Application Stress Scenarios for Performance Engineering

By Abhishek Gedam
a.gedam@globallogic.com

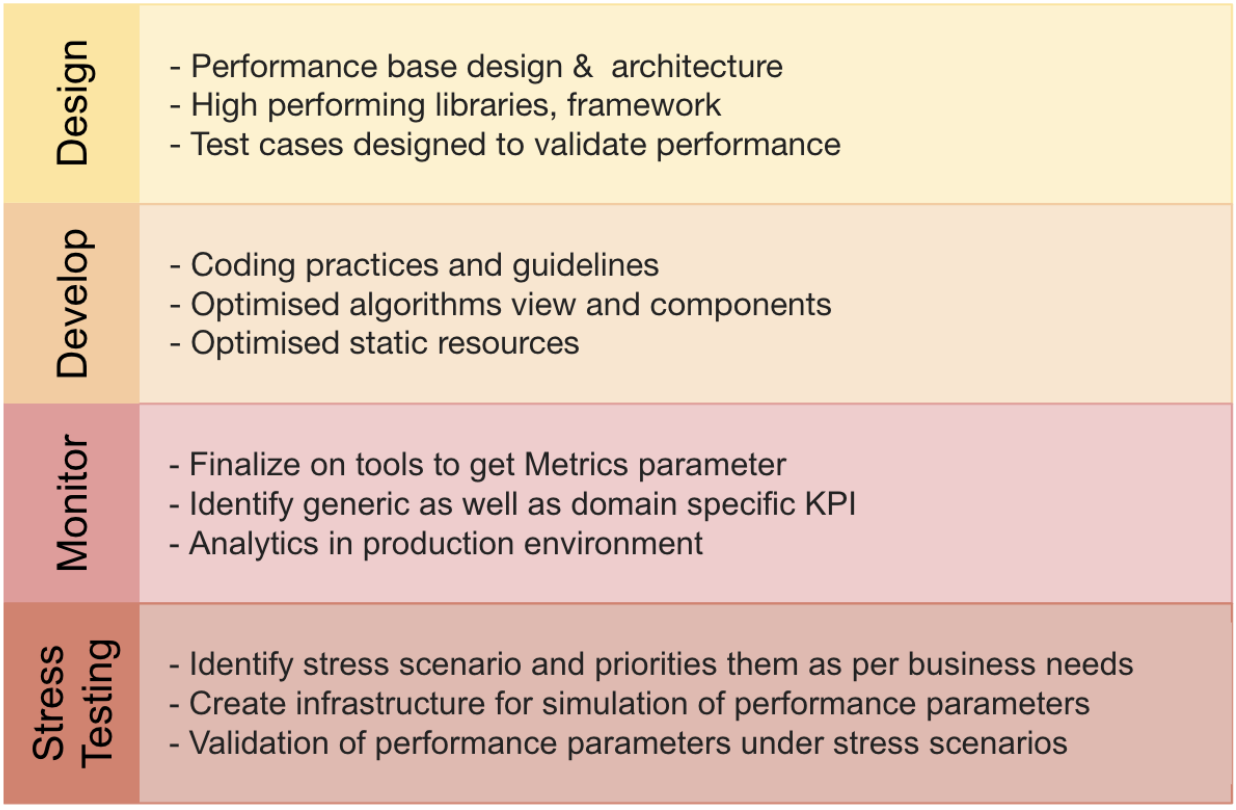
Table of Contents

1. Overview

Performance engineering is an important aspect of software development, it makes sure what is delivered to the end user is optimized to give a smooth user experience, having a small memory footprint consumes less process power and hence consumes less battery. There are different aspects of performance engineering out of which verification of application performance in stress scenarios is an important aspect. This makes sure that even if there is stress on application or host system, application can perform required operations or atleast give appropriate message and exit if required.

This document is focused on providing an overview on stress scenarios required to verify performance of application, their impact and potential way to simulate these scenarios. This document describes potential simulation of stress scenarios for Android devices but this can be done for other platforms as well.

