Testing Solution for VR apps
Presented by Carlos Cárdenas (DEKRA)
Motivation Facts

• Mobile applications consume data differently depending on various network conditions.
• Carries need to understand how the most popular Android and iOS apps consume data from the network.
• Carriers need to understand the network conditions that drive poor/excellent User Experiences.
• Carriers need to test an app in the exact same manner that customer use apps. No simulations...just real apps consuming real data.
DEKRA’s current solution for non VR apps

- Non VR Apps:
  - Downlink Intensive Video Streaming (including 4k)
  - Uplink Intensive Video Streaming
  - Two-way Video Streaming
  - Social Media
DEKRA’s current solution for non VR apps

**Testing Topology:**
High Scalability: M x N devices can be automated simultaneously

- **Test Controller**
- **Automation Agents**
- **Automated Devices**
- **Internet**
- **App Server**

*Automation Agents can be deployed in remote sites*

*ADB (Android Device Bridge) can use USB or Wi-Fi in either case. Automated Devices must be locally attached to the Automated Agent.*
DEKRA’s current solution for non VR apps

Automation Test Flow:
Example – Video Streaming App

KPI: Initial Buffering Time

Step 1: Open App
Step 2: Select Video
Step N: Start Play Back

Spinners Appearance Detection

QoE KPI deduction based on UI object events
DEKRA’s current solution for non VR apps

- **Appium** [open source test automation framework for use with native, hybrid and mobile web apps] for
  - Browsing through the App menu
  - Recognizing UI objects (e.g., spinner, progression bar)

- **ADB** (Android Device Bridge) for device data consumption reporting.

- **OCR** (Optical Character Recognition) for extracting App information:
  - Video Resolution
  - Buffer Health
DEKRA’s current solution for non VR apps

The following KPIs have been proved:

<table>
<thead>
<tr>
<th>Mobile Apps</th>
<th>KPIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>All (App Agnostic)</td>
<td>Battery, Data Usage, Throughput</td>
</tr>
<tr>
<td>Netflix</td>
<td>Initial Buffering, Re-bufferings</td>
</tr>
<tr>
<td>YouTube</td>
<td>Initial Buffering, Re-bufferings, Video Resolution</td>
</tr>
<tr>
<td>Instagram</td>
<td>Access Time, Initial Buffering, Re-bufferings</td>
</tr>
<tr>
<td>Periscope</td>
<td>Initial Buffering, Re-bufferings</td>
</tr>
<tr>
<td>Skype Video Call</td>
<td>Call Setup Time, Call Result, MOS</td>
</tr>
<tr>
<td>WhatsApp</td>
<td>Sharing Time, MOS</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
DEKRA’s current solution for non VR apps

- **Limitations of this approach for testing VR/Gaming apps:**
  - Performing Movement
    - VR and gaming apps require physical movement of the hosting device. As the gyroscope and accelerometer cannot be mocked, a hardware platform is required.
  - Retrieving App state:
    - Unlike other apps, VR and gaming apps are programmed in an Android UI Canvas where the graphical engine works (e.g., Open GL). Appium (or similar) cannot recognize UI objects inside the App gfx canvas.
DEKRA’s current solution for non VR apps

- Limitations of this approach for testing VR/Gaming apps:
Testing solution for VR/Gaming apps

- In order to overcome those limitations we have upgraded the architecture:

Four Servo-Motors:
1. Pitch (-90°, +90°)
2. Yaw 1 (-180°, 0°)
3. Yaw 2 (0°, +180°)
4. Roll (-90°, +90°)
Testing solution for VR/Gaming apps

Architecture

1 Capture UI
2 Act on Platform
3 Emulate touch on Device
Test Solution Requirements

- "Time to load a virtual scene" ($t_2 - t_1$), where
  - $t_1$ = user clicks on "start scene/experience" button
  - $t_2$ = the scene is totally rendered

- "Lagging" ($t_4 - t_3$), where
  - $t_3$ = user sends command to the app (e.g., roll phone)
  - $t_4$ = device UI shows command response (e.g., airplane has rolled)

- "Frame per seconds" as smoothness indicator...

- "Data Consumption"
Test Solution Requirements

- Minimize “reaction time” $t_5 - t_6$, where
  - $t_5 = \text{target appears on the screen}$
  - $t_6 = \text{tap/touch on that target}$

Why?

VR/Gaming: Automate the browsing through the app where some UI could be moving objects.

Gaming: Shoot at moving target
Measuring Lagging Automation Agents

Automated Device / Robotic Platform

Reported Lagging = Measured Lagging - \( \Delta t_1 \) - \( \Delta t_2 \) - \( \Delta t_3 \)

Some calibration is needed!!
More on Reaction Time….

