International Journal of Analysis and Applications
Volume 19, Number 4 (2021), 587-603
URL: https://doi.org/10.28924/2291-8639
DOI: 10.28924/2291-8639-19-2021-587

ANALYSIS OF PARAMETERS USED FOR MEASURING PERFORMANCE OF MOBILE APPLICATIONS

DURGA SOWJANYA KOLLURU[1,*], P. BHASKARA REDDY[2]

[1]*Department of Electronics and Communication Engineering, MLR Institute of Technology, Hyderabad, India*

[2]*Director, Holy Mary Institute of Technology and science, Hyderabad, India*

**Corresponding author:* k.durgasowjanya@gmail.com

ABSTRACT. To make mobile application more reliable, performance is most important parameter. Resource management plays crucial role inside development of mobile application. Careful attention should be required for number of non-functional requirements to achieve ultimate goal of mobile application development. Mobile application and its impact on market are analyzed by performance evaluation process. For getting better performance, efficient utilization of physical resources is required. These resources provide actionable information to application developer for optimizing performance and increases efficiency during development cycle. Due to limited processing power and memory resources, keep hardware components always in running states. But slow applications drain batteries of their devices. To maximize utilization of resources, developers should consider hardware limitations too which make an effort to optimize performance and efficiency of application. This paper presents a qualitative performance analysis of mobile applications. This paper describes factors considered for analyzing performance and corrective measures to achieve better performance.

Received May 14th, 2021; accepted June 7th, 2021; published June 24th, 2021.
2010 Mathematics Subject Classification. 68N30.
Keywords. mobile application; resource management; performance evaluation; optimizing performance.

1. INTRODUCTION

Popularity of mobile applications has fabulous response since last decade. Development of mobile applications witnessed change of devices from land line telephones to high-end smart phones. Smart phones are driven by powerful operating system that allows users to add and remove applications. Business organizations prepare software developers for developing independent application with life cycle which become redundant and expensive [1].

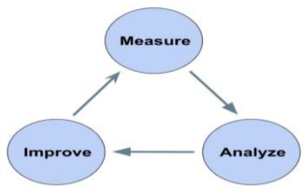


Fig.1. Performance evaluation

Proposed paper presents route to analyze performance of mobile application for android operating system. Figure 1 shows the evaluation process of performance for mobile application. Various parameters are raised by the utilization of resources in mobile application during development time and runtime. Then it describes tips for optimizing performance of mobile application. Proposed paper is organized into different sections as section 2 for related work, section 3 for overview of parameters affecting performance of mobile application, section 4 describes tips to improve performance and section 4 presents results and discussion and section 5 states conclusion.

2. LITERATURE SURVEY

Existing research focused on the analysis of performance optimization by evaluation and comparison of qualitative properties [2]. State-of-art for mobile application development addresses many challenges and opportunities for developers [3]. It mainly focuses on user-perceived application performance. Alternative methods for qualitative analysis are presented in [6], [7] and [8]. Demo application with graphical user interface is presented in [9]. It describes variious parameters like CPU usage, memory usage and battery consumption. Native interface is not used in this method to evaluate behavior of application. It presents study on in-depth parameters for mobile application performance as per quantitative analysis. Therefore, contribution of existing work for proposed paper is explained in three phases. First phase explains CPUusage, memory and battery usage, multiple response time as they facilitate major impact on user experience. Second phase compares java implementations with native implementations for android operating system. Third phase analyzes all parameters of development cycle based on user experience.

3. PARAMETERS EFFECTING PERFORMANCE

Performance of mobile application can be determined with many parameters. Execution time, memory usage and battery consumption are parameters yields for analyzing performance [9]. Application's execution time and launch time, both are collectively called as latency. Analysis of performance is based on various parameters that are measured and evaluated. Parameters considered during development time and run times are described in section 3.1 and 3.2. These parameters specify effect on performance of each parameter relevant to overall performance of application.

