Android Elastic Service Execution and Evaluation

Ramin Heidari
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 30 weeks of full time studies.

Contact Information:
Author:
Ramin Heidari
Ramin.Heidari@gmail.com

External advisor(s):
Dr. Harald Gustafsson-Master Researcher
Song Yuan-Experienced Researcher
Ericsson AB, Lund Sweden

University advisor(s):
Dr. Mikael Svahnberg
School of computing (COM), BTH

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
ABSTRACT

Context. Mobile devices recently have attained huge popularity in people’s life. During recent years, there have been many attempts for proposing several approaches to delegate and execute the computing intensive part of the mobile applications on more powerful remote servers due to shortage of resources on mobile devices. However, there are still research challenges in this area regarding the models as well as principles that govern circumstances of executing a part of mobile application remotely on a server along with effects of execution on the smartphone resources.

Objectives. The aim behind conducting this research is to propose a model for executing the service component of an Android application on the remote server. This study exploits the enhancement of Android operating system functionality to execute services components on a remote powerful machine. It reports the model as well as the enhancements to achieve this purpose. Additionally, an experiment is conducted to realize what factors rule to execute a computation locally on mobile device or offload it to be executed on a remote machine.

Methods. Two research methodologies have been used in preforming this research; Case study and controlled experiment. In the case study we investigates feasibility of functionality enhancement in Android operating system to run service components of Android applications on a remote server. We propose a new model for this purpose and motivate it by several different resources such as journal and conference papers and the Android developer site. A prototype of the model is implemented in order to put into use in the next part of our study. Second, a controlled experiment is conducted on the outcome prototype of the case study to explore the principles that governs executing the service component of Android application on a remote powerful machines and the affection of this execution on the mobile resources.

Results. A Model for executing the service component of Android application on a powerful remote server is proposed. Also, a prototype implemented according to the Model. The effects of executing Android service components in a remote machine on energy consumption as well as performance of a smartphone are investigated. Moreover, we examined when would be beneficial to offload an intensive computation in order to be executed on the remote server.

Conclusions. We conclude that it’s applicable to enhance the Android OS to execute service component of an Android application on a remote server. Also, We conclude that there is a strong coloration between amount of payload and computation of data that require to be executed on a remote server. Basically, offloading the computation is beneficial when there is a large amount of computation with small amount of communication and payload. Furthermore we conclude that the execution time for the intensive computations drastically increase when it’s executed on the server but for less computation data the performance is better when the execution is on the smartphone. Besides that, we express that the energy consumption on the smartphone growth gradually when the payload passes over a particular size.

Keywords: Elastic service execution, evaluation, Execution time, Energy consumption, Android smartphone
ACKNOWLEDGEMENT

This research has been conducted in Software Research Lab at Ericsson AB in Lund-Sweden in corporation with ST-Ericsson.

Foremost, I would like to express my gratitude to my university supervisor at Blekinge Institute of Technology Dr. Mikael Svahnberg for prompt feedback, constructive comments and critical remarks in this study. Also, I would like to express deepest appreciation to my industrial supervisors Dr. Harald Gustafsson and Song Yuan for their patience, ideas and continuous guidance.

Besides my advisors, I would like to thank my thesis work partner Mohammed Alshammari from Lund Institute of Technology in this research for being corporative and collaborative.

Last but not least, I would like to thank my family and friends for their support throughout my life.
LIST OF FIGURES

Figure 1 Clone Cloud mechanism for migrating an Android application to the Cloud
(Adopted from [9]) ................................................. ................................. 6
Figure 2 Cloudlet mechanism (Adopted from [14]) ........................................... 7
Figure 3 Alfredo architecture (Adopted from [20]) ........................................... 8
Figure 4 High-level of MAUI architecture (Adopted from [24]) ........................... 9
Figure 5 Architecture for elastic application (Adopted from [25]) ......................... 10
Figure 6 Research Methodology ................................................................... 16
Figure 7 Proposed architecture .................................................................. 21
Figure 8 Regular Android development process ............................................ 22
Figure 9 Our modified Android development process ..................................... 22
Figure 10 invoking a method from a remote service ........................................ 23
Figure 11 Start/Stop, bind/unbind a remote service ......................................... 24
Figure 12 STE-8500, ST-Ericsson Android phone (Adopted from [43]) .......... 29
Figure 13 Power supplier ............................................................................. 32
Figure 14 Remote service execution environment ......................................... 34
Figure 15 Local service execution environment ......................................... 34
Figure 16 Collecting results from devices for remote service execution .......... 36
Figure 17 Collecting results from devices for local service execution ............. 36
Figure 18 Descriptive chart for Execution time ............................................... 39
Figure 19 Descriptive data for Local VS. Remote service energy consumption for complexity 0,1,2,3 ......................................................... 42
Figure 20 Descriptive data local Vs. remote service execution energy consumption for complexity 4,5,6 ......................................................... 43
Figure 21 Ratio of total execution time of the service remotely vs. locally .......... 46
Figure 22 Ratio of total energy consumption of the service remotely vs. locally 49
Figure 23 Energy consumption for each payload ........................................... 50
Figure 24 use case of real world applications .............................................. 51
Figure 25 Android System architecture (Adopted from [55]) .......................... 60
Figure 26 hierarchical Class diagram of Android components (Adopted from [37]) ...... 62
Figure 27 Starting an activity ...................................................................... 64
Figure 28 Activity and service Interaction ..................................................... 64
Figure 29 content provider interactions ....................................................... 64
Figure 30 Interaction with Broadcast receiver ............................................. 64
Figure 31 Binder communication ................................................................ 66
Figure 32 Data transaction from Process A to B(Adopted from [37]) ............. 66
Figure 33 Transmission data (Adopted from [37]) ....................................... 67
Figure 34 Start/Stop, bind/unbind the service .............................................. 69
Figure 35 Development environment ......................................................... 71
Figure 36 IPC Binder System ...................................................................... 72
Figure 37 Java API wrapper ..................................................................... 73
Figure 38 meta-date sample for remote service definition ............................ 78
Figure 39 Modified Android framework ..................................................... 79
LIST OF TABLES

Table 1 Terms and Abbreviation Glossary .................................................................3
Table 2 Terms and definitions Glossary .................................................................4
Table 3 Comparison offloading computing approaches ............................................13
Table 4 STE 8500 features and Technologies (Adopted from [41])..........................29
Table 5 Payload size and size of byte data............................................................30
Table 6 Complexity and sorting method repetition .................................................31
Table 7 Measuring points.......................................................................................31
Table 8 measurement criteria.................................................................................33
Table 9 Remote execution CSV columns...............................................................37
Table 10 Local execution CSV columns.................................................................37
Table 11 Power trace CSV column name...............................................................38
Table 12 Development software and hardware......................................................70
LIST OF EQUATIONS

Equation 1 Execution time .............................................................................................................38
Equation 2 Total energy consumption in each functionality ......................................................40
Equation 3 Ratio formula for local vs. remote service execution time .....................................45
Equation 4 Energy consumption Ratio .........................................................................................48
LIST OF CONTENTS

