# System Health Monitoring
## (Based on Android Platform)

**Major Project Report**

*Submitted in fulfillment of the requirements for the degree of*

**Master of Technology in
Electronics & Communication Engineering
(Embedded Systems)**

By


**Hemadri Upadhyay**

**(21MECE13)**

**Electronics & Communication Engineering Department Institute of Technology
Nirma University
Ahmedabad-382 481**

# System Health Monitoring
## (Based on Android Platform)

**Major Project Report**

*Submitted in partial fulfillment of the requirements*

*for the degree of*

**Master of Technology**

**in**

**Electronics & Communication Engineering**

By


**Hemadri Upadhyay**
**(21MECE13)**


Under the guidance of

| External Project Guide: | Internal Project Guide: |
| --- | --- |
| **Yakasiri Jayachadra** | **Mr. Jayesh Patel** |
| Principal Software Engineer | Assi. Professor, EC Department, |
| Vantiva | Institute of Technology, |
| Chennai. | Nirma University, Ahmedabad. |

**Electronics & Communication Engineering Department Institute of Technology-Nirma University**

**Ahmedabad-382 481**

# Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

**-Hemadri Upadhyay**
**(21MECE13)**

# Disclaimer

**“The content of this report does not represent the technology, opinions, beliefs, or positions of Vantiva-Pushing the Edge Private Limited, its employees, vendors, customers, or associates.”**

## Certificate

This is to certify that the Major Project entitled **"System Health Monitoring (Based on Android Platform)"** submitted by **Hemadri Upadhyay (21MECE13)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma. Date: Place: Ahmedabad

**Assi. Prof. Mr. Jayesh Patel**
Internal Guide

**Prof (Dr.) N.P. Gajjar**
Program Coordinator

**Prof. (Dr.) Usha Mehta**
Section Head, EC

**Dr. Rajesh Patel**
Director, IT

# Certificate

This is to certify that the Major Project entitled **"System Health Monitoring (Based on Android Platform)"** submitted by **Hemadri Upadhyay (21MECE13)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached the level required for being accepted for examination.

**Mr. Yakasiri Jayachandra**
**Principal Software Engineer**
**Vantiva**
**Pushing the Edge**
**Chennai**

# Acknowledgements

I would like to express my gratitude and sincere thanks to **Prof (Dr.) N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems for guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Assi. Prof. Mr. Jayesh Patel**, guide of my internship project for his exemplary guidance, monitoring and constant encouragement.

I would also like to thank **Mr. Yakasiri Jayachandra**, external guide of my internship project from **Vantiva-Pushing the Edge**, for guidance, monitoring and encouragement regarding the project.

**- Hemadri Upadhyay**

**(21MECE13)**

# Contents

**Declaration** iii

**Disclaimer** iv

**Certificate** v

**Acknowledgments** vii

**Abstract** xii

**Abbreviation Notation and Nomenclature** xiii

**1 Introduction** **13**

1.1 Motivation ....................13

1.2 Problem Statement ....................13

1.3 Approach ....................14

1.4 Scope of Work ....................14

**2 Literature Survey** **15**

2.1 Overview ....................15

2.2 Parameters to validate application ....................18

**3 System Health Monitoring** **21**

3.1 Working Methodology ....................21

3.2 Implantations.....22
3.3 Output.....24
3.4 Live Graph Plotting tool for system performance analysis..... 25

**4 Third party app Validation tool 26**
4.1 Third party apps.....26
4.2 Block diagram of GUI for third party app validation tool.....27
4.3 Output.....31

**5 Conclusion 33**
5.1 Conclusion.....33

**References 34**

# List of Figures


2.1 Step by Step explanation of working of Perfetto ............................................15
2.2 Trace Visualization ............................................................................17

3.1 Basic block diagram of live graph plotting and analysis ..................................21
3.2 Step by step implementation ...............................................................22
3.3 Steps for implementation.......................................................................23

3.4 Graph Plotting .......................................................................................23

3.5 Result .....................................................................................................24
3.6 Live graph Plotting Tool GUI ...............................................................25
4.1 Front-End Design of App Validation Tool........................................................27
4.2 Memory Leak.............................................................................................28
4.3 Memory Footprint.......................................................................................29
4.4 CPU Footprint ...........................................................................................29
4.5 Number of Threads .....................................................................................30
4.6 Front-End of Tool........................................................................................31
4.7 App Validation Tool output for CPU Footprint and Number of Threads .......31
4.8 App Validation Tool output for CPU Footprint and Memory Footprint.........32

# Abstract

Performance testing is the process of determining how well a system responds and remains stable under a specific demand. Performance tests are often carried out to evaluate application size, robustness, and speed. The method includes "performance" indicators like:

- periods for processing server requests
- acceptable volumes of concurrent users
- usage of processor memory; possible app problems, their frequency and nature

Performance testing encapsulates all evaluations that confirm the responsiveness, dependability, robustness, and appropriate sizing of an application. It looks at several parameters, including browser, page, and network response times, server query processing times, the number of concurrent users that may be supported by the architecture, CPU memory consumption, and the quantity and type of errors that may occur when using an application.

Your software will satisfy the expected levels of service and deliver a great user experience if you execute performance tests to make sure. Before your applications are put into production, they will indicate enhancements you should make to them in terms of performance, reliability, and scalability. Applications that are distributed to the public without being tested may have a variety of issues that, in some situations, permanently harm a brand's reputation.

Performance testing needs to be done correctly if applications are to be adopted, successful, and productive.

The implementation of a continuous optimization performance testing approach is essential to the achievement of an effective overarching digital strategy, even though fixing production performance issues can be very expensive.

# Abbreviation Notation and Nomenclature

ADB ........................................................................................................ Android Debug Bridge
Apps ………………………………………………………………………………………… Applications
STB………………………………………………………………………………………….   Set Top Box
PSS………………………………………………………………………………………….   Proportional Set Size
RSS…………………………………………………………………………………………. Resident Set Size

# Chapter 1

# Introduction

## 1.1 Motivation

Before making a system available to the end user, performance testing involves determining if it will function as planned under various performance indicators and imagining what the user will experience. As a result, system performance testing guarantees that system functionalities operate as intended by locating and fixing faults as well as locating and removing any performance bottlenecks. The objective is to gain understanding of a system's characteristics, such as its robustness, scalability, optimal sizing, and speed. Without performance testing, your end customers can face subpar system responsiveness, user experience, and device usability.

## 1.2 Problem Statement

To improvise the performance of the Android TV Set Top Box along with maintaining the upgrades brought previously with the use of a new tool. Functional testing is crucial, but there are other aspects of the user experience that are just as important and should be checked automatically in every build. Performance is one of these other dimensions. Performance is essentially a measure of how responsive your system is to users, and it can take many different forms, from CPU and memory consumption to network request times.

## 1.3 Approach

Customer happiness depends heavily on performance testing; if your application's performance falls short of their expectations, they will switch to a rival. Performance testing is intricate and necessitates expert test design. Understanding the difficulties of performance testing, as well as the procedure and tools available to construct an efficient performance test, is crucial for developing a thorough test strategy.

With the aid of the Android Debug Bridge and the tool called Perfetto, you may gather performance data from Android devices (ADB). Use the ADB shell Perfetto- command to launch the Perfetto utility. Performance traces from your device are gathered by Perfetto using a variety of sources, including:

Atrace for user-space annotation in services and apps
ftrace for information from the kernel
Information on services' and apps' native memory utilization may be found at heapprofd.
Apart from this, we can plot graph from live data collection and analyze the performance of system.

## 1.4 Scope of Work

Apart from Android system performance, Android Apps' performance is also an important process to be considered. There are some parameters which are to be tested for app validation before integrating apps into the Set-Top Boxes.

# Chapter 2

# Literature Survey

## 2.1 Overview

### Perfetto - System profiling, app tracing and trace analysis

An open-source stack for performance instrumentation and trace analysis is called Perfetto. It provides services and libraries for tracing system- and app-level activity, native + java heap profiling, a library for SQL tracing analysis, and a web-based UI for visualizing and exploring multi-GB traces.

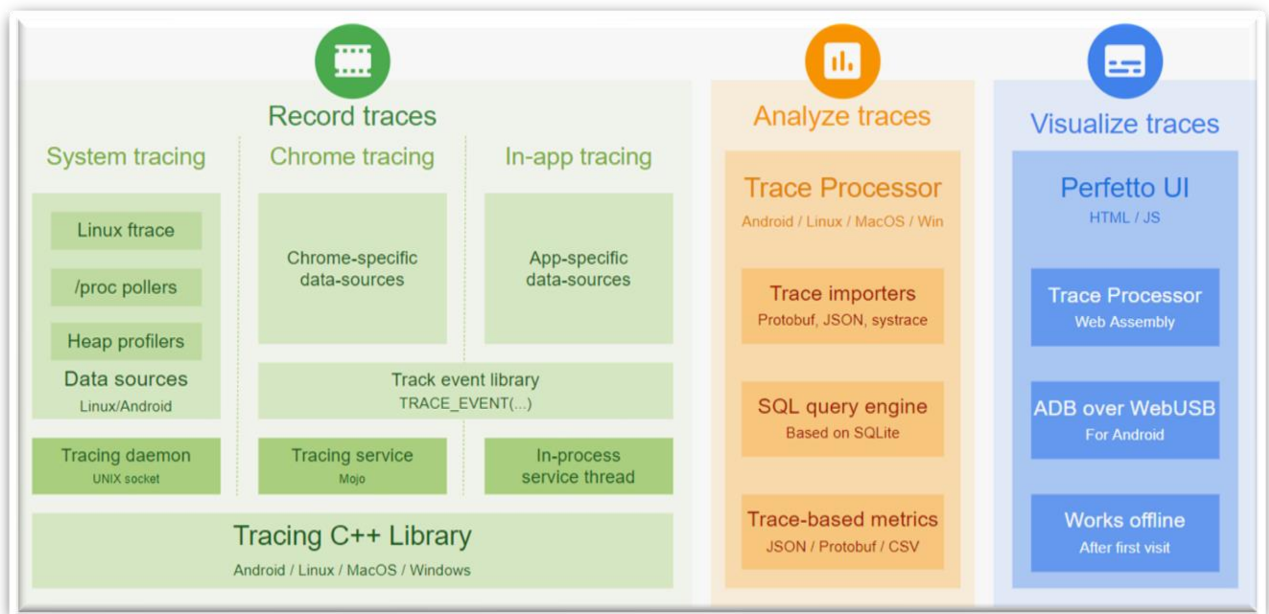



Fig. 2.1: Step by step explanation of working of Perfetto

## Recording traces:

A revolutionary userspace-to-userspace tracing protocol based on direct protobuf serialisation onto a shared memory buffer is the primary innovation of Perfetto. Internally, the built-in data sources employ the tracing protocol, and C++ apps can access it via the Tracing SDK and the Track Event Library.