Major Aspects of Performance Engineering



2. Stress Scenarios

Functional testing is done in a sandbox environment to test the functional aspect of an application, it is necessary to verify that the application is delivering what is required. In reality, functional testing also needs to be tested in a stressful or hostile environment. It is particularly important in gauging performance of applications in stress.

Following are major categories of stress scenario:

System Introduced

Every application is executed on an underline system or OS e.g. Android. When this system gets into stress due to change in system parameters or any external/internal factors, it impacts the performance of the overall system and hence target application also gets impacted. Target Applications may get slow and respond intermittently or even get closed abruptly.
E.g. Low Battery, Low CPU availability

Application Introduced

Target Application itself can get into stress due to wrong coding practices, unoptimized code and majorly not following good architectural and design practices.Target Applications may get slow and respond intermittently or even get closed abruptly.
E.g. Unoptimized code in Complex UI, non optimized network API calls

User Introduced

User action can put both application as well as system into stress, and hence can impact the usability of application. User can introduce stress by opening too many apps, frequency switching between apps etc

3. Important Stress Scenarios

These are the scenarios which can occur in general usage of the application.

Summary of High Priority Scenarios

Following tables list down High priority scenarios for stress testing, Potential way to simulate scenario and also highlight possibility integrating Appium (UI automation) to simulate scenario

Scenario	Simulation	Appium Integration Possibile?
Poor Internet connectivity	Using Charles Proxy tool to change bandwidth	Yes
Low battery	Using Android adb commands	Yes
Quick User Inputs	Using Android adb commands	Yes
Device orientation	Using Android adb commands	Yes
Interruptions due to Calls	Using Third party communication service	Yes
Switching between Foreground and Background	Using Appium Driver API	Yes
Termination of application in background	Using Appium Driver API	Yes
Long running tasks	Adding Appium test scenario	Yes

Poor Internet connectivity

Poor or intermittent internet may be introduced due to low network quality from service provider, switching of cellular network or due to switching between WIFI and cellular network

Impact on Application

- Since Most the application is dependent on server data, overall application performance/response get slows down due to slow internet and often user press multiple button/views to get out of blocking screen which can lead to unresponsiveness
- low internet bandwidth can result in Video/Audio glitches if adaptive bitrate streaming is not implemented
- Due to slow internet, Application screen dependent on server data to render shows loading (downloading of data) for long duration which gives bad user experience