ABSTRACT ........................................................................................................................................ III
ACKNOWLEDGEMENT ................................................................................................................ IV
LIST OF FIGURES .......................................................................................................................... V
LIST OF TABLES ............................................................................................................................ VI
1 INTRODUCTION ........................................................................................................................ 1
  1.1 AIM AND OBJECTIVE ............................................................................................................. 1
  1.2 RESEARCH QUESTIONS ........................................................................................................ 2
  1.3 EXPECTED OUTCOME .......................................................................................................... 2
  1.4 THESIS OUTLINE .................................................................................................................. 2
  1.5 GLOSSARY ........................................................................................................................... 3
2 BACKGROUND AND RELATED WORK .................................................................................. 5
  2.1 AUGMENTED EXECUTION ................................................................................................. 5
  2.2 ELASTIC PARTITIONING/MODULARIZING ........................................................................ 7
  2.3 SUMMARY .......................................................................................................................... 10
3 RESEARCH METHODOLOGY .................................................................................................. 14
  3.1 RESEARCH DESIGN .............................................................................................................. 14
     3.1.1 Case study ....................................................................................................................... 14
     3.1.2 Experiment ...................................................................................................................... 15
4 ARCHITECTURE ........................................................................................................................ 17
  4.1 SYSTEM ARCHITECTURE ..................................................................................................... 17
     4.1.1 Design requirements ....................................................................................................... 17
     4.1.2 Assumptions .................................................................................................................... 18
     4.1.3 Solution design ................................................................................................................. 19
     4.1.4 Constrains and limitations .............................................................................................. 24
  4.2 VALIDITY THREATS ............................................................................................................. 25
     4.2.1 Internal validity ................................................................................................................ 25
     4.2.2 External validity .............................................................................................................. 25
5 INVESTIGATING THE EXECUTION TIME AND ENERGY CONSUMPTION ....................... 26
  5.1 DEFINITION .......................................................................................................................... 26
  5.2 EXPERIMENT PLANNING .................................................................................................... 26
     5.2.1 Context selection .............................................................................................................. 26
     5.2.2 Independent variable selection ....................................................................................... 27
     5.2.3 Dependent variable selection ......................................................................................... 27
     5.2.4 Experiment user stories ................................................................................................ 27
     5.2.5 Hypothesis formulation ................................................................................................ 27
     5.2.6 Selection of subjects ....................................................................................................... 28
     5.2.7 Experiment Design ........................................................................................................ 28
     5.2.8 Standard design type ..................................................................................................... 28
     5.2.9 Instrumentation ............................................................................................................... 28
  5.3 EXPERIMENT OPERATIONS ............................................................................................... 33
     5.3.1 Preparation ..................................................................................................................... 33
     5.3.2 Execution ......................................................................................................................... 34
     5.3.3 Data validation ............................................................................................................... 35
  5.4 RESULT AND ANALYSIS .................................................................................................... 35
     5.4.1 Collecting quantitative data experiment ...................................................................... 35
  5.5 DATA SET REDUCTION ....................................................................................................... 43
  5.6 HYPOTHESIS TESTING ....................................................................................................... 43
     5.6.1 Execution time ................................................................................................................. 44
1 **INTRODUCTION**

Mobile devices have recently gained significant popularity in heavy mobile processing, but still mobile Applications are constrained by limited resources on the mobile such as low CPU frequency, small memory and a battery-powered computing environment. Compared to today’s PC and sever platforms, users expect to run compute-intensive applications such as complex media processing, large scale data management and mining on mobile devices. Therefore, execution of resource intensive applications remains a challenging task in mobile computing that justifying delegation the code execution of the whole or part of the application to a remote powerful machine as an inevitable fact[1].

Sending computation to a more powerful remote machine is not a new notion. During recent years, there have been many attempts to make mobile devices use remote execution to improve the performance and energy consumption. In order to execute some parts of a mobile application remotely on a server, program partitioning is required. This raises some challenges and constrains that require to be addressed. There are some approaches have been proposed to address the challenges and almost all of these approaches focus on finding effective conditions to gain most benefit from offloading computation to a server in terms of performance and saving energy [2].However, the actual realization of these benefits is far from being achieved and there are still some research questions require to be answered regarding effective approach to execute resource intensive component of an Android application in a remote server without source code modification and also evaluation of the solution prototype format from performance and execution time perspective.

1.1 **Aim and objective**

The aim behind conducting this research is to propose a mobile/server model in order not to perform heavily local transaction on Android smartphones and also evaluate the model from energy consumption as well as performance perspective. Following aims and objectives are targeted for this study.

- Propose a model in order to empower Android mobile devices for performing resource intensive mobile application’s components in a collaborative environment, where powerful computing remote server performs code execution on behalf of the weak mobile device.

- Implement the proposed model as a prototype.

- Investigate on energy consumption and performance of a mobile device when resource intensive components of an Android application executes locally as well as execute remotely with respect to the prototype.

Investigate on factors that rule to execute a computation locally on mobile device or offload it to be executed on a remote machine with respect to the model and the prototype.
1.2 Research questions

In order to conduct our research, the following research questions are formulated and will be answered in this thesis:

RQ1: What modifications and extensions to the Android OS framework are necessary to enable applications to partially run on a remote server?

RQ2: What is the effect on resource consumption, energy consumption and performance on mobile phone when part of the application is running on remote server in comparison with running locally?

RQ3: What principles govern whether parts of an application should run locally or on a remote server?

1.3 Expected outcome

Here the following outcomes for this thesis work are expected:

• A Model. In this Model, user interface components of a regular Android mobile application executes on the mobile device and processing components run on a remote server. The communication goes through 3G networks.

• Prototype implementation of the proposed model.

• Result of Evaluation the model in terms of execution time, energy consumption with These results are the outcome of running the application locally on a smart phone versus remotely on a server.

• Identify factors that rules whether executing intensive computing components of an Android application run remotely or locally on a smart phone.

1.4 Thesis outline

The Thesis outline described as following:

Chapter 1: Introduction

This chapter expresses the intention of this study in terms of objectives as well as expected outcomes and the research questions that are being answered in the rest of this thesis.

Chapter 2: Background and related work

This chapter establishes the basis of current study and present adequate background from earlier studies and work in the domain area in order to remark the gaps.

Chapter 3: Research Methodology

This chapter describes methodologies; the specific guidelines including the phases, tasks, method, techniques and tools that will be adopted to answer research questions
of the study. It will also define the steps of performing the methodologies in this study as well as the motivations behind adopting the methodologies.

Chapter 4: Architecture

This chapter describes our prototype architecture including design requirement, assumptions as well as the enhancement on that require to be performed. Additionally the architecture is motivated by studies and researches have been conducted for this purpose.

Chapter 5: Investigation on the execution time and energy consumption

This chapter expresses the controlled experiment that is conducted to support the study. It describes the experiment details including variables, objects, experiment planning, procedure of data extraction as well and presenting the quantitative results along with analysis.

Chapter 6: Answers to research questions

This chapter provides concise answers to the raised research questions of this study along with referencing to the sections that describe the details and provide ample justifications for the work.

Chapter 7: References

This chapter includes list of the cited articles, journal, online resources, studies etc. that were examined in conducting this research.

Chapter 8,9: Appendix

These chapters contain additional information that further clarifies the content of this study. The appendix includes the prototype implantation details as well as details about android framework.

1.5 Glossary

<table>
<thead>
<tr>
<th>Terms</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIDL</td>
<td>Android Interface Definition Language</td>
</tr>
<tr>
<td>APK</td>
<td>Application package file</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-process communication</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>SDK</td>
<td>Software development kit</td>
</tr>
</tbody>
</table>

Table 1 Terms and Abbreviation Glossary
<table>
<thead>
<tr>
<th>Terms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile data Offloading</td>
<td>Deliver some or whole part of a smart mobile application to a remote powerful server or cloud over cellular networks in order to enhance the performance of smartphone applications and reducing the energy usage.</td>
</tr>
<tr>
<td>Payload</td>
<td>Part of a transmitted data, that is the fundamental purpose of transmission.</td>
</tr>
</tbody>
</table>

Table 2 Terms and definitions Glossary
BACKGROUND AND RELATED WORK

There are some motivations behind offloading computing to a remote more powerful machine for execution. First, mobile applications are becoming pervasive and mobile users want to run computation intensive application e.g. game, image processing on mobile phones that it can’t afford or execution performance is low due to constraints in storage capacity, CPU, memory, etc. [3]. Second, produce longer lifetime battery is a major obstacle and bottleneck for future growth of mobile devices. Battery lifetime limitation on smartphones has always been considered as main constrain in mobile industry Longer battery life is not only important for internal mobile functionalities such as screen, camera etc. but also is critical to execute the power intensive applications e.g. voice recognition, image retrieval [2]. Various approaches have been suggested in [4][5][6][7] to enhance the CPU performance and operate screen to save power on smartphones. However, they demand modification on mobile phones hardware that not only cost money but also are not feasible for all mobile device providers. Offloading the intensive computing process to a powerful server or the cloud in mobile devices, enable mobile applications to reduce execution time and energy consumption [8]. Third, nowadays, it’s applicable for mobile users to use their smart phones for checking bank accounts with special banking applications or storing games scores etc. Hence, the information on their mobile phones is vital for them that need to be backed up. In particular offloading approaches a clone version of the phones VM is offloaded to a remote sever or the cloud that is replicated with the VM on the phone. The cloned VM version can be retrieved on the phone again from the remote server or cloud in case of data corruption on the phone or phone missing etc. [9]. Therefore storing backup of data on a remote server or cloud reduces the chances of data and application loss on a mobile device that lead to improve reliability [10]. Fourth, server or network resources can be shared to provide service for a large number of users which result in reduced costs for support and hardware service [10].