Through an extensible protobuf-based capability announcement and data source configuration mechanism, this novel tracing protocol enables dynamic setup of every aspect of tracing. Multiplexing several data sources onto various user-defined buffer subsets enables the streaming of arbitrary lengthy traces onto the filesystem.

## Trace analysis:

Beyond the trace recording features, the Perfetto codebase has a dedicated project called Trace Processor for importing, parsing, and querying both new and old trace formats. A portable C++17 library called Trace Processor offers column-oriented table storage, was created on the fly to retain hours' worth of trace data in memory and exposes a SQL query interface based on the well-liked SQLite query engine. To examine the trace data, the trace data model is transformed into a collection of SQL tables that can be queried and linked in a variety of incredibly powerful and flexible ways.

A trace-based metrics subsystem made up of pre-built and customizable queries that may produce strongly typed summaries about a trace in the form of JSON or protobuf messages is also included in the Trace Processor.

## Trace visualization:

Additionally, Perfetto offers a brand-new trace visualizer that can be accessed at ui.perfetto.dev for accessing and searching hours-long traces. The brand-new visualizer benefits from contemporary web platform technology. The UI is constantly snappy thanks to its multi-threading design and Web-Workers; using Web-Assembly, the analytical capability of Trace Processor and SQLite is completely accessible in-browser. Once it has been opened once, the Perfetto UI can be used entirely offline. The browser processes any traces that are opened through the UI locally; server-side interaction is not necessary.

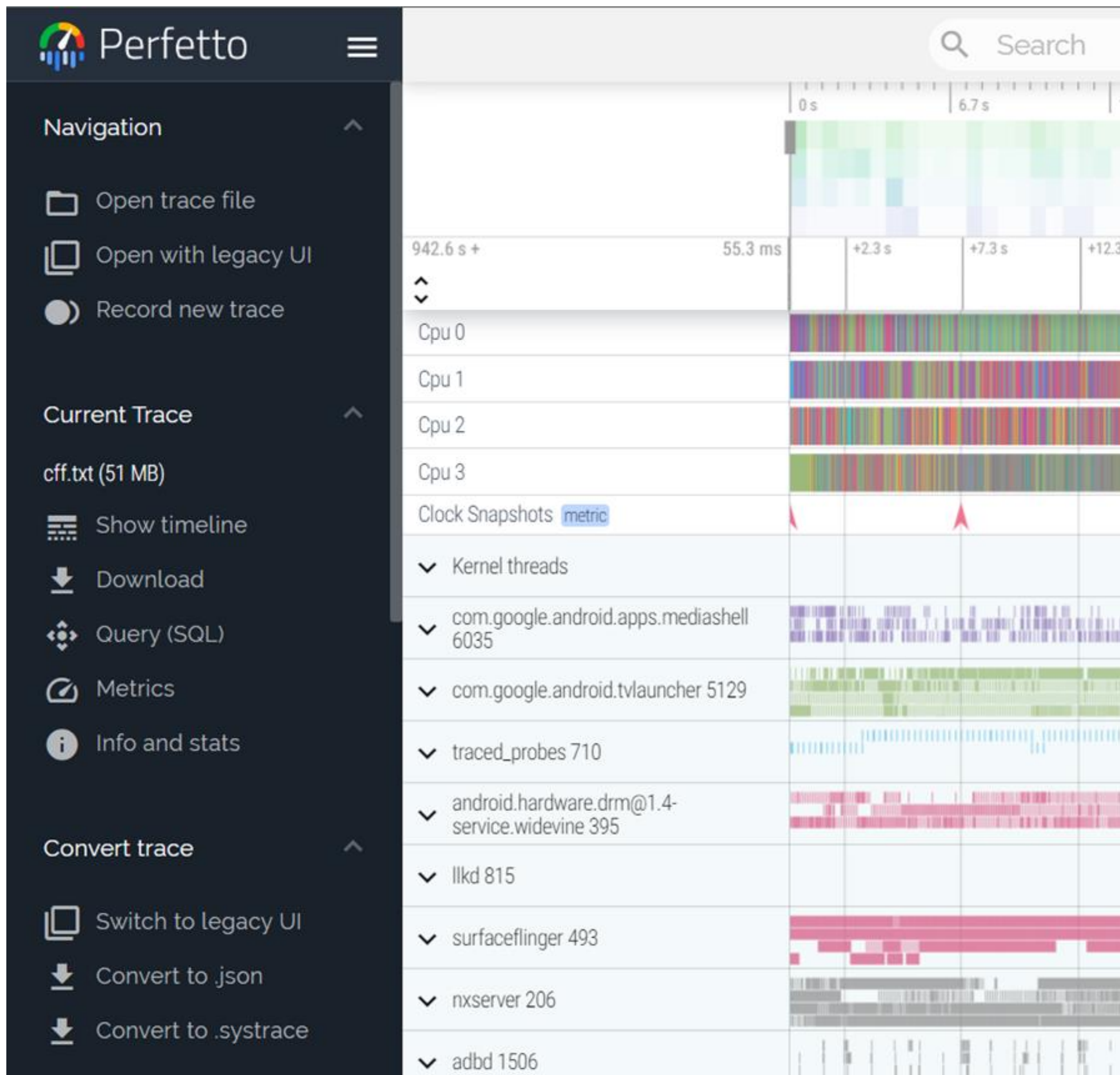



Fig.2.2: Trace Visualization

## 2.2 Parameters to Validate Android Applications

### 2.2.1 Memory Leak/ Memory Footprint:

When a program allocates memory for an object but forgets to release the memory when the object is no longer needed, a memory leak occurs. Leaky memory builds up over time, causing subpar program performance and even crashes.

Memory Leak depends upon PSS, RSS, Heap Allocation.

**PSS:** The Proportional Set Size (PSS), which Android calculates when assessing your app's heap, takes both dirty and clean pages that are shared with other processes into account—but only to the extent that it is proportional to the number of apps that share that RAM.

**RSS:** The amount of memory used by a process while it is being held in main memory is known as resident set size (RSS) in computing (RAM). Because some of the occupied memory was paged out or because some portions of the executable were never loaded, the remaining memory is either in the swap space or file system.

**Heap Allocation:** For dynamic memory allocation, use the Heap.
Android imposes a strict limit on the heap size for each running application in order to ensure a smooth user experience. The maximum heap size varies from device to device and depends on the RAM capacity.
Every application on Android has a maximum heap size limit (which varies depending on the device) denoted by the term "largeHeap." By using the getMemoryClass() API of the ActivityManager service, you may determine the maximum heap size that is accessible for your application. Most Android 2.3 or later devices may return this size as 24MB or greater, however the maximum value is 36 MB (depending on the specific device configuration).
Your software will crash and throw an OutOfMemoryError if it reaches this heap limit and tries to allocate more memory. Object allocation takes place in heap memory. An object is always created in the heap whenever you create one. Since the heap differs from the stack, the objects won't be automatically released after the function is finished. Garbage Collector is a superhero we gave virtual machines like JVM (Java Virtual Machine), DVM (Dalvik Virtual Machine), or ART (Android Runtime) to take care of finding and recovering such unneeded items to free up additional memory. If there is an item in the heap that doesn't contain any references to other objects, the garbage collector will search for those.

The term "memory footprint" describes how much of the system's main memory a running software utilizes or refers to. The term "footprint" typically relates to how much space an object takes up physically, indicating its size. The memory footprint of a software program in computing represents the amount of runtime memory needed to run the program. In addition to the memory needed to hold any additional data structures, such as symbol tables, debugging data structures, open files, shared libraries mapped to the current process, etc., that the program might require while executing and will be loaded at least once throughout the entire run, this also includes the code segment containing the program's instructions, the data segment (both initialized and uninitialized), the heap memory, the call stack, and memory required to hold any additional data structures.

### 2.2.2 CPU Footprint:

Each processor has a finite amount of processing power that it can use to carry out commands and run different programs. More jobs can be successfully completed at once with a better CPU. No matter how powerful the CPU is, every chip ultimately reaches its limit and starts to slow down. Depending on how much CPU you are currently using: In other words, the quantity of simultaneous tasks that are placed on your CPU. Your CPU utilization should be low when you are not using numerous applications, and in a perfect world, everything should go without a hitch. However, if you launch a CPU-demanding application (such a game or video editing software), you could observe that response times grow longer as your CPU utilization rises. CPU use fluctuations are common and not cause for concern if your computer keeps functioning efficiently. Checking your PC's CPU use is the first thing you should do if you're not happy with its performance.

### 2.2.3 Number of Threads:

The priority given to the threads in your program relies in part on where it is in the lifecycle. Setting a thread's priority as you establish and maintain it will ensure that the right threads receive the right priorities at the appropriate times. If it's set too high, your thread can interfere with the UI thread and the RenderThread, resulting in frame drops in your program. If it is too low, async processes (such loading images) may take longer than necessary to complete.

### 2.2.4 Sluggishness:

Motion events use an action code and a series of axis values to represent movements. The action code describes the state change, such as a pointer going up or down, that took place. The location and other movement aspects are described by the axis values.

For instance, when a user first touches a screen, the system sends a touch event with the action code ACTION_DOWN and a collection of axis values to the appropriate View. These values contain the touch's X and Y coordinates as well as details about the pressure, size, and direction of the contact area.

Some devices can report multiple movement traces at the same time. Multi-touch screens emit one movement trace for each finger. The individual fingers or other objects that generate movement traces are referred to as pointers. Motion events contain information about all the pointers that are currently active even if some of them have not moved since the last event was delivered.

The number of pointers only ever changes by one as individual pointer go up and down, except when the gesture is canceled.

### 2.2.5 CPU Footprint during Background Video Streaming:

This parameter testing is only possible in secured STBs, as we can directly feed such STBs with live video streaming. It will check the level of CPU utilization on launcher along with live video streaming in background.

### 2.2.6 OnTrimMemory Callbacks:

To incrementally release memory based on current system limits, you need implement OnTrimMemory(int). By allowing the system to keep your process running longer, using this callback to release your resources not only contributes to a more responsive system overall but also directly improves the user experience for your app. The system is more likely to kill your process while it is cached in the least-recently used (LRU) list if you don't reduce your resources based on memory levels provided by this callback, which would force your app to restart and restore all data when the user returns to it.