3.1. Parameters effected during development of an application:

3.1.1. Threads:

Managing threads determines stable and consistently high-performing application. 90% of performance issues can be solved by effective working with threads. There are two main types of threads which effects performance of an application: main thread, also called UI Thread, and the background thread. Figure 2 shows working of UI thread and its effect on performance. Any changes that made to the UI block the main thread, and this won't be assigned to another method or call until the current task is finished and it follows the FIFO method.

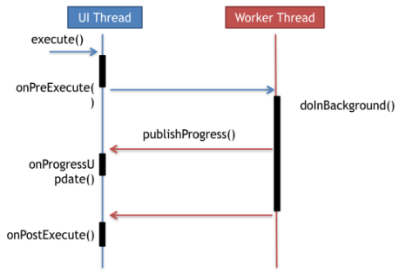


Fig.2. Working of UI threads

3.1.2. Managing memory leaks:

Most important problems that appeared in mobile application are memory leaks. A memory leak is the problem occurred when the garbage collector is not able to collect the allocated memory. For example: If the reference is hold in our code to an unused Activity, all of Activity's layout and, in turn, all of its views and everything else that Activity holds [5].

Main Causes of memory leaks are as follows:

3.1.2.1. Static References:

Keeping static references in the application degrades the performance. An improper reference causes both activities and fragments with a reduced lifecycle. With static reference, an activity or fragment will not be garbage collected.

3.1.2.2. Unregistered events and handlers:

It is to be noticed that lot of times developers don't unregister events and handlers. This causes application could leak.

3.1.3. Removing deprecated APIs:

Deprecation essentially references when an API is removed and, it could be that one or two days after a major release, the application might not work on certain devices. To make things even worse, sometimes application is not dependent on an older version of a library, no way is detected to update both APIs and tools.

The major problem by not using deprecated APIs is that after the API is removed from the Android system, the developed application won't be able to find the method and will crash with a runtime exception. Similarly, always make sure that the libraries are maintained by the providers otherwise the application get stuck in a pothole without a way out.

3.1.4. Compactness:

Compactness is determined by user experience in downloading application over mobile internet connection.

3.1.5. Disk Space:

There are two methods used to measure disk space. One method is considered by space taken for installing application on device. It is use ful for low end devices with limited resources. Another method is verified by space defined for apk size. It is downloadable installer for application in android OS.

3.1.6. Without using static final key:

It is observed that many developers make the mistake as they declare static constants without the final key. Compiler executes initializer method called <clinit> when class is used first time. These values are accessed later with field lookups. So, from the above example, it is understood that when a static field without the final keyword was declared, that variable is considered as a field and not as a constant.

3.2. Parameters effected during run time execution of an application:

3.2.1. Response time:

Response time gains importance regarding user experience. Figure 3 shows graphical representation of response time during application running. Response time is calculated by three different time measures listed as start time, pause time, resume time. Start time is time taken by start of application i.e. time taken from tapping application icon to displaying first screen of application [8]. Resume time is time taken by application moving from background to foreground and vice versa for pause time.