There have been a few attempts done to make mobile devices use remote execution machines [11]. Most of these approaches are classified into two main categories; augmented execution and elastic partition. Below, we briefly summarize previous studies and approaches regarding these two categories along with capabilities, constrains, advantages and drawbacks of each.

2.1 Augmented Execution

Augmented execution is a technique to increase the performance of hardware-limited smartphones by running some parts of mobile application’s processes on the cloud where there application access to limitless memory, power and processors[12]. This Approach is relying on VM migration. In this approach the mobile application or in some cases the entire OS including VM will be migrated to the remote server or cloud[13][11][14]. There are some architecture solutions to address these challenges that I mentioned in following.

Chun and Maniatis [11] proposed a model to overcome the challenges. Based on their model, the execution part of application offloads to run on the cloud [12]. They introduce a framework to offload application to a virtual environment in the cloud automatically and securely. In their approach the developer doesn’t require to redesign their applications. For this purpose, the user installs an agent program to automatic the whole procedure. This agent application subscribes to a virtual machine on the cloud
and setup a virtual environment to emulate the Android phone. Basically it uses a standard image already stored on the cloud to create an environment and make a clone of the operating environment and the application of the phone by copying them on to the virtual environment. The agent program is responsible for synchronizing data between the smartphone and the mirror data on the phone. This mirroring generates heavy traffic on the network between smartphone and the cloud. The agent also is used to migrate data to the cloud and visa versa. In the specific time scales, the agent sends signal to force the application call “OnPause” which lead the application to save its state in to a state file. Subsequently, agent sends the state file to the agent on the cloud. The agent on the cloud receive the state file, save and call “OnResume” function then continue executing the application on the cloud after restoring the state. This group of work is done exactly the same on the phone when state data has been delivered from the cloud to the phone. The merit of the architecture is that the mobile user believes that have more powerful, feature-rich device in hand without requires to modify and partition the application manually. The drawback is that in this solution, in the first step offloading whole data and the mobile application to virtual environment take time and energy from the device .The time and the energy consumption depends on the network connection (3G, 4G, wifi) from one side and network traffic from another side [11][12][9]. Figure 1. Depicts the Clone Cloud mechanism in an Android application as well as the cloud. This model evaluated by their three developed applications. Virus scanner, image search were implemented to evaluate the prototype from execution time and energy consumption perspective on a smartphone with WiFi and 3G connectivity for local execution on the phone and remote execution in the cloud. The results show that the prototype delivers up to 20x speedup and 20x energy reduction for the tested applications[15] .

Satyanarayanan et al.[14] Suggested similar approach .In this model a mobile user utilize cloudlet; a nearby resource rich computer or cluster of computers that are well connected to the internet and available in a WLAN to obtain the advantage of cloud computing. This access to the WLAN results in remarkably improve in response time, interactivity, low latency between mobile device and cloudlet. This solution relies on VM migration and dynamic VM synthesis approach .As shown in figure 2. the mobile device acts as a thin client with all facilities to connect to a cloudlet. In VM migration approach first already executing VM is suspended with all processor, disk, memory states then transferred finally to the cloudlet to resume subsequently in destination from the same suspension state. In the other approach; dynamic VM synthesis, A

![Figure 1 Clone Cloud mechanism for migrating an Android application to the Cloud (Adopted from [9])](image)
Mobile device sends a small VMs overlay to cloudlet infrastructure that already possess the base VM from which this overlay was derived. The infrastructure engages the overlay to the base to develop launch VM. This causes start executing the VM in state that it was suspended. They have built a proof-of-concept called kimberly to demonstrate the feasibility of dynamic VM synthesis. Their experiment contains indication of the Synthesizing that includes overlay transmission, decompressing and applying the overlay on the cloudlet VM. Execution of these steps take 60-90 seconds, which require a significant improve for the real-world deployment[12][14].

![Cloudlet mechanism](image)

Figure 2 Cloudlet mechanism (Adopted from [14])

According to the evaluation conducted in [16], the execution time of application changes with variant number of VMs. The result indicates that the average execution time for each cloudlet increasing hand in hand with number of cloudlets and VMs. The average execution time of the cloudlet increases at least 28% for 2 to 5 cloudlets and maximum 57% for up to 45 cloudlets.

2.2 Elastic partitioning/Modularizing

This approach is based on partitioning the application. In this approach a programmer or an offload application decides which part of the program require to be executed remotely based on network conditions. Therefore, the sub-parts of the application that takes benefit from remote execution are performed on the powerful remote server[17][18][19].

Giurgiu et al. [20] developed an application middleware platform to partition and distribute different parts of mobile applications between mobile device and the remote server with considering low-latency, less data transfer etc. The core of this platform uses AlferedO[21] to automatically and dynamically decide about the distribution time and modules that requires to be offloaded to the server in order to achieve the optimal performance. AlferedO[21] provides a facility for the programmers to decompose and distribute the logic and presentation layer of the application with keeping data layer of the server side. AlferedO is developed based on an extension version of OSGi.
model namely R-OSGi[23]. R-OSGi as the extended version of OSGi allows running services on different virtual machines unlike the origin OSGi. Also It is used to decompose and loosely couple java applications into software modules that are called bundles[12].Figure 3. Illustrates a sample of client and server interaction. Both phone and server run OSGi with the installed R-OSGi , which provides remote service execution over OSGi platforms. Alfredo consists of two bundles; AlfredOClient and Render on the client, and AlfredOCore on the server-side[20].

When the connection is established client requests the selected application. Subsequently, AlfredOCore uses one of partitioning algorithms to determine optimal deployment for the application then returns to AlfredOClient, the application descriptor along with the list of services, which require to be fetched. Renderer generates java AWT or SWT user interface according to application descriptor. These actions are in parallel with fetching specified services that are decided to run on client (S1) by AlfredOClient via R-OSGi. A local proxy is created on the client as an interface for the services that are decided to run on the server[20].

Experiments on the solution revealed limitation and applicability of using AlfredO. In most of applications there is tightly coupled between user interface and service logic. This means that modularization in service logic lead to have modification on user interface and visa versa. But also the experiments have shown even the more complex applications require to be modularized to run on AlfredO, still the effort is reasonable. Moreover, Profiling resource consumption of application’s bundles and their inter-communication require to be considered in algorithms to improve optimization decision. Finally, using OSGi in AlfredO that is maintained by famous software companies such as IBM, Oracle, SAP and device vendors such as Ericsson, Motorola. This is considered as a merit for this solution[20][12].

They built from scratch a prototype application to test their solution under various resources and network constrains for the local execution on the phone and remote execution on the server. The prototype application is a very interactive application exhibiting a good mixture of light and heavy processing components. User of the application uploads the image of their house and furniture items as well as their positions, properties such as color rotation and dimension to the server. Subsequently, user can invoke specialized image processing libraries for image composition. Results indicate that execute the application in remote server with 14 times is faster than execute it locally on the phone[20].

In similar case MAUI[24] is developed to exploit the benefit of code offloading to cloud infrastructure in order to maximize battery life in mobile devices. MAUI provides a development environment, for the developer to annotate the methods, which are supposed to be executed remotely. MAUI gathers profiling information of the
offloaded methods to have a better determination whether code of future execution should be offloaded and executed in the cloud or managed to be executed locally on mobile. It determines optimized condition by analyzing profile information considering specific factors regarding cost of offloading, measure network connection, estimating bandwidth and latency. This information is used to distinguish between the methods, which suits for local and remote execution. Unlike [20], MAUI allows offloading mechanism on the level of single methods while the offloading in [20] occurs on the module level. By using MAUI [24] mobile users are able to offload mobile code to the cloud infrastructure in order to maximize battery life of the mobile. Developers indicate the methods, which should be offloaded for remote execution by annotation during programming phase. In comparison with [20] in the MAUI offloading mechanism is defined on single methods while in [20] the offloading occurs on complete software modules [12]. Figure 4 shows the high-level architecture of MAUI.