Instead of representing a single linear development of memory restrictions, the values returned by OnTrimMemory(int) give you a variety of hints regarding memory availability:

- When app is running:

  1. TRIM_MEMORY_RUNNING_MODERATE
     The memory of the device is getting low. Your app is open and cannot be stopped.

  2. TRIM_MEMORY_RUNNING_LOW
     The system is using a lot less memory. Please free up unneeded resources to enhance system efficiency (which directly affects the performance of your app, even though it is currently running and cannot be killed).

  3. TRIM_MEMORY_RUNNING_CRITICAL
     The system is running out of RAM quickly. You should release non-critical resources right once to avoid performance deterioration even though your program is not yet regarded as a killable process. If apps do not release resources, the system will start killing background processes.

- When app's visibility changes:

  1. TRIM_MEMORY_UI_HIDDEN
     Now that your app's user interface (UI) is hidden, it's a good idea to release sizable resources that are exclusively needed by your UI.

- When app's background LRU list process is active:

  1. TRIM_MEMORY_BACKGROUND
     Your process is located close to the start of the LRU list, and the system is running low on memory. Although the system may already be killing processes in the LRU list, your app process is not at a high danger of being killed. As a result, you should release resources that are simple to recover so your process will stay in the list and resume soon when the user returns to your app.

  2. TRIM_MEMORY_MODERATE
     Your process is located close to the middle of the LRU list, and the system is running low on memory. Your process may be terminated if the system's memory requirements increase.

  3. TRIM_MEMORY_COMPLETE
     If the system does not recover memory right away, your process will be among the first ones to be killed. The system is now low on memory. Release everything that isn't necessary for restarting your app state.

### 2.2.7 Memory Leak during App Switching:

It is very important for the system to support multiple app access at a same time for which app switching should work smoothly. If memory profiler is indicating sudden increase in memory usage, then chances are high of memory leakage occurrence. That is why it is very important to check whether memory is leaking during app switching or not.

### 2.2.8 Quick Navigation Scripts:

Navigation is the basic requirement for any Set-Top Box Launcher. Quick navigation scripts check how smoothly launcher is working or how much time it is taking to navigate throughout the launcher screen. It checks the responsiveness of Set-Top Box.

# Chapter 3

# System Health Monitoring

## 3.1 Working Methodology

As shown in fig. 2.3, Stop Box is connected to PC through ADB shell. Through ADB commands, we can trace the data from STB to PC. (i.e. using Push/Pull commands). After collecting data through ADB Push/Pull commands We need to segregate important information plotting which into the graphical format, we can analyze system performance.

To segregate important information from collected data we are using SQL Queries.

Fig. 3.1 Basic block diagram of live graph plotting and analysis

## 3.2 Implementation

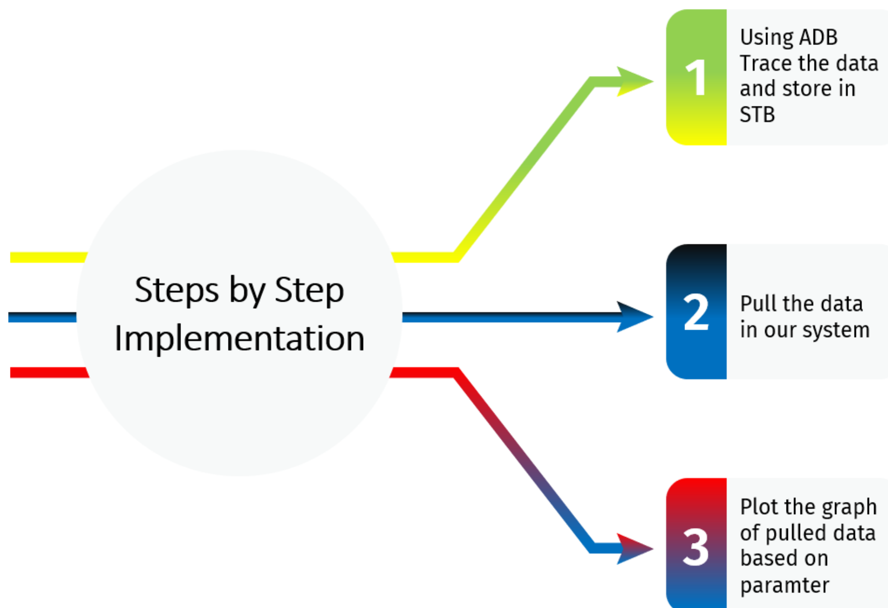



Fig. 3.2 Step by step Implementation

As mentioned in above figure, first of all, using ADB shell commands we will trace the data and store it into the STB. After that we will pull the data into our system. For further analysis we will segregate the information and plot it into a graphical representation which will be more useful in further analysis. Step by step implementation is mentioned in below figures.

## 3.3 Output

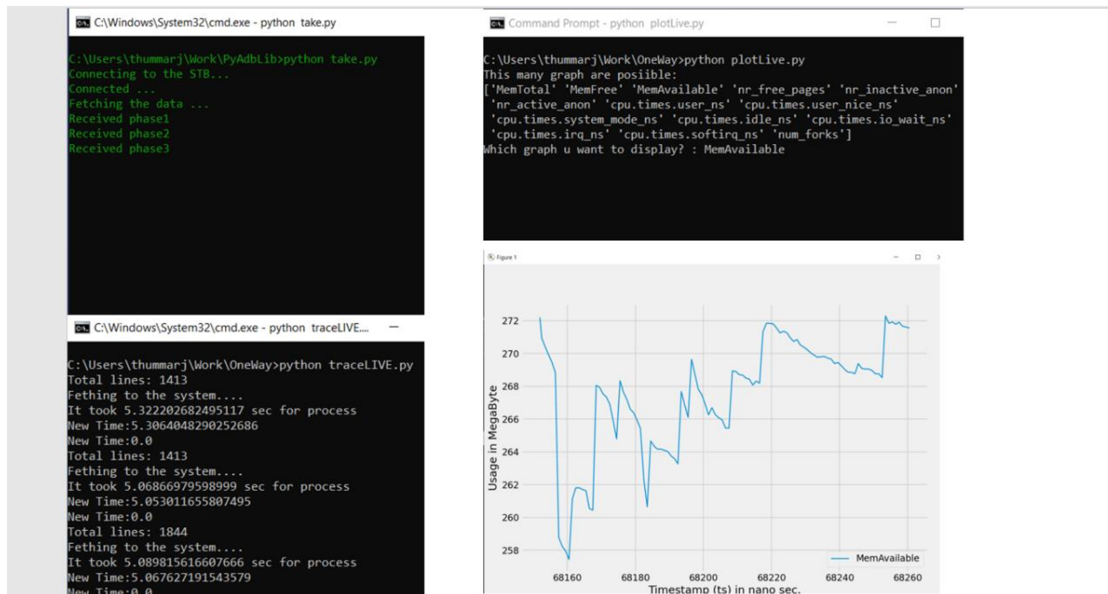



Fig. 3.5 Result

From above mentioned figure, you can see the final output of lie graph plotting. Through this graph plotting we can easily analysis the performance of any android system. Here you can see the graph is indicating how much memory is available with system with current usage.

## 3.4 Live Graph Plotting Tool for System Performance Analysis

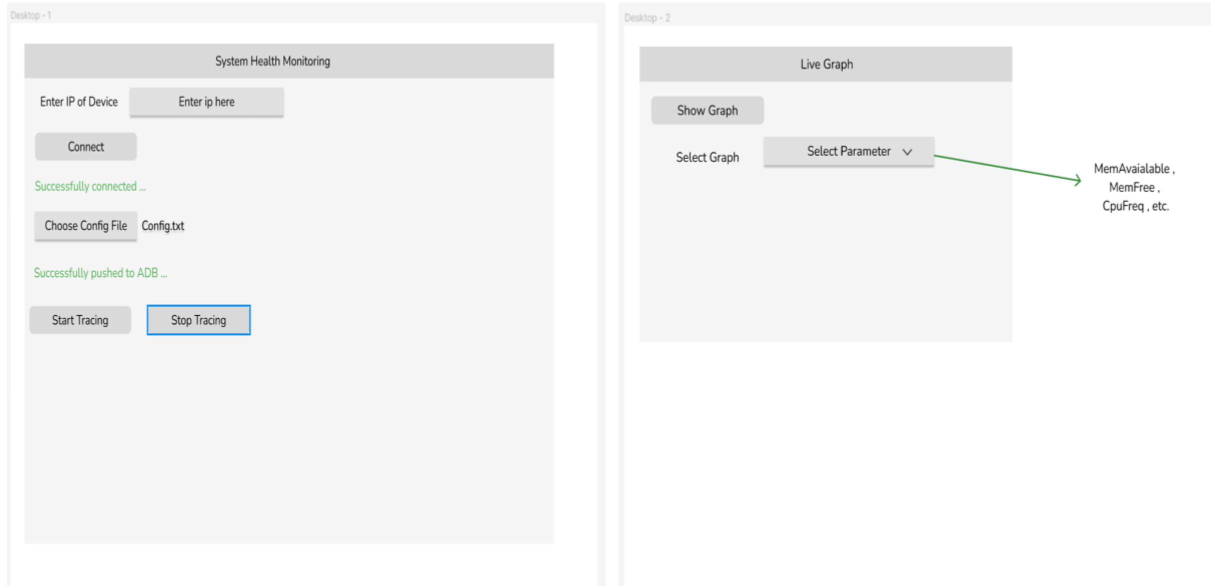



Fig. 3.6 Live Graph Plotting Tool GUI

To combine all the above mentioned steps, we have developed one tool which we perform all the tasks step by steps. As shown in above diagram, first of all we need to enter an IP address of the available STB. After connecting to IP Address we have to choose the config file using which ADB commands through we can collect data and store it into STB. Through Start and Stop Tracing button we can collect and store trace file into STB as well as we can pull that file into our system. Last step is Graph Plotting, which we can do based on the parameter available with us. This tool is designed using Python Tkinter and for backend development we have used Python Language.

# Chapter 4

# Third Party App Validation Tool

## 4.1 Third Party Apps

A software program created by a party other than the creator of a mobile device's operating system is referred to as a third-party app. For instance, several programs are made for Apple's or Google's operating systems by app development businesses or individual developers. The owner of the device or website may approve or disapprove of third-party apps. For instance, Apple developed the Safari web browser app that is a first-party, built-in program for the iPhone, but there are other web browser apps available in the App Store that Apple only authorized for use on the iPhone. They are third-party applications. Some apps that Facebook didn't create are allowed to operate on its social media platform. These are independent apps.