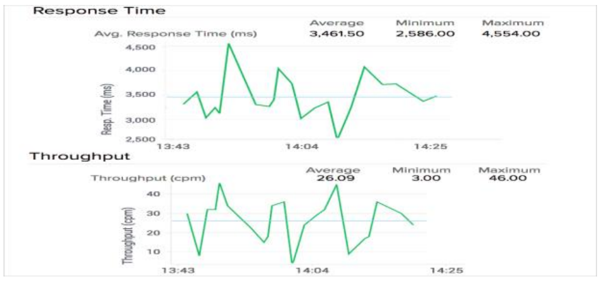
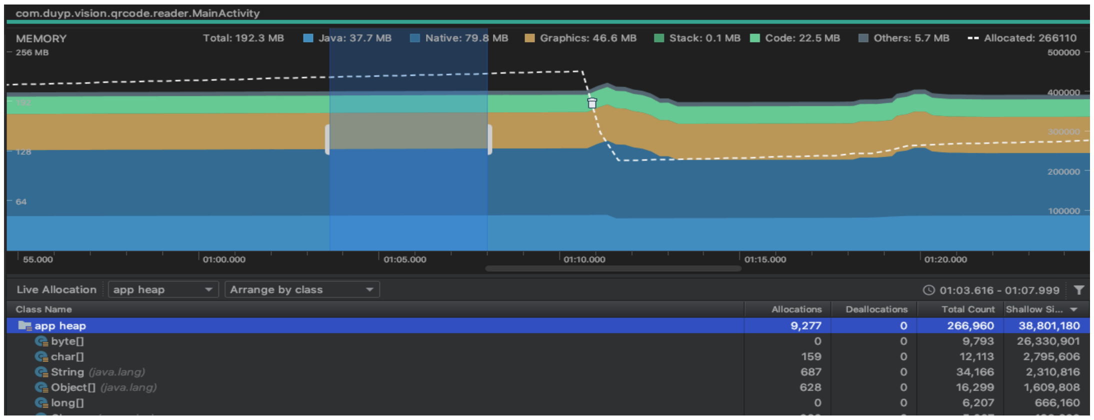


Fig.3. Graphical representation of response time

3.2.2. Memory Usage:

Memory usage defines amount of RAM allocated for application. Figure 4 shows memory usage of application on profiler. Event timeline, which shows activity states, user input events, and screen rotation events as indicated by x-axis. Memory usage by each memory category being shon by stacked graph as indicated by y-axis on left and color key at top. Dashed line indicates number of allocated objects as indicated by y-axis on right. Each garbage collection event is shown by icon. In background, it measures RAM memory [9].



Class Name	Allocations	Deallocations	Total Count	Shallow Si...
app heap	9,277	0	266,960	38,801,180
byte[]	0	0	9,793	26,330,901
char[]	159	0	12,113	2,795,606
String (java.lang)	687	0	34,166	2,310,816
Object[] (java.lang)	628	0	16,299	1,609,808
long[]	0	0	6,207	666,160

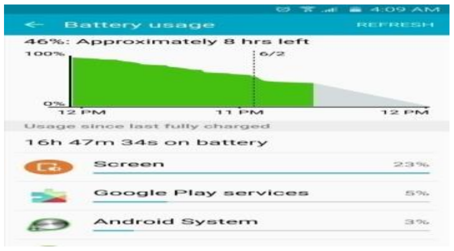
Fig.4. Memory usage on profiler

3.2.3. CPU Usage:

CPU usage measures percentage amount of work done by CPU between two specific time intervals. It depends on operating point of CPU determined by frequency. So, care should be taken to normalize CPU usage for maximum operating frequency which gives comfortable user experience.

3.2.4. Battery consumption [18]:

For any application, battery consumption is important. Figure 5 shows battery consumption of application. If device batteries are drained, no user prefers to use applications. Efficient battery usage of application reasons user experience.



46%: Approximately 8 hrs left

Usage since last fully charged

16h 47m 34s on battery

Screen	23%
Google Play services	5%
Android System	3%

Fig.5. Battery usage

3.2.5. Application Launching Time [25]:

Application launching time is described by one of two states. It is described by time taken for application visible to user. The two states are:

- Cold state
- Warm state

3.2.5.1. Cold State:

Cold state refers to state of application starting from scratch launched for first time after device boot or reload from system.

Cold state performs three tasks. They are:

- Loading and launching application.
- Displaying blank window immediately after launch.
- Creating application window.

3.2.5.2. Warm State:

Warm state provides lower-overhead than cold state. Warm state defines state of activity moves from backgroung to foreground. If all activities are resided in memory, application can avoid object initialisation, layout inflation, and rendering.

3.2.6. Latency :

Latency is the time between moments that request data till the moment that actually start receiving it.