![Figure 4 High-level of MAUI architecture (Adopted from [24])](image)

To evaluate the prototype, a video game, chess application, and face recognition are implemented. These three resource-intensive applications used to measure execution time and saving energy for running the applications locally versus using MAUI for remote execution on a remote server. Offloading the code has been done over WiFi and 3G. The results indicate that the consumed energy on mobile when offloading code is done over 3G is 2.5 times higher than offloading the code to a nearby server. Saving energy for offloading to nearby server is 27% for the video game and 45% for the chess. MAUI presents a substantial performance improvement for performing face recognition as well as video game. MAUI performance for the face recognition 9.5 times and for the video game 4.8 times is better than local execution. For the chess, performance becomes worse due to the high latency when the connection is over 3G [24].

Zhang et al. [25],[26] proposes a framework for partitioning an application into multiple elastic components. These components are called weblets. Also, the framework handle dynamic adaptation of weblet execution configuration. Weblets can be platform independent or platform dependent and able to be executed transparently.
on different computing infrastructure such as mobile device, cloud Amazon EC2 or S3. The main functional components shown in figure 5. The framework consists of UI component, weblets and xml manifest. The xml manifest describes requirements and constraints of the application. Weblets are autonomous software entities that execute on device or cloud and expose RESTful[27] webservice interface. They take care of performing computing, storing, and networking tasks. On the device, elasticity manager is responsible for elastic migration, instantiation, and migration of weblets. Unlike AlfredO and R-OSGi, developing Weblets are not dependent on one specific programming language. This leads to cover a broader range of applications [12][25].

![Figure 5 Architecture for elastic application (Adopted from [25])]()

The evaluation results of the solution indicate that the execution time and energy consumption for local versus remote execution is heavily dependent on I/O operations. Also, the power consumption of network interfaces to migrate the weblets may override the benefits of the remote execution[25].

### 2.3 Summary

We studied and analyzed the related existing solutions in this domain and summarized their specification as well as their pros and cons in table 3. Basically, a comparison of exiting approaches in this area points out the way to find a better solution in this domain. Information of available application models are summarized according to following attributes:

- Technologies: what kind of underlying technologies is used in the solutions?
- Code modification/Clone: Does the application code need to be modified or cloned?
• Solution generality: Does the solution work for only small group of applications or all different type of applications?
• Implementation complexity: How difficult is to modify an available application or develop new application to use this solution?
• Saving energy: how much energy reduction is expected on mobile device by using solution?
• Framework/Agent: Does this solution work as a framework or Agent?
• Drawback(s): what are the drawbacks of this solution?
• Benefit(s): what are the benefits of this solution?

Except Code modification/Clone, Saving energy, Framework/agent, drawback and benefits attributes that have been provided in the table, the other attributes are adopted from[12]; a previous study done in this domain. Study new attributes in addition to what studied in [3] not only assist us to compare the exiting approaches for mobile cloud but also point out the way to the better solution. By examining the existing solutions we found number of reasons to propose new approach as follows.

First, all of the existing solutions based on augmented or elastic mechanisms approaches require to be installed as an application or framework on the phone and require third party libraries for performing. For instance in[11] an agent handles the cloning code and manage the interaction between the client on the phone and the other part on the cloud. However, This is a potential security risk for the cloud service providers because the remote execution applications can be manipulated and used as a malware application. Malware execution is potentially a security risk with the catastrophic consequence for the service provider[28].

Second, despite the implementation complexity in all of existing solution except [15] is low, still it’s a burden for the developer to modify or develop an application to use an specific library and annotation in their code to execute some parts of the program remotely. For instance in[25] developer requires to use a specific SDK to develop the program or in [24] developer requires to put annotation on those method that are supposed to be executed remotely.

Finally, In augmented mechanism at beginning of process, the whole individual application or full VM as well as vendor specific libraries or system services need to be offloaded to the cloud. This mechanism creates a clone environment on the cloud as compatible as the version in the mobile device and by mirroring of the phone device in the cloud. This functionality consumes phone battery and also increases the network traffic.

This thesis is targeting to address the above limitations of the existing solutions by proposing a new solution. The contribution of the new model will be:

• Proposed model neither require to clone the whole application code same as augmented mechanism nor annotating the code or using special SDK to develop the code. This causes reducing the network traffic as well as consuming energy consumption on the phone. The reason is that it doesn’t require sending the whole OS and individual application to the server.

• It’s a built-in model on the phone not a framework or an application that requires installation by the phone user on the Android phone. This solution will be a part of the Android operating system that can be implemented in Android by mobile device providers such as Sony, Samsung etc. to be used in
their phones. This built-in model not only helps to improve the solution's generality but also provides a secure service for end users.

- In this model, developers neither need to modify the application code nor annotate some parts of the code, for specifying the code, which are supposed to be performed on the server side. Also, developers don’t require using a special SDK to develop their applications that are supposed to be executed in this model.

Furthermore, we implement a prototype of the model to evaluate the solution from execution time and energy consumption perspective. The motivation behind this evaluation is to find out how much execution time and energy consumption will be reduced by executing partial of the application remotely.

During our preliminary reviews of previous studies, it is also noticed that there are some inconsistencies in between the experiment environment as well as outcomes and statistical data, which rise as a challenge in comparing the result of our experiment with previous studies. In order to mitigate the mentioned issues and challenges for the future researches and make our results comparable to the other research results, we will describe the methodology as well as detail of experiment environment, outcomes and results clearly in our thesis.

Specially, there was not sufficient detail regarding the applications involved in experiment as well as experiment environments such as network, devices and related detailed statistical data etc. therefore, we won’t be able to compare the results of our experiments with previous researches.
<table>
<thead>
<tr>
<th>Model</th>
<th>Underlying Technologies</th>
<th>Code modification</th>
<th>Solution generality</th>
<th>Implementation complexity</th>
<th>Saving Energy</th>
<th>Framework/Agent/Application</th>
<th>Drawback(s)</th>
<th>Benefit(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chan and Mantinits (CloneCloud)[11]</td>
<td>Dalvik VM (Android)</td>
<td>Clone code</td>
<td>Low</td>
<td>High</td>
<td>20% energy reduction for testing simple application [13]</td>
<td>Application</td>
<td>-Inability to migrate native state and to export unique native resources remotely.</td>
<td>Does not require the developers to redesign their application users can migrate their application from one mobile to another quickly[9]</td>
</tr>
<tr>
<td>Satyanarayan et al. (Cloudlets)[14]</td>
<td>VirtualBox Dynamic VM synthesis</td>
<td>Clone code</td>
<td>Low</td>
<td>Low</td>
<td>-</td>
<td>Application</td>
<td>-Requires an expensive nonvolatile memory to store data from clouds.</td>
<td>Looks for the closest available cloudlet to achieve low latency and one-hop, high-bandwidth wireless network.</td>
</tr>
<tr>
<td>Giangia et al., [20]</td>
<td>OSGi, Java</td>
<td>Medium</td>
<td>Low</td>
<td>-</td>
<td>-</td>
<td>Application</td>
<td>Depend on programming language[12]</td>
<td></td>
</tr>
<tr>
<td>Cuervo et al. [24] (MAUI)</td>
<td>.NET</td>
<td>Annotation the code (code portability)</td>
<td>High</td>
<td>Low</td>
<td>Saves 27% for the video game and 45% for chess</td>
<td>Application</td>
<td>Programmer should decide which part of the program need to be executed remotely[15]</td>
<td>-Execute the same program on the x86 server without access to program source code can make optimal decisions for partitioning based on the linear programming formulation</td>
</tr>
<tr>
<td>Zhang et al. [25] (Webllets)</td>
<td>REST, C#</td>
<td>Developer uses an SDK to develop the application</td>
<td>High</td>
<td>Low</td>
<td>Heavily depends on the I/O operations it performs. the power consumption of network interface may override the benefits of the migration Augmenting[25]</td>
<td>Application</td>
<td>Develop application use a specific SDK developed for this purpose[25]</td>
<td>Platform independent (Java or .Net bytecode or Python script)</td>
</tr>
</tbody>
</table>


3 **RESEARCH METHODOLOGY**

This chapter presents the research approaches and disciplines as well as research instruments of this study.

3.1 **Research Design**

In this research, based on[29] in order to answer our research questions “mixed research” will be adopted.