### Types of Third-Party Apps

The phrase "third-party app" may appear in several distinct contexts.

- Third-party applications are those developed for official app stores by developers other than Google (Google Play Store) or Apple (Apple App Store), and that adhere to the development standards necessary for those app shops. A third-party app is one that has been approved by a developer for a service like Facebook or Snapchat. A first-party app is one that is created by Facebook or Snapchat.

- Third-party apps are those that are distributed through unofficial third-party app stores or websites and are made by organizations independent from the hardware or operating system. To avoid infection, exercise caution when downloading programs from any source, but especially from unlicensed app stores or websites.

- A third-party app is one that establishes a connection with another service (or its app) in order to offer better functionality or access profile data. An illustration of this is the quiz app Quizzstar, which requests authorization to access specific areas of a Facebook profile. No one downloads third-party apps of this type. Instead, through its link to the other service or app, the app is given access to potentially sensitive data.

### How First-Party Apps differ from Third-Party Apps

Applications developed and delivered by the hardware or software developer are referred to as first-party apps. Music, Messages, and Books are a few first-party iPhone apps as examples.

These apps are referred to as "first-party" since they were developed by a manufacturer specifically for their line of mobile devices, frequently using proprietary source code. For instance, a first-party app is one that Apple develops for an Apple device, such as an iPhone. Examples of first-party apps for Android devices include Gmail, Google Drive, and Google Chrome for mobile devices because Google developed the Android mobile operating system.

There may be a version of an app accessible for various sorts of devices even though it is a first-party app for one type of device. For instance, the Apple App Store offers a version of Google apps that is compatible with iPhones and iPads. On iOS devices, those are regarded as third-party apps.

## 4.2 Block diagram of GUI for Third Party App validation Tool

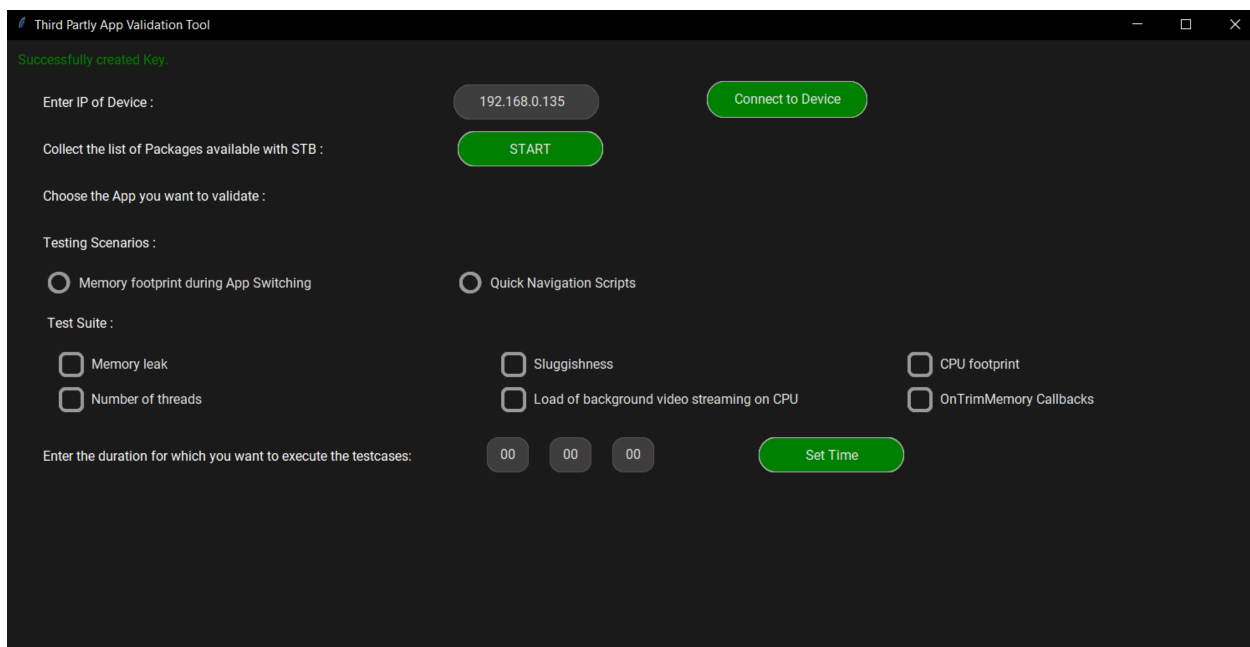



Fig. 4.1 Front-End design of App Validation Tool

As per mentioned in above figure, it is featuring the GUI of App Validation Tool in which, first, we must enter the IP address of the given STB so that we can connect STB to PC or system. As we are connecting to STB we will get the list of packages available with STB from which we have to choose one to validate. There are six test cases or parameters and two testing scenarios which this tool validates to test the performance the App.

## 4.3 Flowchart for Third Party App validation Tool

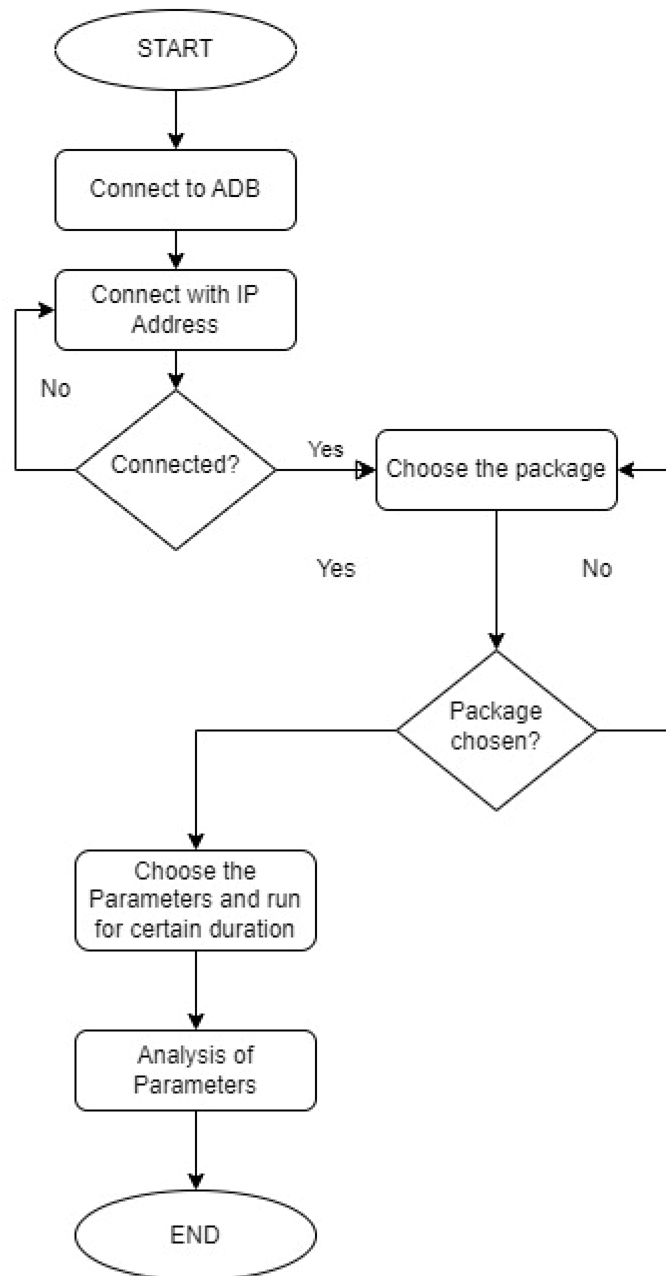



Fig. 4.2 Flowchart for App validation Tool

As shown in the flowchart above, in order to connect to a device's IP address, we must keep both devices connected to the same network. Once connected to the specified IP address, we can access the STB packages and select the desired package. After selecting the package, we must decide the parameters we want to test for the package for a specific period of time. At the conclusion of the validation, we will receive the output in graphical style along with one analysis report.

## 4.3 Parameters and Scenarios for App Validation

### 4.3.1 Six parameters are:

1. Memory Leak/ Memory Footprint
2. CPU Footprint
3. Number of Threads
4. OnTrimMemory Callbacks
5. CPU Footprint during Background Video Streaming

### 1. Memory Leak / Memory Footprint:

```
C:\Users\upadhyayh>adb shell
jade21:/ $ dumpsys meminfo com.netflix.ninja
Applications Memory Usage (in Kilobytes):
Uptime: 398346 Realtime: 398346

** MEMINFO in pid 4028 [com.netflix.ninja] **
                   Pss  Private  Private  SwapPss      Rss     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty    Total     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------   ------
  Native Heap     1426     1256      164    15060     1920    34784     7779    12177
  Dalvik Heap      454      244      192     3338     1156     5231     3924     1307
 Dalvik Other      306      264       36     2375      996
        Stack      344      116      228      772      348
       Ashmem        0        0        0        0        8
    Other dev       12        0       12        0      228
     .so mmap      314       44       16      145     8568
    .jar mmap      662        0        0        0    17808
    .apk mmap      924       80      844     1004      924
    .dex mmap      506        0      452     7036      960
    .oat mmap      232        0        0        0     8224
    .art mmap      369      124       84      557     7424
   Other mmap       11        8        0        0      564
      Unknown     1631     1140      472    27744     1868
        TOTAL    65222     3276     2500    58031    50996    40015    11703    13484

 App Summary
                       Pss(KB)                        Rss(KB)
                        ------                         ------
           Java Heap:      452                           8580
         Native Heap:     1256                           1920
                Code:     1436                          36528
               Stack:      116                            348
            Graphics:        0                              0
       Private Other:     2516
              System:    59446
             Unknown:                                    3620

           TOTAL PSS:    65222            TOTAL RSS:    50996       TOTAL SWAP PSS:    58031
```

Fig 4.2 Memory Leak/Memory Footprint

From above mentioned figure, we can see that memory information is given for Netflix, through which we can get memory profile. From this we must monitor Native Heap, Dalvik Heap, Java Heap for PSS and RSS as well as Total PSS. We can Access this data through ADB shell Dumpsys Meminfo.

Two key ideas in relation to how much memory Android apps use are memory leak and memory

footprint.

The term "memory footprint" describes how much memory an Android app needs to function. The size of the app's code, the number of features it provides, and the volume of data it processes all have an impact on how much memory the app uses. The app may utilize more system resources, such as CPU time and battery life, if it has a large memory footprint.

A memory leak, on the other hand, happens when a software allocates memory but forgets to release it after it is no longer required. This may cause the app to utilize more memory over time, which may potentially result in performance problems and app crashes. Programming flaws, poor resource management, and ineffective memory usage are just a few of the causes of memory leaks.