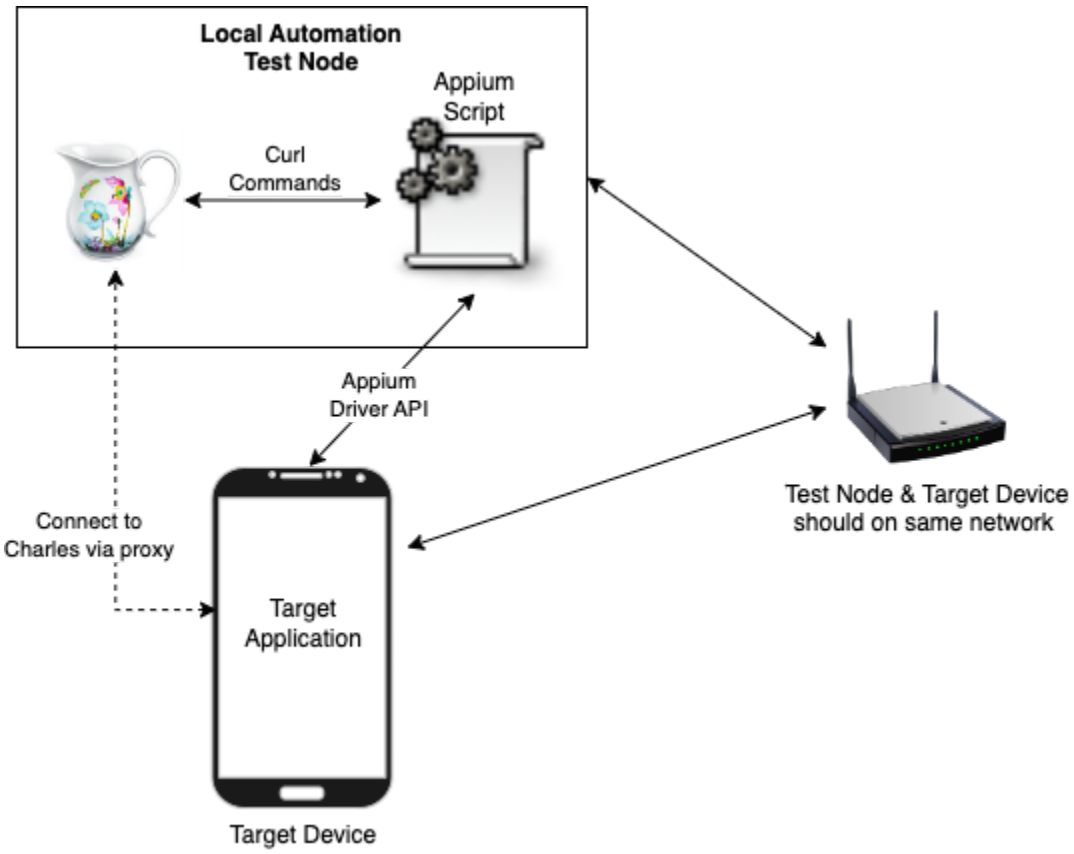
Simulation

Simulation of poor internet connectivity can be done by using Charles Proxy tool, Once Charles proxy is configured, During Automation scripts execution for required scenarios poor internet connectivity can be introduced by executing curl commands from scripts. e.g. `curl -v -x http://192.168.29.9:8888 http://control.charles/throttling/activate?preset=512+kbps+ISDN%2FDSL` will reduce target device bandwidth to max 512 kbps

Some of the important simulation are: Throttling & stability

Poor Internet connectivity simulation

Following is solution overview



Low battery

Low Battery can occur due to battery drainage, following can be potential reasons

- Too many application/services running in background
- Execution of CPU intensive complex mathematical or encryption instructions
- WIFI and network scanning
- Unoptimize device configuration e.g. longer duration to turn off display

Impact on Application

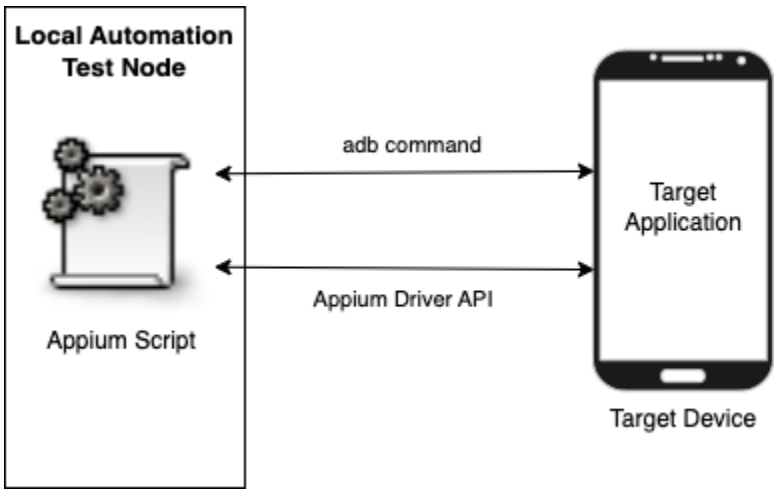
In case of low battery, System gets into power saving mode and hence it may kill services and applications to save battery. These services may be used by target applications hence have an impact on application behavior.