3.1.1 **Case study**

Case study will be adopted to answer the **RQ1** in this research. According to [30] Case studies are very suitable for industrial researches in software and science phenomenon since it studies contemporary single entity in its context in a specified time window. The case study is evidence base for professional application. The case study primarily uses for exploratory purpose with the aim of investigating how to the development conducted by the researcher under different condition and trying to find out what is happening [30]. Furthermore, case study allows the researcher to gain tremendous insight into a case also gather data from a variety of sources and to coverage the data to illuminate the case that is quite match with our research. In our case study, we explore the feasibility of enhancing the functionality of Android OS in order to enable it to execute a part of Android application on a remote server and some other parts of the same application locally in Android framework. The next step is to investigating to develop a prototype by modifying the Android OS. Therefore case study research method allows us to provide detail information and elaborate about the proposed model as well as prototype implementation. In order to familiarize the reader to the Android concepts, some Android OS functionalities will be described with helping class diagrams, sequence diagrams etc. that are accessible in the Appendix I. in order to have optimal performance in the prototype execution, operating the optimal implementation will be considered. In the final of the case study, study evidences, figures, statements are linked together to support a strong and relevant conclusion. The outcome of the case study will be prototype of our proposed model that can be used to answer RQ2, RQ3. The case study can be lengthy [31]. Therefor, in order to hold the reader interest, we tried to provide informative information in digestible manner. To reduce the concern of lack rigor in this case study and not be bias in findings, being systematic in data collection and taking steps to ensure validity and reliability has been considered.

The research process of this case study is flexible design process because the key parameter of the study is changed during the course of study. In conclusion of this study case study evidence, figures, statements are linked together to support a strong and relevant conclusion.

All modifications; alternations and enhancement to the Android OS framework are logged in chapter 4 of this study where RQ1 is answered. Also, no further analyses need to be taken into consideration in the scope of this thesis regarding the modifications and enhancement.
3.1.2 Experiment

Experiments are concerned with a limited scope and most of the times are run in a laboratory setting and often formal, highly controlled. In experiment, Controlled test are performed to look for a casual relation between two treatments [32].

In the second part of this research, a controlled experiment will be conducted to answer the RQ2, RQ3. The motivation behind performing the control experiment is that the prototype and proposed model is different than the available solution therefore there is no study or evaluation has been conducted before regarding this solution to present the evaluation of the proposed model’s prototype from different perspective. Moreover, find the factors that motivate the execution of some parts of an application remotely on the server is another reason for conducting the control experiment. Prototype of the proposal model is the expected outcome of the case study that will come into use as the key element to conduct the experiment. Performing such an experiment on the prototype will give us fair idea about the factors as well as the functionality of the proposed model. Furthermore unlike the previous studies in other solutions, the experiment steps in this study as well as instruments will be described in detail and the results clearly presented which allow the other researchers to re-perform it. The details of the experiment are as follows: the length of the experiment is bounded to the number of functionalities. Functionalities are generated during the execution of application experiment based on defined specifications:

**Experiment application/framework:** Used for conducting the experiment that will be an Android application and an enhanced version of Android framework.

**Dependent variables:** Total execution time and energy consumption on the phone

**Independent variables:** execution environment, number of executions, Network, payload and complexity of the data unit.

**Objects:** enhanced Android framework

At first, the environment of experiment will be arranged in order to conduct the experiment. It’s important to assure that everything is in place before beginning the experiment. For this purpose the steps of conducting the experiment will be written down before the starting the experiment. The results of the experiment will be collected automatically from the devices, which are involved in this experiment because the experiment will be performed more than one time. In order to have consistent results of repeating same experiment, the experiment will be performed in working days preferably in the same time. The reason behind that is to have similar network functionality. The Heatmap[33] will be used along with other charts to analyze and present the results because it provides understandable and meaningful representation of the complex experiments’ data. Part of Data synthesis and plotting the heap map are performed by Octave[34]. Overview of the research methodology has been depicted in figure 6.
Figure 6 Research Methodology

RQ1: What modifications and extensions to the android OS framework are necessary to enable applications to partially run on a remote server?

RQ1: Fulfills objective 1,2

RQ2: What is the effect on resource consumption, energy consumption and performance on mobile phone when part of the application is running on remote server in comparison with running locally?

RQ3: What principles govern whether parts of an application should run locally or on a remote server?

RQ2, RQ3: Fulfills objective 3

Answer of RQ1

Answer of RQ2, RQ3
4 ARCHITECTURE

In this chapter, we propose an execution framework as well as its architecture in order to run mobile application service on a remote server. Understanding the architecture and the framework requires adequate knowledge regarding primarily concepts of Android framework as well as anatomy of Android applications. This information is addressed in Appendix I and Appendix II.

4.1 System architecture

In this section, we present our new model in a high-level overview of components on mobile device as well as in the infrastructure in order to understand how they all integrate into one platform for developing mobile applications. The goal of our solution is to maximize the advantages of code offloading for today’s mobile devices.

Imagine a server environment that is capable of running the same application as the mobile device in a user’s hand. The user may use either the mobile device or the remote server to execute part of the application on the server. However, there are several key issues in designing a framework to facilitate the aforementioned scenarios:

How to separate the service component of an Android application from its package to be executed on a remote machine?

How to execute the service part of an Android application on a remote machine without modifying it or with just a minimum change on the application metadata without using an SDK to develop the application?

How to provide the service execution facility mechanism as a built-in mechanism in Android OS and not using third-parties?

How to handle multiple services and multiple applications in the remote service execution?

In order to address the obstacles described earlier, we have enforced the following design requirements onto our framework

4.1.1 Design requirements

4.1.1.1 No manual modification of application source code

The programmer or developer should not be required to use any additional API or third party to develop the application. The Standard Android SDK APIs that are used regularly in developing the application can be offloaded later to the remote server manually using our built-in framework.

4.1.1.2 Built-in functionality in Android OS

Our framework should not be required to be installed, as a regular Android application. The offloading mechanism should be implemented inside the Android OS.
and be used as an Android built-in functionality. We believe this requirement is a key factor for our framework to come into use practically by smartphone manufactures and telecom providers.

4.1.3 Clear partition of application

Our built-in framework should partition an application clearly in sense that the activity components able to be executed in the Android OS on the smartphone and the application services able to be executed on the remote server.

4.1.4 Handling multiple applications

Our built-in framework should be able to handle multiple applications at a same time on the smartphone and the remote server. For instance two or more applications on the phone can use their remote execution functionality simultaneously.

4.1.2 Assumptions

We make two following assumptions on the applications that should be deployed on our enhanced Android OS in order to offload intensive computing to the remote server.

4.1.2.1 Use MVC(Model view control) design pattern

There are four types of application components have been defined in Android OS architecture: content providers, broadcast receivers, activities and services. The last two components have been considered in our built-in framework; Activity provides the user interface for end user and service execute background tasks. In order to use our solution, developer require to use Model-view-control (MVC) pattern in developing their applications which result in isolating application logic and the user interface from each other. Basically, our built-in solution assumes that the compute-intensive part of the application is implemented in service components and user-interaction in activity components. Also, we assume that the activity and service are executed in two separate processes but in the same or different package.

4.1.2.2 Use AIDL

In Android architecture one way for communication between activity and service process is through Inter-process communication (IPC). IPC includes some particular methods that use Binder to interact information between the processes. A special type of massage that is called intent does passing the information between the processes. Intent is not appropriate for this case because the handing multiple service operations as well as handling the service results and returning the result to the activity can be complex and error prone.

Our built-in solution requires using AIDL in order to handle IPC between the activity and the service. AIDL is a built-in mechanism in Android that provides an interface. This interface is used for interaction between the service and activity through IPC. It provides a mechanism to decompose the objects into primitive data type such
as string, integer etc. and serialize the decomposed object to pass through Android OS all the way from the activity to the service and visa versa.

4.1.2.3 Use Specific tag

The configuration of installed application is stored in Manifest file of the Android application. The manifest file composed of some detail information about activities and services. For example for the services, Manifest includes the address of the service file as well as the service types.

In our solution, the developer requires to specify a special tag in order to execute their service remotely. This tag is used at the run time by the components in Android OS to distinguish the remote service from regular services.

4.1.3 Solution design

The development of our solution outlined in figure 7. Essentially our solution comprised of two parts; Client side, server side. Client side is an Android smartphone. Our built-in framework in smartphone is developed as an enhancement in Android OS and works as a built-in solution. This means that the developer doesn’t require to install the framework on the smartphone and the functionality consider as an Android built-in functionality.