## 2. CPU Footprint

```
Tasks: 237 total,   1 running, 236 sleeping,   0 stopped,   0 zombie
 Mem:  2003656K total,  1936676K used,    66980K free,     2068K buffers
 Swap:  1502736K total,   174080K used,  1328656K free,   601512K cached
400%cpu   6%user   0%nice  13%sys 374%idle   0%iow   3%irq   3%sirq   0%host
  PID USER         PR  NI VIRT  RES  SHR S[%CPU] %MEM     TIME+ ARGS
 9866 shell        20   0  16M 3.6M 2.9M R 16.1   0.1   0:00.07 top -n 1 -b
 4164 u0_a73       16  -4 1.4G  99M  66M S  3.2   5.0   0:14.73 com.google.android.apps.mediashell
  232 system       20   0 229M 2.5M 2.4M S  3.2   0.1   0:03.76 android.hardware.gatekeeper@1.0-service.bcm
   11 root         20   0    0    0    0 I  3.2   0.0   0:01.97 [rcu_preempt]
 9863 root         20   0    0    0    0 I  0.0   0.0   0:00.00 [kworker/2:0-sock_diag_events]
 9848 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/3:1H]
 9784 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/u9:0]
 9768 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/0:2H-mmc_complete]
 9766 shell        20   0  16M 2.5M 1.9M S  0.0   0.1   0:00.00 sh
 9552 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/1:0H]
 9551 root          0 -20    0    0    0 I  0.0   0.0   0:00.01 [kworker/2:2H-kblockd]
 9498 u0_a80       20   0 1.2G  61M  37M S  0.0   3.1   0:00.41 com.android.settings.intelligence
 9466 system       20   0 1.3G 104M  73M S  0.0   5.3   0:04.61 com.android.tv.settings
 9406 root         20   0    0    0    0 I  0.0   0.0   0:00.01 [kworker/u8:4-kverityd]
 9405 root         20   0    0    0    0 I  0.0   0.0   0:00.08 [kworker/u8:3-kverityd]
 9387 root         20   0    0    0    0 I  0.0   0.0   0:00.07 [kworker/2:2-events]
 9357 root         20   0    0    0    0 I  0.0   0.0   0:00.06 [kworker/u8:1-kverityd]
 9338 root          0 -20    0    0    0 I  0.0   0.0   0:00.01 [kworker/u9:3-v3dclient completion]
 9328 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/3:0H]
 9320 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/0:0H-mmc_complete]
 9031 u0_a83       20   0 1.3G  91M  65M S  0.0   4.6   0:01.18 com.google.android.gms
 9022 u0_a81       20   0 1.3G  83M  57M S  0.0   4.2   0:00.56 com.android.vending:instant_app_installer
 9020 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/2:0H-kblockd]
 9018 root          0 -20    0    0    0 I  0.0   0.0   0:00.00 [kworker/1:2H]
 8986 root         20   0    0    0    0 I  0.0   0.0   0:00.05 [kworker/2:1-events]
 8884 root         20   0    0    0    0 I  0.0   0.0   0:00.13 [kworker/u8:2-kverityd]
 8492 u0_a82       20   0 1.2G  76M  50M S  0.0   3.8   0:01.00 com.google.android.tungsten.setupwraith
 8164 u0_a74       20   0 1.6G 108M  79M S  0.0   5.5   0:01.95 com.google.android.katniss:search
 8021 u0_a74       20   0 1.6G 117M  87M S  0.0   5.9   0:02.95 com.google.android.katniss:interactor
 7961 root          0 -20    0    0    0 I  0.0   0.0   0:00.08 [kworker/u9:1-v3dclient completion]
```

Fig. 4.4 CPU Footprint

From above figure, it is defined that at an average, CPU utilization for mediashell package is 3.2%. To collect the CPU usage information ADB shell’s Top -n 1 -b command was used. CPU utilization is important factor to test the system performance as we are working with multitasking and multithreading technologies.

The amount of processing power needed by an Android app to function on a device is referred to as the app's CPU footprint. The complexity of the app's code, the number of features it provides, and the volume of data it processes can

all have an impact on how much computational work an app must do, which is directly proportional to the CPU footprint.

Since the design and functionality of an Android app can greatly affect its CPU footprint, there is no set CPU footprint for Android apps. However, there are several approaches to lessen an Android app's CPU footprint, including:

Code optimization: By improving the code, programmers can limit the number of instructions that the CPU must execute, resulting in a smaller CPU footprint.

Keeping the program's background activity to a minimum: An app that stays active in the background even when it's not being used can burn up a lot of CPU power. By making the most of Android's activity lifecycle techniques and optimizing background processes, developers can reduce the amount of background activity in their apps.

lowering the app's memory consumption: A memory-hungry app can increase the CPU footprint by making the system swap memory to the disc.

### 3. Number of Threads

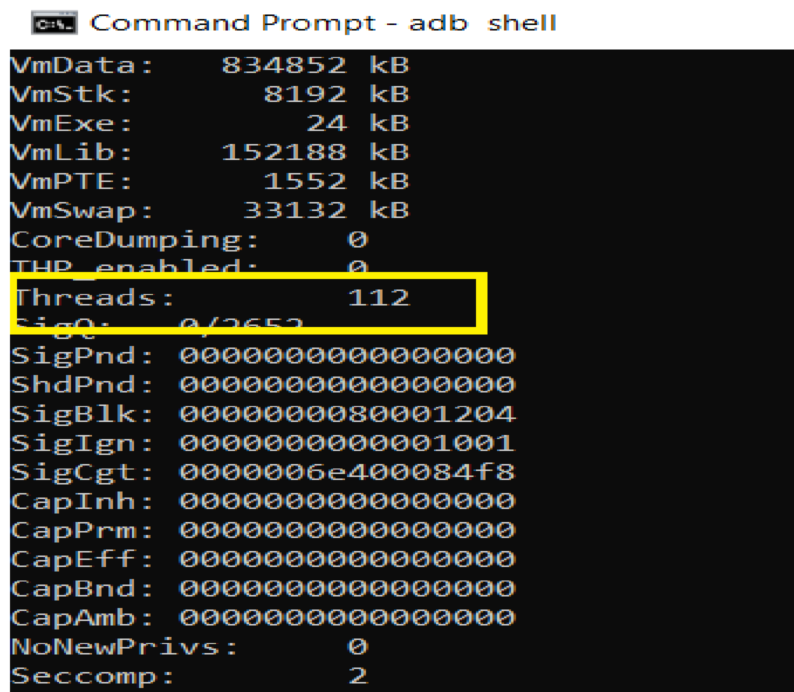



Fig. 4.5 Number of Threads

As mentioned in above figure, every time, the number of threads for each individual package must be verified. Although more threads equate to faster execution, running more than a predetermined number of threads concurrently can impact CPU use. Virtual memory is needed by each software thread for its stack and personal data structures. Time slicing hinders speed because it forces threads to compete for real memory, like how caches do. To collect the data related to threads we have used proc/status command of ADB shell.

An Android app's design and the tasks it must complete determine how many threads it needs. A thread is a quick process that may function independently and concurrently with other threads, enabling an application to carry out several activities at once.

Generally speaking, the majority of Android apps may run on just one thread, which is the main UI thread in charge of managing user interface interactions. However, certain applications need more threads to handle background

operations like network operations, file I/O, or sophisticated calculations.

Developers must consider several criteria, such as the following, when deciding the number of threads necessary for an Android app:

Task complexity: To avoid blocking the main UI thread, an app may need additional threads if it needs to complete difficult or time-consuming tasks.

The number of concurrent tasks: To prevent performance lag, a program may need numerous threads if it must carry out several tasks at once.

The available system resources: An app shouldn't create too many threads because doing so can result in the program using up too much memory and CPU time.

In order to manage threads in Android apps, it is generally advised to use a thread pool as this can help to improve performance and minimize resource exhaustion.

### 4. OnTrimMemory Callbacks

4.6 OnTrimMemory Callbacks

When there is not enough memory available, the Android system calls the onTrimMemory() function, which is a callback method. Beginning with Android API level 14 (Ice Cream Sandwich), this callback is accessible.

Apps may be asked to release some of their resources when the system is short on memory in order to clear up space. The onTrimMemory() method is used to alert apps about low memory conditions and to request that they relinquish any non-essential resources they own.

An integer parameter that specifies the degree of memory pressure the system is under is passed along

when the onTrimMemory() function is invoked. One of the following values can represent the degree of memory pressure:

TRIM_MEMORY_COMPLETE: The application needs to free up all unused resources since the system's memory is running low.

TRIM_MEMORY_MODERATE: The application needs to free up some non-essential resources since the system's memory is getting low.

TRIM_MEMORY_BACKGROUND: The application should free up whatever non-essential resources it can because the system's memory is getting low.

TRIM_MEMORY_UI_HIDDEN: The user cannot see the app's user interface (UI), thus it should free up any resources that are not required for state maintenance. The system's memory is at a critical low point, so any resources that may be released, including those that are vital, should be done so.

TRIM_MEMORY_RUNNING_LOW: The system's memory is running low; therefore, the software has to free up whatever non-essential resources it can.

TRIM_MEMORY_RUNNING_MODERATE: The application needs to release some non-essential resources since the system's memory is running low.

The app should release any resources that are not required to retain its state when the onTrimMemory() method is called. This may entail terminating background processes, releasing cached data, and releasing unused resources. An app can assist the system by releasing resources in response to the onTrimMemory() callback, allowing the system to free up memory and enhance speed.

**5. CPU usage during Background Video Streaming**

Since we may stream live video straight to such STBs, this parameter testing is only possible with secured STBs. It will monitor the launcher's CPU usage while also broadcasting live video in the background. Total CPU Usage can be considered as the difference between total CPU usage and Idle CPU usage.

The CPU consumption for Android TV apps while streaming video in the background varies based on a number of variables, including the particular program, the video codec being used, the hardware of the device, and any additional processing being done by the app.

The CPU use of an Android TV app should typically be lower when it is streaming a video in the background than when it is actively playing the video on the screen. This is because rendering the video on the display requires no additional processing from the device.

The majority of the video decoding work is often offloaded to specialized video processing units thanks to hardware acceleration found in modern Android handsets and Android TV platforms. This decreases CPU utilization, even in the background, while playing back videos.

Developers can use hardware acceleration, design effective video decoding methods, and make sure that other background operations aren't unduly using system resources to optimize CPU use while background video streaming in Android TV apps.