3.2.7. Slow Rendering:

Slow Rendering is common performance issue for mobile applications. Application screen should be updated after every 16ms by attempting redrawn of application activities after every 16ms. If the frame is not completed within 16ms, it is called as dropped frame. Figure 6 shows process of slow rendering. If application takes 21.763 ms to draw new picture, system was not ready because time exceeds 16ms. Therefore, user can saw new picture with refreshed after 32 ms instead of 16ms. One dropped frame in application causes animation will not be shown with smooth start. Rendering is defined as how many times application runs in smooth process at a constant rate of 60 FPS without any dropped or delayed frame.

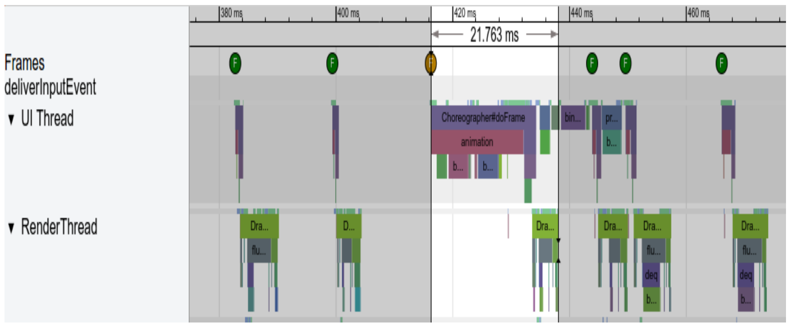


Fig.6. Slow rendering process

3.2.8. Layouts:

Layouts are fundamental part for any Android application, and they directly affect user experience. Inadequate implementation of application layout will lead to memory hungry with slow UI. Efficient layout structure adds and initializes widget and layout to application. Nested instances of linear layout produce deep view hierarchy. Nesting linear layout is defined by layout_weight parameter. For list view or grid view, layout is inflated repeatedly.

4. OPTIMIZING PERFORMANCE OF ANDROID MOBILE APPLICATION

Mobile application is developed by giving lot of freedom to developers. After developing application, owner will accesses ever-growing user base. For that developers face many challenges in application development. Inventing many versions make developers hard to keep up all updates in application development. It creates big challenge since thousands of devices are running on same OS with different versions. Application performance can be improved by keeping few things in mind. Following are the tips or remedies which optimize application performance.

4.1. Never do following things on UI thread:

- Load images or streams
- Parse a JSON
- Access a local or remote server
- Make API calls

4.2. Steps to remove memory leaks [8]:

- Avoid using static variables.
- Always unregister events and listeners.
- Use tools such as Eclipse Analyzer or Leak Canary to find these leaks.
- Perform code reviews and code review implementations. Sometimes peer devices can point out things that might not have noticed at all. Usually, this happens when the application have been looking at the same code over and over again.
- Understand the architecture properly before writing any piece of code.

4.3. Corrective methods for rendering:

4.3.1. Hierarchy Viewer:

Hierarchy viewer is a tool which allows developer to inspect properties and layout speed for each view in application layout hierarchy. It measures structure of application view hierarchy and simplifying hierarchy to reduce over draw.

4.3.2. Profile GPU Rendering:

Profile GPU rendering presents visual representation of time taken by UI window to render frames relative to 16-ms-per-frame benchmark. Rendering can be improved by enable profile GPU rendering.

4.3.3. Visual Output of GPU Profiler:

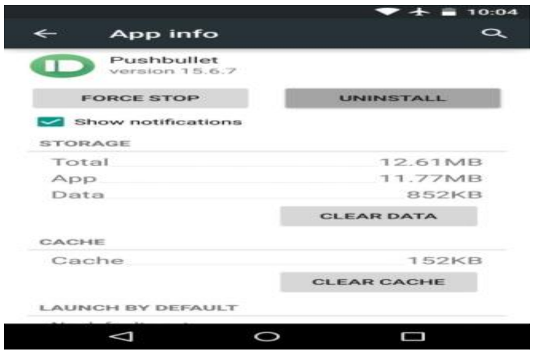
Visual output possess graphical representation consists of horizontal axis and vertical axis for elapsing time and frame time respectively in milliseconds. Interacting application indicated by growth of vertical bars on screen. Each vertical bar determines frame of rendering. Green line indicates target at 16 ms. Whenever all frames crosses green line, missing frame displays stuttering images on user output.