4.1.3.1 Client Side

In client side most of all enhancements have been performed in Application Layer and Native layer of Android OS. Client side essentially composed of five components: Binder, ActivityManager, package manager, metadata and SocketClient. PackageManager includes all configuration information about the running application on the Android OS[35]. Android framework loads up all configuration into PackageManager in application framework layer from AndroidManifest.xml at the application runtime, which includes Android service specification such as name and package address and type of the service as well. In order to identify between the remote and local service in Android framework, we decided to consider a specific meta-data tag that includes the definition of remote service in androidmanifest.xml file. Therefore, this data will be accessible at the runtime from PackageManager component inside the Android operating system. Also the Internet permission tag must be added into the AndroidManifest.xml, which let the application, have access to connect to the Internet over the phone in order to communicate with the remote service.

MetaData: This component is fully implemented by us. MetaData is used to keep the service configuration of the application at run time. This data includes type and state of the services at run time such as current state and previous state of a service. The data retrieve from the MetaData is used by other Android components to distinguish different type of service in the application.

ActivityManager: It’s an Android component which interact with the overall activities running in the system [36]. This Component is used by packageManager to retrieve information about a particular service or activity that is currently running in the system. We modified this component to retrieve data of the services that are supposed to be executed on the remote machine.
SocketClient: This component is fully developed by us in Android OS. It’s responsible to establish and maintain a TCP connection between the smartphone and the server machine. Also, socketClient is responsible for marshaling/un-marshaling the data parcel that travels over the network between the processes. The reason behind using the socket is that java socket is implemented with available java standard APIs in Android framework therefore, there was no need to add 3rd party APIs in Android OS. Furthermore, socket provides a facility to have more control over data transmission and buffering the data and also it’s an efficient way for two-way connection in our solution. As the cost and overhead of re-establish socket connection over the network between the client and the server is expensive, in this solution the connection between the client and the server is established at the first invoke and maintain available till it drops by the client or due to the network issues.

Binder: It’s the key point in our solution. In Android, communication between the process is go through the Binder [37]. In AIDL, Activity calls the transact() method in the Binder from the proxy to reach the service through the kernel methods that are running in another process. The implementation of the transact method is in Native Layer of Android OS. The default transact method doesn’t have an ability to distinguish the regular requests from the request coming for executing a method from a remote service. In our solution, we modified the Binder and transact method to make it able to distinguish between the method calls. Our developed transact method check the received parameters and based on the request, executes the original transact method for the regular IPC transaction or call the SocketClient to send the request to the remote server.

The data that are used for distinguishing the different requests are retrieved from the ActivityManager and PackageManager components. The results of the invoked service method on the remote machine are received by the SocketClient component from this step forward; the response behaved as a regular Binder transaction and results passes to the Binder and proxy to be used in the activity.

The solution on the physical smartphone is multithread solution. This means that solution allow developer to run multiple application with multiple services at the same time.
In the rest of this sub-section, we describe how the developer requires changing the application metadata in order to use the remote execution solution. Build process of an Android application, which is developed based on AIDL, has been shown in figure 8. The configuration of resource files such as GUI layout; Activity and service etc. are developed and addressed in AndroidManifest.xml file. The AndroidManifest.xml is generated to R.java in the building process by aapt tool. The aidl tool converts the implemented aidl file that is implemented by developer to a java file that contains Proxy and stub interface. The aidl tool comes with Android SDK and generate java interface class (IService.java). The generated java files along with other java source codes, which are implemented by the developer, are compiled and then are packaged with other files such images into an zip format file with apk extension. This apk file is installable on Android.
Developer requires specifying the service that is supposed to be executed on the remote machine with a particular tag in AndroidManifest xml file. Also, we expect developers add user permission tag in the Manifest file so that it gives a permission to establish a connection with the server. Developers don’t require modifying the source files.

We assume that the mobile application meets the assumptions described in section 4.1.2. Figure 9 illustrates our modified development process.
4.1.3.2 Server side

The Server side solution is responsible for handling the requests coming from activities of the smartphone by executing services. Server side is an Android X86 OS installed in Oracle virtual Box. The server side solution is installed on the Android X86 OS. Essentially, Server side solution composed of one component: Broadcast receiver. Broadcast receiver starts running by starting the Android X86 OS. It includes a socket that always listens to a specific port for establishing connection with smartphone. Broadcast receiver, unmarshals the data when it’s received by the socket. The data contains information of the destination method and the service as well as the method values. The target service is instantiated by using replication mechanism and subsequently the target method in the service is invoked through the Binder. The result of the method execution is marshaled and delivers to the smartphone. The sequence diagram of whole process in invoking a method of a remote service by the Android activity is shown in figure 10.

![Figure 10 invoking a method from a remote service](image)

The same mechanism is used for binding /start/stop a service. Figure 11 outlined the sequence diagram of start/stop/binding a remote service. However, the only difference is that the Intent is used to stop/start/bind a service. Unlike the invoking methods that the request and response go through the proxy and stub interfaces, the Intent goes through the activityManager.
4.1.4 Constrains and limitations

The constrains and limitations regarding the proposed solution are as following:

In our solution, it’s not possible to marshal a parcel that includes the binder object and send it over the network to the server. Binder object uses shared memory mechanism in kernel binder to transmit from one process to another. Shared memory mechanism is adopted because it’s faster and reliable. In our solution, if the parcel to the remote server contains a binder object, the Android framework will throw a runtime exception, once the marshal method in the parcel is called to convert the parcel into the bytes. The exception shows up as “Tried to marshal a Parcel that contained Binder objects”. This exception throws in middleware layer in `android_util_Binder` inside the implementation of marshaling method.

For instance in AIDL a bitmap object is submitted through the kernel binder with using shared memory mechanism from activity to the service. This works for the local services. In our solution; we tried to send the bitmap object from an activity to a remote service on different machine over the network to be processed. The runtime exception was thrown when our modified framework tried to marshal the parcel, which contained the bitmap object. The work around regarding the problem is to convert the bitmap image to the byte array using `Bitmap.compressedFormat()` in advance and then pass it as the parameter to the remote method service. In this case, the bitmap image will be converted to byte code two times; first time by the developer who needs to change the source code of the application to convert the bitmap to bytes and then pass the byte code as one of the remote service parameter in activity; second time by our modified Android framework where the marshal method in parcel with bitmap image is executed. This way is not efficient because converting the bitmap to byte in the Android and again decodes it to the object takes time more than the regular case. The more convenient solution would be changed the Bimap API in Android framework to exclude the binder object for the remote services. As we had time constrain in this thesis we consider it for the future work.

![Figure 11 Start/Stop, bind/unbind a remote service](image-url)
4.2 Validity threats

4.2.1 Internal validity

In this study, during the implementation of the model we tried to modify and enhance the Android operating system with the less impact on the performance of the whole Android OS execution. However, this possibility shouldn’t be ignored that the modification, enhancement and alternation may impact on the performance of Android OS. Although the impact on the performance was not tangible for us during conducting the experiment, it is worth to be considered as a threat in this study because can influence on the result of the experiment.

4.2.2 External validity

First threat is related to the operating system constrain. The prototype of the solution is implemented in Android framework as a built-in functionality and it’s different than that other mobile operating system windows etc. Therefore, it is not possible to generalize the solution into other mobile operating system in this family such as Apple iOS[51] for Apple iPhone or windows mobile OS. Second threat is related to the Android framework version that is used in this study. As mentioned earlier, the solution is implemented on the customized version of Android framework from ST-Ericsson and patched according to Android 4.04, which was the last available version of Android OS at the time of starting this study, and it was never tested on other prior versions. To eradicate this we just changed those files and functionalities, which were in common in all Android versions from 2.5 till 4.04. Therefore we can be sure that the mechanism works on the prior versions in all Android phones from different brands. The type of phone that is used in this experiment can be a threat to the external validity because it’s not look like the regular phones. But from functionality point of view it’s the same as the regular Android phones.
5 INVESTIGATING THE EXECUTION TIME AND ENERGY CONSUMPTION

The motivation behind conducting this experiment is to explore the effects on energy consumption and execution time when Android service application executes remotely versus locally. Also, study principles that lead to execute the Android services on a remote server profitable.

5.1 Definition

Based on [40] The subject is the person who is taking part in an experiment or study in order to evaluate an object. As there is no person as the subject to get involved in this experiment the subject is not applicable here. The dependent variables are total execution time and phone energy consumption and the independent variables for the experiment are the execution environment, number of executions, execution-specific parameters e.g. payload and complexity of the data which travels between the activity and the service. In order to study this experiment, we use a guideline that is adopted from [40]. This guideline has been presented by Wohlin.