### 4.3.2 Two testing Scenarios are:

1. Quick Navigation Script
2. Memory Footprint during App Switching

#### 1. Memory Leak during App Switching

The system must allow users to access many apps simultaneously and moving between them must be seamless. The likelihood of memory leaking is high if a memory profiler shows a sudden increase in memory usage. Because of this, it's crucial to determine whether RAM is being lost when switching between apps. For Memory Leak data collection, we have used Dumpsys command of ADB shell. Android apps may experience memory leaks if adequate memory management procedures are not followed, notably when moving between apps. To avoid memory leaks and excessive memory utilization, it's critical that resources and memory needed by the preceding app be appropriately released when switching apps.

Here are a few typical reasons why Android memory leaks happen when moving between apps:

Static references: If an app maintains references to resources or objects that are never supposed to be garbage collected, it could result in memory leaks. Make sure that when static references are no longer required, they are correctly nullified or cleared.

Memory leaks can be caused by holding onto references to Android Context objects (such Activity or Context Wrapper) after they have outlived their intended usefulness. Context object references should not be kept indefinitely, especially within static or singleton instances.

Callbacks and listeners: Failure to deregister callbacks or listeners when a component (such as an Activity) is deleted might result in leaks. To free up related resources, make sure to deregister any callbacks or listeners in the relevant lifecycle functions (such as onDestroy()).

huge object allocations: Allocating huge objects, like bitmaps or other demanding resources, without properly releasing them might result in excessive memory utilization and possible leaks. Utilize strategies like resource release or bitmap recycling when no longer required.

Improper management of background tasks: Background tasks might continue to use resources and memory even when the app is not active if they are started during app switching and are not properly handled or stopped. Make sure that background processes are properly controlled and terminated as necessary.

Following Android's memory management best practices is crucial to preventing memory leaks when moving between apps:

Retain objects only if necessary and avoid needless object retention.
Utilize the proper lifecycle methods to properly release resources, deregister callbacks, and cancel tasks.
To find and fix memory leaks during development and testing, use tools like the Android Profiler or Memory Leak Detection libraries.

### 2. Quick Navigation Scripts

The fundamental necessity for any Set-Top Box Launcher is navigation. Quick navigation scripts measure how quickly the launcher functions or how long it takes to move around the launcher screen. It

evaluates the Set-Top Box's responsiveness. Any Set-Top Box (STB) launcher must have navigation as a basic component. Users can interact with the STB and access different programs, content, and settings using the launcher, which serves as its user interface. The effectiveness of the launcher's response to user input and the ease with which users can move across the launcher screen are evaluated using rapid navigation scripts.

Any Set-Top Box (STB) launcher must have navigation as a basic component. Users can interact with the STB and access different programs, content, and settings using the launcher, which serves as its user interface. The effectiveness of the launcher's response to user input and the ease with which users can move across the launcher screen are evaluated using rapid navigation scripts.

A smooth and seamless user experience depends on the STB launcher's quickness. Users anticipate swift, seamless motions through the launcher with no discernible pauses or delays. By monitoring how quickly interface elements and transitions are presented and how quickly the STB responds to user input, quick navigation scripts can assess how responsive the launcher is performing.

These scripts often simulate user events, including button presses or directional movements, and track how long it takes the STB launcher to carry out the related tasks. The scripts can evaluate a number of navigational elements, including the response time when choosing and launching apps, the time it takes to transition between launcher screens, and the speed at which lists, or menus scroll.

Quick navigation testing can help designers and producers find any performance hiccups or bottlenecks in the STB launcher that might impede easy navigation. This enables them to enhance the launcher's functionality, increase responsiveness, and give users a better experience.

All things considered, quick navigation scripts are useful resources for assessing the responsiveness and effectiveness of a Set-Top Box launcher, assisting in making sure that users can easily navigate through the interface and have a smooth contact with their STB.

## 4.3 Testcase Results:

We tested the **X-set top box** once to verify the tool's output. We carried out testing for an hour and gathered the data for it. It is a good idea to gather test data for the X-set top box throughout a one-hour run in order to assess its performance and spot any problems or areas that require development. We have taken the following actions to fully utilize the test results:

Analyze the information gathered: Examine the test results and compile pertinent information, including CPU utilization, memory usage, response times, and any other metrics that were tracked throughout the test. For simpler analysis, arrange the data in an organized manner.

Find performance patterns: Scan the data for any discernible trends or patterns. Find any performance issues, irregularities, or areas that need more research. For instance, increases in CPU or memory usage may be a sign of trouble.

Compare test results to performance benchmarks: If you've defined performance benchmarks or goals for the X-set top box, compare the test results to those targets. This aids in determining if the performance satisfies the intended criteria or calls for improvement.

Take a closer look at any outliers or anomalies in the data and investigate them. These might point to potential performance problems or strange behavior that needs more research. Investigate the circumstances under which these outliers arose and make an effort to comprehend the underlying causes.

Document observations, conclusions, and advice. Recommendations should be based on the test results. Keep track of any performance problems you run across, any recommendations you have for enhancements, and any prospective adjustments you might make to improve the X-set top box's performance.

Continue with extra testing or optimizations: In light of your analysis, you can think about running additional tests to confirm the findings or to assess the success of any adjustments you made to solve performance issues.

The graphical presentation of all test results is described in the section below. Additionally, a PDF version of the final analysis report has been created.

(1) Memory Leak/Memory Footprint Test:

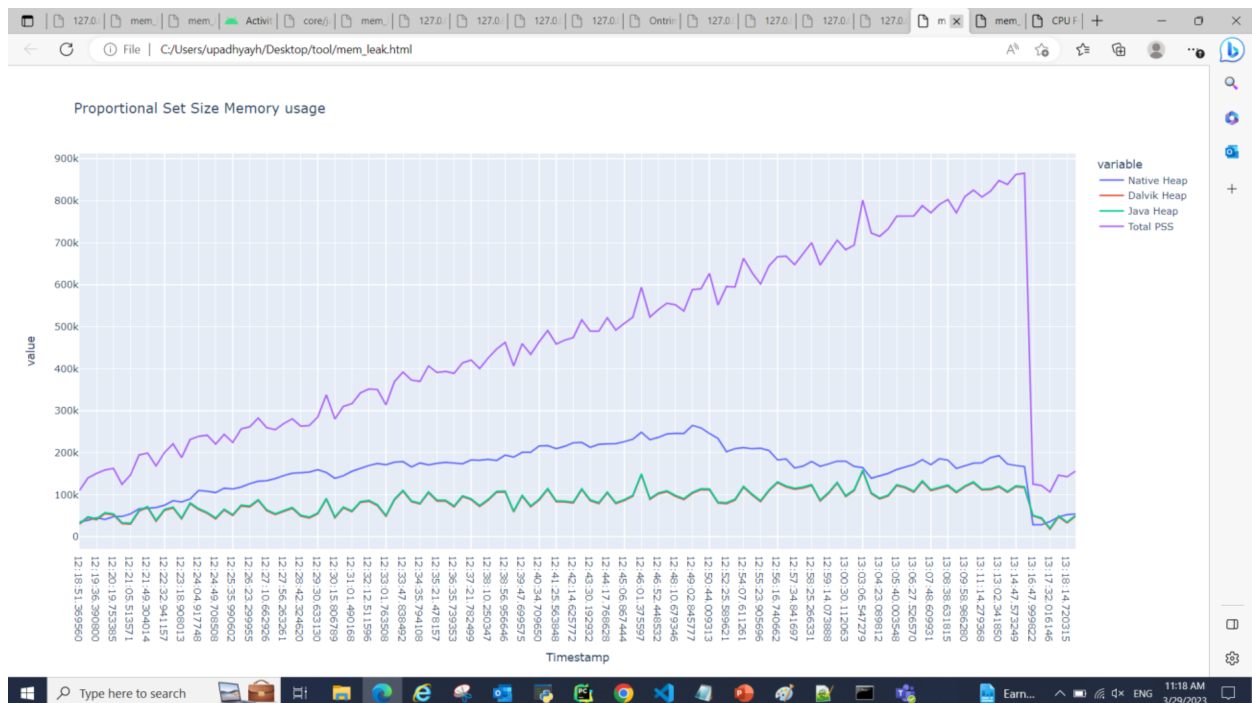



Fig. 4.6 App Validation Tool Output for Memory Leak Test of X-Set Top Box

This is the memory leak detection graph, as seen in the aforementioned figure, and it allows us to determine whether a device has any memory leaks that have been discovered using this tool. We have tracked the total proportional set size, or PSS, of memory in order to find memory leaks. The Proportional Set Size (PSS) metric is used in Android to gauge how much memory an application or process is using. It indicates the piece of memory that is assigned to a particular process, including both shared and private memory (memory that is utilized by many processes concurrently).

Because it takes shared memory into account, the PSS value—which is an estimation—offers a more realistic picture of memory utilization than the raw private memory usage (RSS, or Resident Set Size). When several processes share resources, such system libraries or other shared components, it is especially helpful.

The shared memory is divided by the quantity of processes sharing it to produce the PSS value, which is then added to the private memory. It offers a more equitable distribution of memory consumption among the associated processes. To track and retrieve memory-related data, including the PSS values of active processes, Android offers the Android Profiler and command-line applications dumpsys and procrank.

Finding memory-intensive components or potential memory leaks in an application can be facilitated by tracking and comprehending the PSS values of processes. It enables developers to prioritize memory optimization efforts, optimize memory utilization and guarantee effective memory management in Android applications.