Simulation

In Android to simulate low battery condition, Android provides adb command to take device battery to lower level e.g. `adb shell dumpsys battery set level 4` Take device battery 4%

Simulation of adb commands

Following is solution overview



Quick User Inputs

In general an application processes only one synchronous event and once that event is processed completely then only the application is available to process the next event. e.g. Screen Navigation. But user can press buttons/views quickly hence such scenarios should be identified and tested.

Exception to above case in asynchronous operations like file download, where user can perform other operations while file is getting downloaded asynchronously.

Impact on Application

If application code is not written to accept correctly or ignore completely quick user input/click/touch then the application may get into an unresponsive state or even crash.

Simulation

In Android to simulate quick user inputs, Android provides adb command e.g. `adb shell adb -s DEVICE_ID --throttle 300 -p 20000`
This command inserts 20000 random events on target device with ID “DEVICE_ID” with 300 milliseconds interval between to events.

Simulation solution can be same as that of Simulation of adb commands

Device orientation

When device orientation is changed, the complete screen is re-rendered with the same or different UI layout and views. If configured as auto then orientation is changed as soon as the user physically changes device orientation, Also screen orientation can be changed manually by the user.

Impact on Application

If orientation is not handled correctly in code, application can give unexpected results e.g. Screen may not rotate or it can rotate but all views are not visible or even crash if initialization is not done correctly on orientation event.

Simulation

In Android to simulate orientation, Android provides adb command, few examples
Change to landscape

```
adb shell content insert --uri content://settings/system --bind
name:s:user_rotation --bind value:i:1
```

Change to Portrait

```
adb shell content insert --uri content://settings/system --bind
name:s:user_rotation --bind value:i:0
```

Simulation solution can be same as that of Simulation of adb commands

Interruptions due to Calls

Interruptions due to calls can cause connectivity loss for the internet. If there is an incoming/outgoing call on the same connection which is providing internet then there will be connectivity loss.This loss of internet can have an impact on application behavior.

Impact on Application

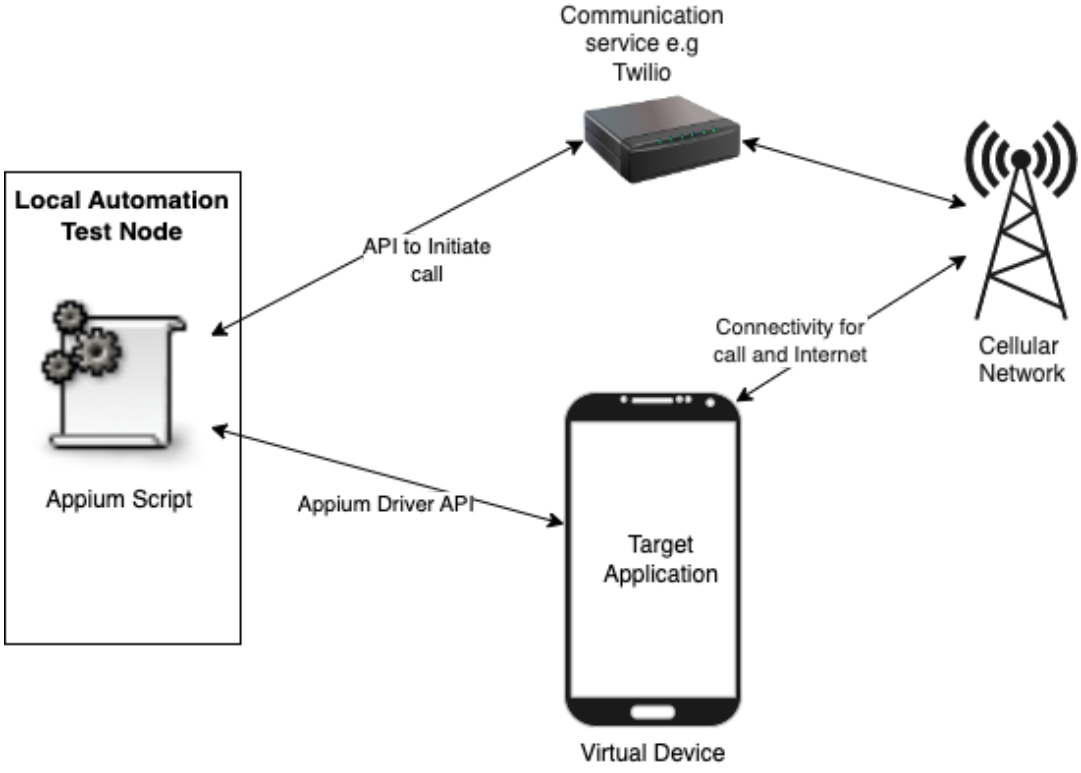
Applications will face loss of internet connectivity due to incoming/outgoing calls, this can result in unexpected behavior especially if connectivity is lost during server communication or video streaming. If proper handling is not done then even if connectivity is back, application might not resume from where server connectivity was lost.