Objects of the study: The object of the study is the prototype of Android framework that includes the ability to execute the service on a remote machine as well as the regular Android framework functionalities.

Purpose: The purpose of this experiment is to investigating the execution time and energy consumption on the mobile phone when the Android application remote service are executed on a remote machine versus same services are executed locally on the mobile.

Perspective: The perspective of researchers and practitioners takes into consideration. Practitioners are interested to know the advantage and disadvantage of executing Android services remotely over locally where as researches are interested in principles that make execution of the Android services on a remote server profitable.

Context: The experiment is conducted by researchers who have developed the prototype.

5.2 Experiment planning

5.2.1 Context selection

According to [40] the context refers to the experiment environment. There is no subject in this experiment but the prototype of the model as the outcome of the case study is considered as the object. Also measuring energy consumption and execution time for local and remote service execution as our tests. Therefore “multi test within object study” have been chosen [40].
5.2.2 Independent variable selection

The independent variables are those variables that we able to control and revise them in conducting the experiment. They also describe the experiment’s treatment [40].

**Independent variables:** execution environment, network, number of executions, payload and complexity of the data unit.

5.2.3 Dependent variable selection

The Dependent variables are those variables, which are observed to spot whether they are influenced by independent variables [40]. Sometimes selecting the exact set of independent and dependent variable(s) of an experiment is difficult due to lack of domain knowledge.

**Dependent variables:** Total execution time and energy consumption on the phone.

5.2.4 Experiment user stories

Below user stories are studied for the experiment:

User story 1: Run an Android service locally on the phone with different data payload and complexity.
User story 2: Run an Android service remotely on a server with different data payload and complexity.

The motivation behind the selection of these user stories and problem is that they can simulate the real world Android application comprehensively. Usually, in real world Android applications, services run various functionalities with different payload and complexity, which result in different execution time for the functionalities in the Android services as well as energy consumption on the smart phone.

5.2.5 Hypothesis formulation

The objective of Hypothesis testing for statistical analysis is to decide whether there is an effect of the value of the independent variable(s) on the value of the dependent variable(s). Accepting or rejecting the hypothesis is based on the result of analysis on the data which is collected during the experiment [40].

**Null hypothesis, H₀:** Total execution time for functionalities with specific payload and complexity remains the same in local and remote service execution.

**Alternate hypothesis, H₁:** Total execution time for functionalities with specific payload and complexity increases in local service compared to remote service execution.

**Alternate hypothesis, H₂:** Total execution time for functionalities with specific payload and complexity decreases in local compared to remote service executions.

\[ H₀: \text{Total execution time } \text{Local} = \text{Total execution time } \text{Remote} \]
$H_1$: Total execution time $\text{Local} >$ Total execution time $\text{Remote}$

$H_2$: Total execution time $\text{local} <$ Total execution time $\text{Remote}$

**Null hypothesis, $H_0$**: Power consumption for functionalities with specific payload and complexity remains the same in local or remote service execution.

**Alternate hypothesis, $H_1$**: Power consumption for functionalities with specific payload and complexity increases in Local compared to remote execution.

**Alternate hypothesis, $H_2$**: Power consumption for functionalities with specific payload and complexity decreases in Local compared to remote execution.

$H_0$: Power consumption $\text{Local} = $ Power consumption $\text{Remote}$

$H_1$: Power consumption $\text{Local} >$ Power consumption $\text{Remote}$

$H_2$: Power consumption $\text{Local}< $ Power consumption $\text{Remote}$

### 5.2.6 Selection of subjects

As there is no subject in this experiment, hence there will be no subject selection.

### 5.2.7 Experiment Design

The planning before the experiment often referred as the design of the experiment. Randomizing the treatment between participants is a principle in an experiment[40]. As there is no participant for this experiment therefore randomization is not applicable here.

### 5.2.8 Standard design type

As there is no subject participated here standard design type is not applicable.

### 5.2.9 Instrumentation

Instrumentation of an experiment mostly encompasses objects, measurement instrument [40]. Experiment instrument used in this experiment are STE 8500, Android server x86, Telia Simcard, Telia Network, Ericsson Network. In following each of these instruments are introduced and described.

#### 5.2.9.1 Android phone lab

STE-U8500[41] will be used as an Android phone in this experiment. The STE-U8500 is a lab smartphone platform created by ST-Ericsson[42], offering a dual ARMv7-A Cortex A9 core, with strong hardware decoding power and ARM Mali 400 GPU. Figure 12 illustrated the U8500 ST-Ericsson phone and table 4. Depicts more specification and features of the phone.
STE 8500 Features & Technologies

* Full HD 1080p camcorder, multiple codecs supported via OMX (H264 HP, VC-1, MPEG-4)
* High-resolution, touchscreen display support up to XGA
* Simultaneous dual display support
* High performance 3D graphics, support for OpenVG 1.1 and OpenGL ES 2.0
* Dual camera support with Integrated ISP 18 Mpixel and 5 Mpixel
* Wi-Fi, Bluetooth and GPS enabled platform
* Built-in USB 2.0, HDMI out
* Highly efficient, low-power ARM dual Cortex™-A9 processor
* Dual multimedia DSP for low-power, flexible media processing
* High-bandwidth LP-DDR2 interface
* ARM Mali™ 400 GPU and NEON® CPU extensions
* State of the art HSPA+ (High-Speed Packet Access) Release 7 modem
* Advanced power saving architecture enabling class-leading audio and video playback times
* Memory technology LP-DDR2

| Table 4 STE 8500 features and Technologies (Adopted from [41]) |

5.2.9.2 Proposed architecture prototype

An enhanced prototype version of Android 4.0.4 has been provided for this experiment. Based on the enhancement that we have done on the Android 4.0.4 (see APPENDIX I, APPENDIX II) the Android framework is able to execute Android services on a remote machine in addition to their regular functionalities.

The new files and modifications to Android 4.0.4 will be provided as a patch from Git[44] source code Repository and patches into Android STE-8500 source code framework. The ST-Ericsson Android framework includes some additional functionalities and components. This additional functionalities and components enable the Android framework to work and being compatible with the ST-Ericsson-8500 platform e.g modem driver.
5.2.9.3 Application for evaluation

An Experiment application developed to evaluate the prototype of proposed architecture. The application is called ‘Sorting’. The intention of the application is to address the comparison of the execution time and energy consumption in local and remote execution on the prototype. ‘Sorting’ is a single thread application which consists of an activity component and a service component. We developed the application according to the assumptions in 4.1.2. The activity is a graphical user interface which includes a text field to accept the IP address of destination server and a button to invoke a specific method on the service. The activity sends an invoke to the service as long as the IP address of the server destination is filled out and the button is pressed by the user. When the button is pressed by the user, activity produce random arrays with specific but different payloads which varies from one byte and 1000,000 bytes and call sorting method in the service along with sending the data and complexity to it. The data includes an array with integer numbers, which are generated randomly in the activity and are executed several times according to the complexity. Complexity is a second parameter of the sorting method in the service. It represents the number of sorting times of the array. The complexity of an array in the experiment means the number of times that the array requires to be sorted in the service. The service includes a sorting method which sorts the array with 100 integer numbers that is received from activity with the indicated complexity. In order to prevent caching mechanism on the result of experience, the array object, which contains the data, is disposed at end of each execution and recreates and populated again for new execution. The higher complexity is reached by repeating the execution of sorting method in the service. The complexity controls the process that varies between 1 and 7 which meets 1 to 10,000,000(log scale 10) represent the repetition of the sorting method in a loop. Table 5. Shows the payload size and the data byte size after marshaling and table 6. Depicts the relation between the complexity and the number of repetition of sorting. The Activity in the sorting application combine each of different payloads with each complexity level of the process to generate 63 scenarios (9*7). In order to have more accurate and stable data for analysis, each scenario is repeated 5 times and the average of this repetition has been considered for analyzing. Therefore, totally, 315(63*5) methods execution are done for every set of tests. The application able to record timestamps in different stages in the course of execution, The timestamps are stored in an array list during the execution time in the client application as well as inside Android OS and server side. These timestamps are collected at the end of each set of experiment in order to be analyzed.