(2) CPU Footprint Test:

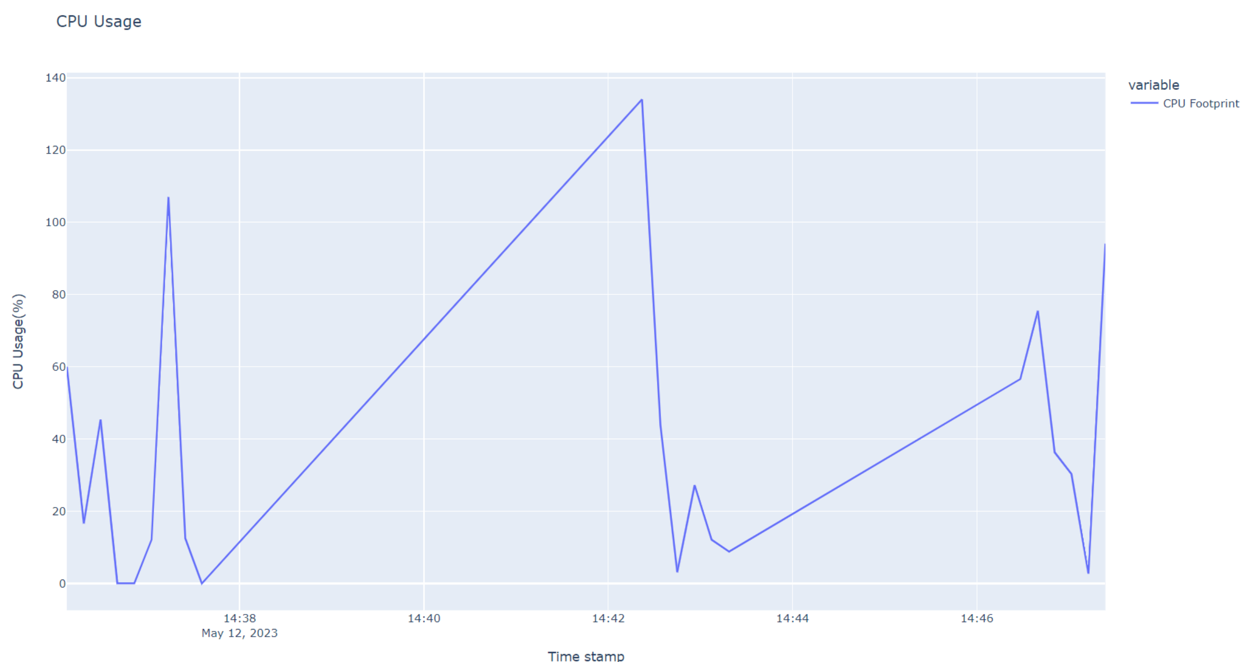



Fig. 4.7 App Validation Tool Output for CPU Footprint Test for X-Set Top Box

As mentioned in above figure, we have used top command to measure the CPU usage through this tool. In Android, CPU use is a term used to describe how much processing power the device's central processing unit (CPU) uses while carrying out various operations and processes. Monitoring CPU consumption can assist in locating performance bottlenecks, streamlining resource distribution, and ensuring effective use of the device's processing power.

The Android Debug Bridge program offers command-line access to a number of device features, including details about the CPU. The adb shell top and adb shell dumpsys cpuinfo commands can be used to get information about the CPU use of active processes.

Third-party monitoring libraries: These libraries, which offer further information on CPU consumption and performance profiling, include ACRA (Application Crash Reports for Android), LeakCanary, and ProcessCpuTracker.

It's crucial to take the measurement's context into account while analyzing CPU utilization. The efficiency of the software being run, the complexity of the processes that are active, background duties, and the hardware capabilities of the device can all affect CPU utilization.

Monitoring CPU usage can assist in identifying situations when high CPU usage may result in degraded performance, higher battery consumption, or overheating. Developers can optimize their apps to use less CPU, respond more quickly, and improve overall device performance by recognizing certain instances.

(3) OnTrimMemory Callbacks:

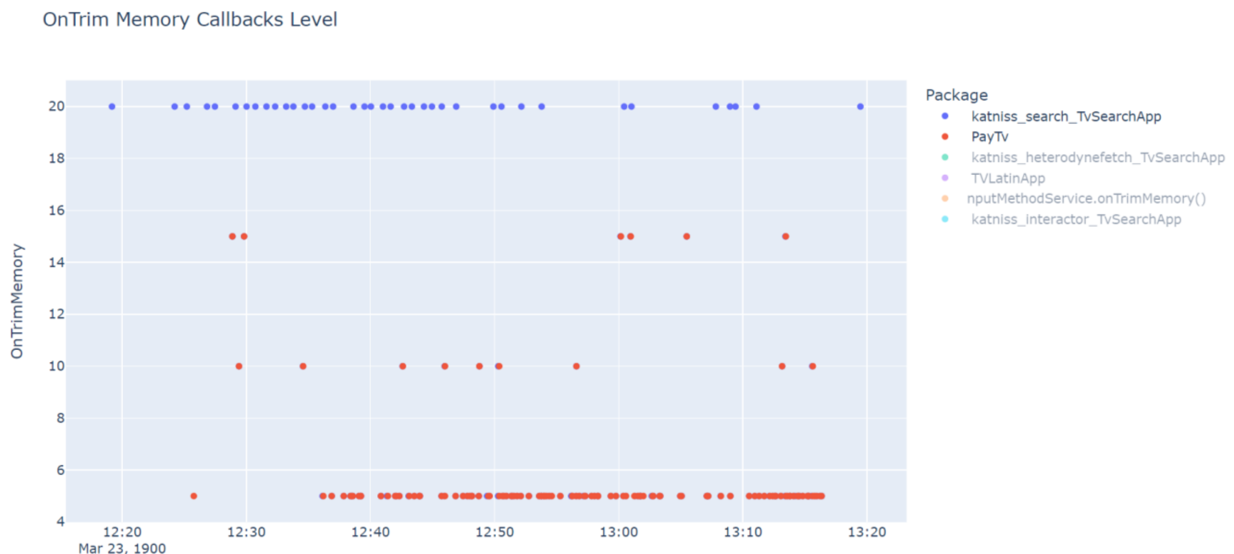



Fig. 4.8 App Validation Tool Output for OnTrimMemory Callbacks Test for X-Set Top Box

Data from OnTrimMemory Callbacks is plotted, as seen in the above figure. The adb logcat command is used to collect this data. The onTrimMemory() callback, which is a part of Android's ComponentCallbacks2 interface, alerts an application when the system's total memory use is approaching a critical level. To assist save memory and boost system performance, this callback enables the program to react and remove any unused resources. Levels and constants for each level of OntrimMemory Callbacks are mentioned below:

| OnTrimMemory Levels | Constants |
|---|---|
| TRIM_MEMORY_RUNNING_MODERATE | 5 |
| TRIM_MEMORY_RUNNING_LOW | 10 |
| TRIM_MEMORY_RUNNING_CRITICAL | 15 |

| TRIM_MEMORY_UI_HIDDEN | 20 |
|---|---|
| TRIM_MEMORY_BACKGROUND | 40 |
| TRIM_MEMORY_MODERATE | 60 |
| TRIM_MEMORY_COMPLETE | 80 |

Table.4.1 OnTrimMemory Callbacks levels

(4) Memory Footprint during App Switching:

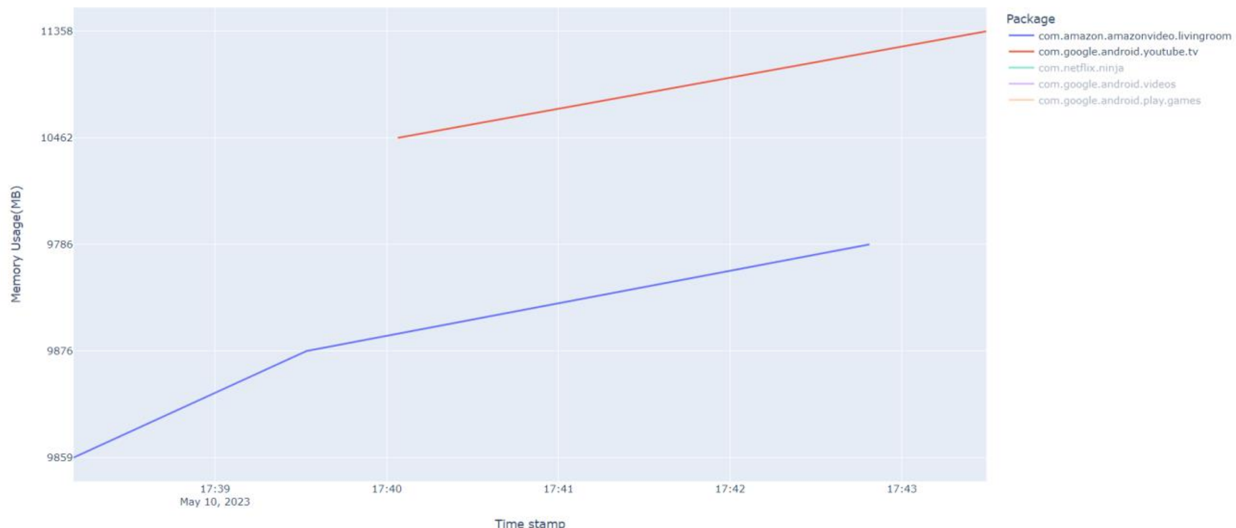



Fig. 4.10 App Validation Tool Output for App Switching Scenario Test for X-Set Top Box

We have taken into consideration five apps for the App Switching Scenario, including PrimeVideo, Netflix, Youtube, Google Games, and Google Videos. With this scenario, we are able to simultaneously and with specified intervals gather all memory-related data for each individual app. On an Android device, switching between several applications or tasks is referred to as "app switching." The operating system controls the transition when a user moves from one app to another, making the process seamless. An overview of Android's app switching functionality is provided below:

Launching a new app: The operating system starts the launch procedure for the selected app when a user chooses another one from the device's home screen, app drawer, or recent applications list. It could entail launching the app, initializing the user interface, and carrying out any setup procedures that are required.
The operating system signals to the presently running app that it is being halted in a series of lifecycle callbacks before switching to the new app. These callbacks, such as onPause() and onStop(), can be used by the app to save its state, release resources, or carry out any necessary cleanup.

Animation of transition: Android shows an animation that seamlessly switches from the currently open app to the just launched app in order to create a visual transition between apps. Depending on the device and version of Android, the animation can have fading, sliding, or zooming effects.
Resuming the newly opened app: After the transition animation is finished, the app comes to

the foreground. The new app receives lifecycle callbacks from the operating system (such as onStart() and onResume()) informing it that it is now active and viewable by the user. The program has the ability to reset its state and resume any active tasks.
Handling background apps: While the new app is active, the operating system handles any background apps that were already open. In order to ensure effective system performance and preserve battery life, it might modify resource allocations, pause background tasks, or carry out other optimizations.
Returning to the previous app: The user can utilize the navigation buttons, gestures, or recent applications list on the device to return to the previous app that was open. The operating system resumes the preceding app's operation by bringing it back to the foreground and calling the necessary lifecycle callbacks.