4.4. Resolving application launching time delay:

Application launching delay was resolved by initialising objects immediately needed rather than creating global static objects. If objects are moved to singleton pattern, application can perform lazily by different threads. It causes delay in application launching. Instead of moving objects to singleton pattern, initialise objects for accessing first time. It can be achieved by view hierarchy. In view hierarchy, nested layouts are used for flattening application. It is done by reducing redundant. It updates visual properties depend on bitmaps and other resources.

4.5. Battery usage reduction [9]:

In android application development, important parameter that ultimately leads to retain user is battery usage. Figure 7 shows battery usage reduction by uninstalling unused applications. Optimizations of battery draining issue done by reducing uninstall application many times.


App info

Pushbullet
version 15.6.7

FORCE STOP | UNINSTALL

Show notifications

STORAGE

Total 12.61MB
App 11.77MB
Data 852KB

CLEAR DATA

CACHE

Cache 152KB

CLEAR CACHE

LAUNCH BY DEFAULT

Fig.7. Battery usage reduction

4.6. Improving battery usage [9]:

- Reducing network calls of user.
- Avoiding wake lock.
- Using GPS carefully.
- Using alarm manager carefully.
- Perform batch scheduling.

4.7. Avoid issues with deprecations:

- Understanding and using of proper APIs.
- Refraction of application dependencies.
- Avoid using accessing private methods with reflection.
- Update application dependencies and tools periodically.

4.8. Avoid Abuse:

- Avoid calling private APIs by reflection.
- Avoid calling private native methods from NDK or C level.
- Avoid using adb shell am to communicate with other processes.
- Avoid Runtime.exec to communicate with processes.

4.9. Prefer static methods over virtual methods:

Invocations will be about 15%-20% faster if static methods are accessed than virtual methods. It is a good practice because developer can easily noticed that application method signature which is calling another method cannot alter object's state.

4.9.1. Using static final for constants:

If constants are declared as static final, then the class no longer requires initialization method. Constants in static field initializes from dex file. Constants declared as integer will use integer value directly. Constants accessed as tostring is relatively inexpensive because "string constant" declaration is used instead of field lookup.

4.9.2. Using enhanced for-each loop instead of for loop:

For-each loop is also known as enhanced-for loop. It is used for iterate through count that implements iterable interface. It is used for arrays. Iterator was allocated to interface calls to next (). For iterator, hand-written counted loop was 3 times faster than list array. But for other use cases, enhanced-for loop is exactly same as explicit iterator. So enhanced-for loop is used as default loop but for performance-critical array list iteration, hand-written counted loop is to be considered.

4.9.3. Use Profilers to profile performance:

Profiling is one of the most important steps that you should perform on any application that has any kind of performance issues. There are multiple profiler's available by Android that can be used for application.

4.9.4. Avoid using float:

Thumb rule states that floating-point is 2 times slower than integer. For advanced device operations, float and double are same for speed but in terms of space double is 2 times larger than float. For desktop devices space is not an issue, so, developers should prefer double to float [9].

4.10. Corrective methods for layout performance improvements:

Android sdk uses certain methods to identify problems in layout performance. Following corrective methods are used to implement smooth scrolling UI's with minimal memory trace [10].

4.10.1. Re-using layout:

Reusing layout allows applications to create reusable complex layout. Common elements of application across multiple layouts can be extracted and managed separately. After that those are included in each layout. Custom view is used for creating individual UI components by re using layout file and hence it is easy to progress.