Simulation

Simulation of incoming calls can be done using third party communication service providers like Twilio, Bandwidth. Once configured, these services provides API which can be called from Appium scripts to initiate call which can be received on target device

Simulation of incoming calls

Following is solution overview



Switching between Foreground and Background

Switch between foreground and background is very normal user behavior, along with user action this switch can happen due system alerts or interrupts like incoming calls.Application should provide consistent behavior irrespective on number of foreground/background switches

Impact on Application

When the application goes to the background, system gives an event to the application indicating it is being paused, if this callback is not handled properly as suggested by underline platform, it can lead to memory leaks and at times crash.

Simulation

Simulation of taking applications to foreground and background can be done using Appium driver API. e.g. `driver.background_app(5)` this takes application to background for specific duration, and after mentioned duration application will comeback to foreground

Termination of application in background

If the application is in the background or idle for a long duration, the system can terminate the application, also the user can terminate the application forcefully.

Impact on Application

In both termination cases (especially termination by user) there is a chance that the application might be performing an important operation which will be terminated abruptly and may take the application to unrecoverable state. e.g. database sync with server, file download,

Simulation

Simulation of terminating applications can be done using Appium driver api. e.g. `driver.close_app` this terminates application immediately.

Long running tasks

Due to the possibility of hours of streaming media consumption, OTT applications are prone to issues due to long running tasks and hence need to be tested.

Impact on Application

Long running task may consume heap memory which may not be claimed by system, due to this application may starve for memory and can lead to out of memory error, Also Long running tasks can block CPU for long duration, in this case system may kill application to conserve resources

Simulation

There is no direct way of simulation of long running tasks and hence automation scripts need to be utilized to initiate tasks (e.g. video) and keep it running for required hours to create various scenarios.

Application API/Payload

Application should be ready to handle maximum data from API response, maximum data may vary from API to API and some API may return data as pages. It is important to identify these issues so that API response limit can be changed (may be as per device family)

Impact on Application

Large size API responses can slow down applications or even cause low memory issues. In rare cases due to unoptimized code application crash is also possible.

Simulation

There is no direct way for simulation of large api response hence there can be specific environments with User/account/servers/region setup to send large API responses. In these environments each Network call returns maximum possible data, even if data is returned in chunks/pages, it should return maximum pages possible

Complex UI Population

UI screens driven by API response or screen generated due to user actions may get complex as data increases. Examples for such screens are the EPG guide generated from server response.

Impact on Application

Complex UI generated at runtime based on a large data set can slow down application along with UI and navigation issues due to increase in UI complexity, In unoptimized code this also leads to low memory issues due to large consumption of static resources.

Simulation

There is no direct way for simulation of complex UI population hence there can be specific environments with User/account/servers/region setup to send large API responses, These responses then can be used to generate complex UI screens.

4. Closing Notes

This document highlights some of the important stress scenarios which can be automated directly using various solutions, there are many more scenarios which cannot be directly automated e.g. Low RAM availability, High CPU usage. For such scenarios third party or external tools need to be developed