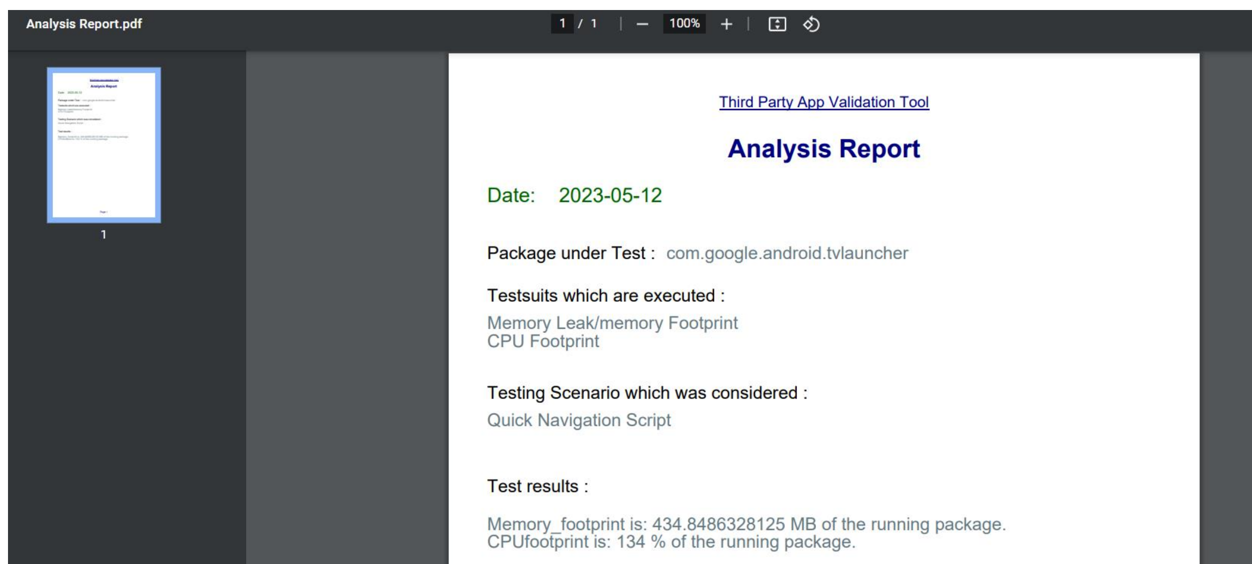
## 4.4 Final Result:

At the conclusion of the tool's execution, an analysis report and all of the executed parameters' graphs will be saved in a graph folder as an HTML Index file. The package

name for which the validation test was run, the scenario that was taken into consideration during the test, the parameters that were taken into consideration for validation, and the results for each parameter are all included in this analysis file.

Fig. 4.11 App Validation Tool Graph Index HTML File

Using libraries like Plotly, Matplotlib, or Bokeh, one can generate an HTML file in Python that contains 3–4 graphs. These libraries offer tools for creating different kinds of graphs and save them as HTML files. The desired graphs were made using the Plotly package. There are many different types of charts available, including bar, line, scatter, pie, etc. Set the information, labels, and preferences unique to each graph.The graphs were saved as an HTML file using the offline.plot() function from Plotly's offline module. This will create a separate HTML file for every graph.


Analysis Report.pdf

Third Party App Validation Tool

Analysis Report

Date: 2023-05-12

Package under Test : com.google.android.tvlauncher

Testsuits which are executed :

Memory Leak/memory Footprint
CPU Footprint

Testing Scenario which was considered :

Quick Navigation Script

Test results :

Memory_footprint is: 434.8486328125 MB of the running package.
CPUfootprint is: 134 % of the running package.

Fig. 4.12 App Validation Tool Analysis Report

A well-liked Python library for producing PDF files is the fpdf package. It offers a simple user interface for creating PDF files containing text, graphics, and several formatting choices. Here is a quick guide on how to utilize the fpdf library:
Using ‘pip install fpdf’ we can install fpdf library then we have to import fpdf class from fpdf module using command from fpdf import FPDF. To define athe PDF document, we have created a subclass of the FPDF class. To change the PDF's header, footer, and main content, respectively, override the corresponding methods in header(), footer(), and body().
To add text, photos, and other components to the PDF, use the different methods given by the FPDF class inside the body() method. Several typical techniques include:

set_font(): Changes the font's style, size, and type.
cell(): Insert a text-filled cell.
image(): Adds a picture to the PDF.
Set the current location for text insertion using set_xy().
ln(): Go to the following line.
Add a new page to the PDF with add_page().

Python programmers can easily generate PDF documents by using the fpdf package. For more complex usage and features, consult the library's documentation and examples.

# Chapter 5

# Conclusion

## 5.1 Conclusion

Hence it is concluded that, to ensure that your system meets the necessary service levels and provides a positive user experience, you should run performance tests. Your applications will highlight performance, reliability, and scalability improvements that need to be made before they are put into production. Applications that are released to the public before they have been thoroughly tested may contain several problems that, in certain cases, might permanently damage a brand's reputation.

Conducting performance tests is essential to ensuring that your system fulfils the necessary

service levels and offers a satisfying user experience. Before deploying your applications in a production environment, you can find out what has to be improved in terms of performance, reliability, and scalability by running performance tests on them. Without adequate testing, releasing applications to the public can lead to a number of problems that might harm a brand's reputation.
Applications must undergo thorough performance testing in order to become popular, profitable, and useful. It enables you to spot any performance-related problems and fix them, guaranteeing that your apps can manage the anticipated workload and provide a positive user experience. You may identify bottlenecks, improve resource usage, and fine-tune your applications for optimum performance by conducting performance tests.

Implementing a continuous optimization performance testing technique is essential for the success of your overall digital strategy, even if solving production performance problems can be expensive and time-consuming. Delivering a top-notch user experience, upholding customer contentment, and safeguarding your brand reputation can all be accomplished with the aid of routine performance testing and optimization throughout the lifecycle of your apps.

You can proactively deal with performance issues, find areas for improvement, and make sure that your apps run at their best under varied circumstances by investing in performance testing and optimization. You can provide dependable and high-performing applications thanks to this proactive strategy, which ultimately helps your digital initiatives succeed and remain competitive.

Even though correcting production performance issues can be quite expensive, implementing a continuous optimization performance testing technique is crucial to the realization of a successful overall digital strategy.

# References


[1] Xue, D. 2015. Design and Implementation mobile application performance monitoring system based on Android. Xidian University of Electronic Science and Technology.

[2] Chenhui Xie, Jian Zhou, ShanShan Li ,Lu Ying Jia , The Design and Implementation of Mobile Monitring System of Transmitting Station based on Android Platform,[D] 2012 International Conference on Mechnical and Electronics Engineering, Beijing,2012.

[3] Qi Luo, A. N. 2017. FOREPOST:findind performance problems automatically with feedback directed learning software testing. Empir Software Eng (2017)22.6.

[4] Wen, Design automation software for Android test system performance and implementation Of [D], 2016.

[5] Wei, Explore the development of automated software testing tools under Android platform [J] , ( 2015) 30 (2): 155.

[6] Lukun, Research and Implementation Android smart phone performance automated test system [D] Beijing, 2017.

[7] Yun, performance of key technology of software test platform research and application of [D] Beijing, 2017.

[8] Xingnong, Android-based APP automated test platform design and implementation of [D] Dalian , 2016.

[9] Warren, I. and Meads, A. (2017) Towards a Technology Agnostic Approach to Developing Mobile Applications and Services. Journal of Soft-ware Engineering and Applications, 10, 500-528.

[10] Ashwaq A.Alotaibi, R. J. "Novel Framework for Automation Testing of Mobile Application using Appium". International Journal of Modren Education annd Computer Science (IJMECS),Vol.9,No.2,pp.34-40, 34-40.

[11] iTest. 2018. Testing Tool https://soft.shouji.com.cn/down/29068.html

[12] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of GT.(n.d.).TestingToolRetrievedfromhttps://blog.csdn.net/harryzzz/arti cle/details/81381920

[13] 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT). (2019).

[14] Performance Testing of Android TV Applications Using Robotium and ADB Commands" Authors: Sagarika Sahoo and Tapan Kumar Panda Published in: 2020 2nd International Conference on Computing, Communication, and Security (ICCCS). (n.d.).

[15] Performance Testing and Analysis of Android TV Applications using Monkey and ADB Commands
2020 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)

[16] 2016 IEEE 15th International Symposium on Parallel and Distributed Computing (ISPDC) Han R, Wang Y, Wang W

[18] Performance Evaluation of Android TV Applications" Authors: Hyeonjeong Jo
Ryu S, Kim M, Ko B

[19] Performance Testing of Android TV Applications using Monkey and AspectJ" Author: Shweta Jain

[20] A Performance Testing Framework for Android TV Applications
Han R, Wang Y, Wang W
2016 International Conference on Networking, Architecture, and Storage

[21] Performance Evaluation of Android TV Systems
Chen YH, Chen SW
IEEE International Symposium on Circuits and Systems (ISCAS), 2018

[22] Performance Testing of Android TV Apps: A Comprehensive Study" Authors: Jacek Dominiak and Piotr Gawron Published in: 2019 6th IEEE International Conference on Data Science and Advanced Analytics (DSAA)

[23] Performance Testing of Android TV Apps Using a Cloud-Based Testing Platform
2016 International Symposium on Consumer Electronics (ISCE)

[24] Performance Analysis and Testing of Android TV Apps in an IoT Environment" Authors: Himanshu Gahlot, Mohit Yadav, and Rahul Johari Published in: 2018 International Conference on Communication and Signal Processing

[25] Performance Testing and Analysis of Android TV Apps" Authors: Prasanta K. Jana and Kalyani P. Poojary Published in: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)

[26] Performance Testing and Analysis of Android TV Apps" Authors: Prasanta K. Jana and Kalyani P. Poojary Published in: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)

[27] Performance Testing of Android TV Applications: An Industrial Case Study
Toro M, Cubo J
2017 IEEE International Conference on Software Testing, Verification and Validation Workshops

[28] 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications

[29] Performance Testing of Android TV Applications on Emulated and Physical Devices" Authors: Mika Koskela, Teemu Kanstrén, and Casper Lassenius

[30] Performance Testing of Android TV Applications Using Robot Framework
IEEE 4th International Conference on Big Data Intelligence and Computing (DataCom), 2018.

[31] Authors: Wei Jin, Shangping Ren, and Yuke Zhang Published in
14th IEEE International Conference on Automatic Face & Gesture Recognition (FG), 2019

[32] Performance Evaluation and Optimization of Android-based Interactive TV Applications
2012 IEEE International Conference on Multimedia and Expo (ICME)

[33] Performance Testing and Evaluation of Android TV Applications" Authors: Muthumariappan Ramkumar and Sundararajan Rajamanickam Published in: 2018 International Conference on Advanced Computation and Telecommunication (ICACAT)

[34] Performance Testing of Android TV Applications Based on Resource Usage and User Experience
2015 International Conference on Electronics Technology (ICET)

[35] Performance Testing of Android TV Applications using Monkey and Robolectric" Authors: Abhilash Kulkarni and Ravindra Gad