4.10.2. Loading views on demand:

Application layout requires rarely used complex view. Complex views include details of item used, progress indicator for itm and undo process of item. As per their need only, views can be loaded in an application to reduce memory usage and to speed up rendering.

4.10.3. Optimization of Toolbars:

Prefer Toolbar over Action Bar, and prefer Recycler View over List View, especially for animations, and if the application has a bunch of huge images, because it's optimized for that. It has simpler APIs, light-weight, transparent compression, better response caching, and other amazing features [10].

4.10.4. Re-use layout with <include/> and <merge/>:

Android have specific controls to provide efficient and re-usable interactive elements. Developers require a special layout to re-use larger components like custom controls. For efficiently reuse layouts, "include" and "merge" tags are placed to show items for one layout inside another layout [11]. Reusable layout allows creating a reusable complicated layout. Custom view is used to reuse layout.

4.10.5. Delayed view loading:

With certain conditions complex views are rarely used in layout. Loading view reduces memory usage and speed up layout rendering. Delayed loading view of resources is used whenever complex views are needed [10]. It is implemented by using view stub for complex and rarely-used views.

4.10.6. Optimized hierarchy:

Using basic layout structures is not efficient way of consistent and high-performance UI controls for layouts. But it is common misconception amongst community. Nesting layout structures possess high performance than basic layout structures. Nesting several instances of linear layout uses layout_weight parameter which is time-consuming. Each child needs to be measured twice. In list view or grid view, layout is inflated as in view repeatedly. Hierarchy viewer and lint are used to optimize application layout performance.

Hierarchy viewer allows layout to be examined while application is running. It helps to find out bottlenecks in layout performance [12].

- Use compound drawable: Linear layout contains both image view and text view which are handled as compound drawable resource.
- Merge root frame: Replacing root of layout with merge tag for frame layout. Then it does not provide background or padding etc.
- Useless leaf: Layout with no children or no background is treated as useless leaf. Removing useless leaf provides flatter and efficient layout hierarchy.
- Useless parent: Layout with only children but no siblings are called as parent layout which does not comes under scroll view or root layout. As there is no background, removing parent leaf causes children leafs moved directly into parent for flatter and efficient layout hierarchy.
- Deep layouts: Maximum depth for nesting is 10 for achieving good performance. Layouts with more nesting causes bad performance. To enhance performance, instead of nesting, flatter layout like relative layout or grid layout should be used.

4.10.7. Working with Lint:

Lint tool provides possible optimizations for view hierarchy. Lint replaces layout opt tool and hence achieves greater functionality.

Lint automatically fixes issues, providing suggestions for others and offending code for review. Lint is integrated into Android Studio and it will automatically run. In android studio, lint has inspection run choices like specific build variant or all build variants [13].

4.11. Know and use libraries:

Developers always have the knowledge on APIs and libraries of application. Always prefer to use a library’s code over writing application. System has privilege to replace function calls with library methods using hand-coded assembler.

4.12. Using native methods carefully:

Application developed in native interface with android NDK API is efficient than programming in Java. Moreover, there is loss associated with Java-native transition. If resources like memory on native heap, file descriptors are allotted, it is more complex to arrange resources [4].

Native code is more useful when native codebase ported to android application. But it is not used for speed up android application written in Java. JNI tips should be followed for native code in android.

5. RESULTS AND DISCUSSION

Proposed paper analyzes various parameters that are measured and evaluated for android application. Proposed paper discussed that response time and memory usage are to be measured by four different actions, namely start of application, pause application, resume application and navigating from one page to another page on application. Response time analyzes user experience.

CPU usage provides an interesting benchmark to optimize performance. Disk space is preferred for low end devices with limited resources. Compactness is relevant to users. Battery consumption is essential to reach the application to more users. Launch time was measured in phases of launching application and finishing launching. It is recommend from above discussion that limited amount of app-specific code should be added to lifecycle methods. Proposed paper discusses corrective methods for each parameter to optimize performance of android application.