**Goal**

Average Human Reaction Time: 284 ms

[https://www.humanbenchmark.com/tests/reactiontime/statistics]

**Implemented Reaction Time:**

- $\Delta t_1 \sim 1-10$ ms $\rightarrow$ TCP socket latency
- $\Delta t_2 \sim 100-200$ ms $\rightarrow$ Image Recognition performance
- $\Delta t_3 \sim 1-10$ ms $\rightarrow$ Serial Comm latency
- $\Delta t_4 \sim 100$ ms $\rightarrow$ Time to aim at object (Robotic action)
- $\Delta t_5 \sim 1-10$ ms $\rightarrow$ TCP socket latency

DEKRA’s solution Reaction Time = (230, 330) ms
Test Solution Requirements

- High performance screen capture
  - Requirement: Higher than 24 frames per second
- Low delay screen touch
  - Requirement: Lower than 10 ms
- IR (Image Recognition)
  - Requirement: High pattern matching accuracy and high performance
The IR matching score is another important trade-off parameter.

<table>
<thead>
<tr>
<th>Options</th>
<th>Trade off</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher screenshot resolution</td>
<td>Higher $\Delta t_1$ and $\Delta t_2$</td>
<td>Less measurements, more accurate KPI, slow for gaming apps</td>
</tr>
<tr>
<td></td>
<td>Less false negative IR detections</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Less true IR positive detections</td>
<td></td>
</tr>
<tr>
<td>Lower screenshot resolution</td>
<td>Lower $\Delta t_1$ and $\Delta t_2$</td>
<td>More measurements, less accurate KPI, suitable for gaming apps</td>
</tr>
<tr>
<td></td>
<td>More false positive IR detections</td>
<td></td>
</tr>
<tr>
<td></td>
<td>More true positive IR detections</td>
<td></td>
</tr>
</tbody>
</table>
## KPIs Implemented

<table>
<thead>
<tr>
<th>KPI</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Resources Usage</td>
<td>Data Usage, Throughput</td>
</tr>
<tr>
<td>Device Resources Usage</td>
<td>Battery, CPU, GPU</td>
</tr>
<tr>
<td>Time to load the virtual world</td>
<td>Time elapsed from selecting a scenario (world, experience, etc.) to loading the 3D visual context</td>
</tr>
<tr>
<td>Immersion Cut-off</td>
<td>Probability that successfully started immersion is ended by a cause other than the intentional termination by the user</td>
</tr>
<tr>
<td>Lagging</td>
<td>Time elapsed from acting on the device to the reaction of the UI</td>
</tr>
</tbody>
</table>
Showcase: Testing Google Cardboard App

- VR experience, e.g., for Google Earth
- Replacing the mouse by the head movement
Showcase: Testing Google Cardboard App

Automatic test cycles: **40 repetitions / BW configuration**
Showcase: Testing Google Cardboard App

Open App

Navigate through the app until click “start experience”

Automatic test cycles: 40 repetitions / BW configuration

Measurement
Showcase: Testing Google Cardboard App

KPI: Average Time To Load Scenario (s)

9.57 s (best scenario)

KPI: Time to load scene

KPI: Network Data Usage

8 MB (all scenarios)

X-Axis: Imposed BW (Mbit/s)
Showcase: Testing a Cloud Gaming app

• Test script:
  • Open App, Select game
  • Start game
  • Leave the car until it crashes with the first roadblock in its way (this happens after 280 seconds approximately)
  • Close game

We have selected this use case for repeatability across different network conditions
Showcase: Testing a Cloud Gaming app

<table>
<thead>
<tr>
<th>Baseline Network Condition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT*</td>
<td>17 ms</td>
</tr>
<tr>
<td>DL Speed</td>
<td>90 Mbit/s</td>
</tr>
<tr>
<td>UL Speed</td>
<td>75 Mbit/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Baseline Frame Rate (fps)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>38</td>
</tr>
<tr>
<td>Avg</td>
<td>51.1</td>
</tr>
<tr>
<td>Max</td>
<td>55</td>
</tr>
</tbody>
</table>

Max data rate consumed by the game 17.9 Mbit/s
Showcase: Testing a Cloud Gaming app
Key-takeaways

• The “time to load scenario” KPI is severely impacted by the quality of the network access, mainly the available bandwidth (Mbit/s).
  • Online Virtual Reality apps consumes huge amount of network data, which has impact on network planning and deployments.
  • Online Virtual Reality apps requires high device GPU performance, so need flag-ship device for a good User Experience.

• 4G mobile networks are not suitable for 5G cloud gaming use case because...
  • The frame rate at the user device is unacceptable with a link capacity below 10 Mbit/s, or with a link loss (at IP level) above 0.1 %
  • The frame rate and the lagging at the user device is severely affected by fluctuations in the round trip time of the network (a.k.a. jitter).
Lesson-learnt

• Objective performance measurements provide insights about 5G VR and Gaming use cases.

• Thanks to the fast closed-loop response time of the solution on Android, the solution can be also used to measure online games apps.

• The image recognition library matching score parameter has impact on the accuracy of the “time to load scenario” measurement.

• The testing solution needs another upgrade to automate a gamepad. Online games may use external gamepad (instead of gyroscope/accelerometer) for which the implemented robotic platform is not suitable.
This testing solution has been developed inside the scope of TRIANGLE project.

TRIANGLE Project
5G Applications and Devices Benchmarking

Co-funded by the Horizon 2020 Framework Program of the European Union

www.triangle-project.eu
info@triangle-project.eu