Sample application developed and run on device Samsung SM A505F. Figure 8 shows experimental results for connected device on profiler. Go to profiler will displays screen with parameters CPU usage, Memory usage at specific time intervals.

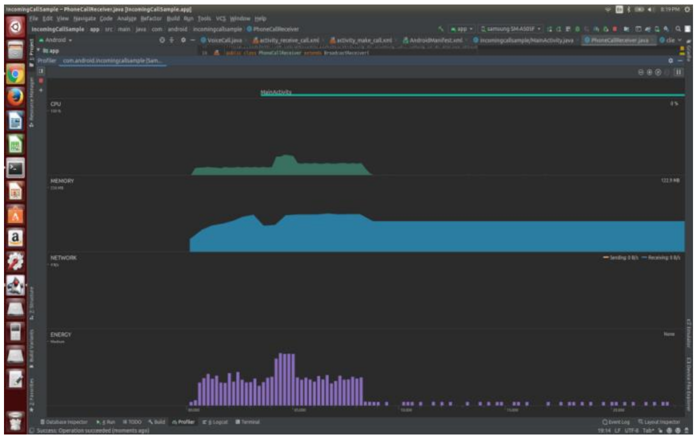


Fig.8. Performance of mobile application on profiler

6. CONCLUSION

Proposed paper presents performance analysis of android mobile application by various parameters. The parameters effected during development time and run times are summarized. After that corrective methods used to improve performance or optimizing performance are discussed. Finally it is concludes that performance of mobile application depends on architectural requirements and layout hierarcehy. So application developers need to consider essential hardware and software limitations to achieve qualitative, effective and efficient application.

ACKNOWLEDGEMENTS

I would like to thank Department of Science and Technology (DST), India under WOS-A Scheme for financial support. I would like to express my very great appreciation to P. Bhaskara Reddy, my project mentor for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

Conflicts of Interest: The author(s) declare that there are no conflicts of interest regarding the publication of this paper.

REFERENCES

[1] V.S. Sundara Rajan, A. Malini, K. Sundarakantham, Performance evaluation of online mobile application using Test My App, in: 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, IEEE, Ramanathapuram, India, 2014: pp. 1148–1152.

[2] M. Willocx, J. Vossaert, V. Naessens, A Quantitative Assessment of Performance in Mobile App Development Tools, in: 2015 IEEE International Conference on Mobile Services, IEEE, New York City, NY, USA, 2015: pp. 454–461.

[3] D.S. Kolluru, P.B. Reddy, Review on Communication Technologies in Telecommunications from Conventional Telephones to Smart Phones, in: International Conference on Advances in Signal Processing, VLSI, Communications and Embedded Systems (ICSVCE-2021), Hyderabad, India, April 2021.

[4] W. Gao, L.-Y. Duan, J. Sun, et al. Mobile media communication, processing, and analysis: A review of recent advances, in: 2013 IEEE International Symposium on Circuits and Systems (ISCAS2013), IEEE, Beijing, 2013: pp. 869–872.

[5] L. Corral, A. Sillitti, G. Succi, Evolution of mobile software development from platform-Specific to web-Based multiplatform paradigm. In: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software, 2011: pp. 181–183

[6] K. Suankaewmanee, D.T. Hoang, D. Niyato, S. Sawadsitang, P. Wang, Z. Han, Performance Analysis and Application of Mobile Blockchain, in: 2018 International Conference on Computing, Networking and Communications (ICNC), IEEE, Maui, HI, 2018: pp. 642–646.

[7] L. Delia, N. Galdamez, L. Corbalan, P. Pesado, P. Thomas, Approaches to mobile application development: Comparative performance analysis, in: 2017 Computing Conference, IEEE, London, 2017: pp. 652–659.

[8] Durga Sowjanya Kolluru, P.Bhaskara Reddy, IP to IP Calling through Socket Programming, in: IEEE Sponsored Asian Conference on Innovation in Technology (ASIANCON)- 2021, Pune, Maharashtra, India, 2021.

[9] D. Kayande, U. Shrawankar, Performance analysis for improved RAM utilization for Android applications, in: 2012 CSI Sixth International Conference on Software Engineering (CONSEG), IEEE, Indore, Madhay Pradesh, India, 2012: pp. 1–6.

[10] S.V.S. Prasad, T.S. Savithri, I.V.M. Krishna, A new technique for color based image segmentation using support vector machines, in: 2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom), IEEE, Greater Noida, India, 2014: pp. 189–192.

[11] M. Hort, M. Kechagia, F. Sarro, M. Harman, A Survey of Performance Optimization for Mobile Applications, IEEE Trans. Software Eng. In press, https://doi.org/10.1109/TSE.2021.3071193.

[12] P. Yoosook, P. Apirukvorapinit, Performance monitoring tool for mobile application, in: 2018 5th International Conference on Business and Industrial Research (ICBIR), IEEE, Bangkok, 2018: pp. 177–182.

[13] P.K. Aggarwal, P.S. Grover, L. Ahuja, A Performance Evaluation Model for Mobile Applications, in: 2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU), IEEE, Ghaziabad, India, 2019: pp. 1–3.

[14] D.S. Kolluru, P.B. Reddy, Development of Voice Call Transfer Service between Android Smart Phone and Tablet, Rev. Geint.-Gest. Inov. Tecnol. 11 (2021), 467-480.

[15] S.V.S. Prasad, T. Satya Savithri, I.V. Murali Krishna, Performance Evaluation of SVM Kernels on Multispectral Liss III Data for Object Classification, Int. J. Smart Sensing Intell. Syst. 10 (2017) 829–844.

[16] K.D Sowjanya, Ch. Srinu, Instant Message Transfer between Two Smart Phones Using Wi-Fi, Int. J. Adv. Eng. Manage. Sci. 2 (2016), 1949-1951.

[17] K. Kuan, T. Adegbija, Energy and Performance Analysis of STTRAM Caches for Mobile Applications, in: 2019 IEEE 13th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC), IEEE, Singapore, Singapore, 2019: pp. 257–264.

[18] L. Zhang, B. Tiwana, Z. Qian, et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones, in: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis - CODES/ISSS ’10, ACM Press, Scottsdale, Arizona, USA, 2010: p. 105.

[19] L. Pu, Performance Analysis of Short Message Service, in: 2009 International Symposium on Intelligent Ubiquitous Computing and Education, IEEE, Chengdu, China, 2009: pp. 370–373.

[20] M.E. Joorabchi, A. Mesbah, P. Kruchten, Real Challenges in Mobile App Development, in: 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, Baltimore, Maryland, 2013: pp. 15–24.

[21] S.V.S. Prasad, Double Block Zero Padding Acquisition Algorithm for GPS Software Receiver, J. Autom. Mobile Robot. Intell. Syst. 12 (2019), 58–63.

[22] L. Delia, N. Galdamez, P. Thomas, L. Corbalan, P. Pesado, Multi-platform mobile application development analysis, in: 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS), IEEE, Athens, Greece, 2015: pp. 181–186.

[23] R.R. Nimbalkar, Mobile Application Testing and Challenges, Int. J. Sci. Res. 2 (2013), 56-58.

[24] K.D. Sowjanya, V.G. Devi, Voice call between Android devices using Wireless Sensors, Int. J. Adv. Res. Eng. Technol. 4 (2016), 10-12.

[25] S. Blom, M. Book, V. Gruhn, R. Hrushchak, A. K, Write Once, Run Anywhere A Survey of Mobile Runtime Environments, in: 2008 The 3rd International Conference on Grid and Pervasive Computing - Workshops, IEEE, Kunming, China, 2008: pp. 132–137.