<table>
<thead>
<tr>
<th>Payload size (Byte)</th>
<th>Data Size after marshaling (Byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>106</td>
</tr>
<tr>
<td>100</td>
<td>196</td>
</tr>
<tr>
<td>1000</td>
<td>1096</td>
</tr>
<tr>
<td>10000</td>
<td>10096</td>
</tr>
<tr>
<td>100000</td>
<td>100096</td>
</tr>
<tr>
<td>350000</td>
<td>350096</td>
</tr>
<tr>
<td>700000</td>
<td>700096</td>
</tr>
<tr>
<td>1000000</td>
<td>1000096</td>
</tr>
</tbody>
</table>

Table 5 Payload size and size of byte data
<table>
<thead>
<tr>
<th>Complexity</th>
<th>Sorting Repetition for 100 integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
</tr>
<tr>
<td>5</td>
<td>100000</td>
</tr>
<tr>
<td>6</td>
<td>1000000</td>
</tr>
<tr>
<td>7</td>
<td>10000000</td>
</tr>
</tbody>
</table>

Table 6 Complexity and sorting method repetition

The execution time of the specific steps in the application or group of commands in the application, prototype and the server side are recorded for analyzing. There is a Java Library available to perform this task. The nanoTime()[45] method of the java.lang.System in standard java API library is used to measure execution time. This method returns the current value of the most precise available system timer, in nanoseconds. Basically, The nanoTime() command is stated right before and after the execution of each command or group of commands. The subtraction after and before the execution reveals the execution time of that single or group of commands. Table 7. Depicts the measuring points that record times in specific stages in the test application as well as the prototype and the server side.

<table>
<thead>
<tr>
<th>Framework/Application</th>
<th>Measuring points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Application</td>
<td>Invoke sorting method</td>
</tr>
<tr>
<td>Prototype</td>
<td>Marshaling parcel</td>
</tr>
<tr>
<td></td>
<td>Sending Data</td>
</tr>
<tr>
<td></td>
<td>Receiving Data</td>
</tr>
<tr>
<td></td>
<td>Un-marshaling received parcel</td>
</tr>
<tr>
<td>Android Server app</td>
<td>Un-marshaling Data</td>
</tr>
<tr>
<td></td>
<td>Executing method</td>
</tr>
<tr>
<td></td>
<td>Marshaling parcel</td>
</tr>
</tbody>
</table>

Table 7 Measuring points

5.2.9.4 Server side

The server is an enhanced version of Android X86 4.0 [38, p. 86]. The enhancement has been done by us to able Android X86 execute Android services without the activity component on the server side. The enhanced version or our server side prototype has been installed in the Oracle virtual Box 4.0 [39]. The virtual machine is installed on a X86 machine with 8GB RAM, 16 dual core CPU 2.6M and Ubuntu 10.4[46] as OS. The prototype of the server program has been installed inside Android x86.

5.2.9.5 Power supplier

The phone has been connected to a power supplier, which is configured on 5V out. Also, the power supplier is connected to a desktop computer with installed version of
Matlab[47]. The Matlab application collects the power consumption by the phone from power supplier during the experiment. Figure 13 shows the power supplier.

Figure 13 Power supplier

5.2.9.6 Network

A Telia 4G compatible SIM card has been provided for this experiment. For the remote execution the phone is connected to Telia network and to the server through 3G protocol. The server machine is in the Ericsson Lab network. The Lab network is connected to the Internet through a firewall. The firewall has been configured to allow traffic on port 32000 for the server static IP address. Also, The Telia network detail network configuration was not available in Ericsson AB.

5.2.9.7 Controller

The controller is a Desktop PC in this experiment that is used to start the experiment and also collect the results e.g. logs, CSV files from different devices at the end of each set of experiment. The phone is connected through the USB port to our desktop computer. Desktop computer uses this USB connection to send all information and commands to the phone.

5.2.9.8 Power Trace Script

A python script is used to trace the modem, CPU and memory activities by monitoring the Android kernel and save the result in a log file. The output log file contains information regarding activity of modem, CPU, memory of the STE-U8500 smartphone in the experiment. This script provided by ST-Ericsson and has been customized by us to meet our requirements in this experiment.

Also, There is another python script that is used to interpret the Android kernel logs. This script interprets the log files into a CSV file to be performed in analyzing step. This script is provided by ST-Ericsson and customized by us to meets our requirements.
5.2.9.9 Measurement criteria

We collect the data of the experiment from the computer and devices that are involved in the experiment by an automatic bash script and then create an excel file from all experiment result files as well as logs that have been created during the experiment. Therefore, all data of the experiments are collected from all devices involved in the experiment through running a bash script.

The measurement criteria that have been used for each of result variables are shown in table 8.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Measurement</th>
<th>Motivation for selected measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total execution Time</td>
<td>Subtraction of time before calling the method of service from time after calling service</td>
<td>Total execution of a command or a function of an application is calculated by subtraction of time after the execution from time of before execution of a command or function. In this case the time of execution of the specific method from the local or remote service is considered.</td>
</tr>
<tr>
<td>Energy consumption</td>
<td>Calculate the amount of surface under the chart between sets of functionality execution</td>
<td>A Power supplier provides power for the phone. The consuming power by the smartphone is recorded by Matlab program within the specific periods during the experiment.</td>
</tr>
</tbody>
</table>

Table 8 measurement criteria

5.3 Experiment operations

5.3.1 Preparation

In the experiment preparation process, the environment of the experiment is prepared. In the 1st step a patch of the enhancement are provided from Git repository to import into STE-U8500 Android framework. This patch includes all enhancements that we have done to enable Android OS execute application services on a remote server. In 2nd step the patched framework is compiled and installed on the phone through the USB connection with phone and ADB commands. From this time forward the phone works with our prototype. Two sets of experiments have been settled for evaluating the prototype one for execution of the Android application service on a remote server and another for local execution of the Android application service in the phone lab. The experiment application is the same for both sets of experiments except that in the remote execution application the androidManifest.xml includes the tag that allow the Android able to connect to the remote service instead of the local service. Therefore two versions of the test application are provided and installed on the phone. In the remote service execution experiment the experiment application is connected to the server through the 3G network and the one which executes the service locally. In 3rd step, the server version of experiment application is installed on the server machine inside the Android X86 OS. The power supplier is connected to the phone to provide...
the required power on the phone and also connected to a Matlab program that monitors, collects the voltage fluctuations. The trace script will be uploaded to the phone to be started in execution phase. As shown in figure 14 in the remote execution experiment environment the phone has been connected to the server through the network while in the local experiment environment the phone has not been connected to the network as it is depicted in figure 15.

5.3.2 Execution

Two sets of experiment will be done to evaluate the solution and prototype in terms of execution time and power consumption. During conducting the experiment the execution time and energy consumption will be collected in two courses, one for remote service execution and another for local service execution. As we conduct the experiment multiple times, a shell script has been developed to perform all the process regarding the start, stop and collecting experiment data automatically. The two sets of experiment will be conducted in following order.

- Run the experiment application on the phone and the server; not start it.
- Start the power trace script on the phone.
- Start the Matlab program to collect the phone power fluctuation.
- Start the experiment application.
• Collect the experiment data from the involved devices and experiment application except the data of power consumption from the Matlab program, which is collected manually when the experiment is finished.

In order to have more accurate data, we considered 25 sec wait and a blink screen before each invoke. All data for two sets of experiment regarding the remote and local execution are collected separately right after finishing the experiment.

5.3.3 Data validation

Check the correctness of data experiment is important before analyzing step[40]. We verified whether the raw experimental data is reasonable and the experiment is performed based on the instruction. The data verified manually and also by using the Microsoft Excel and Octave [34] Moreover, The data of the experiment are gathered into a separate CSV files separately, which was easier to be checked and validated.

5.4 Result and analysis

This section of study presents the results and analysis of the experiment according to the descriptive statistics collected from each sets of experiment regarding the local and remote service exaction. The data has been analysis by performing following steps:

First step of data analysis is descriptive statistics. The main aim behind this step is to represent the data set to receive a good understanding of data set as well as identify outlier. An outlier represents a value that is abnormal and unexpected in the data set[40]. There are different ways to represent descriptive statistics.in this study, we plot each data set to identify the outlier and handle them.

In Second step, the identified outliers of each data set will to be handled. Each outlier is handled separately for the data set. This step is called data reduction. In data reduction we studied the results by plotting them and also we replaced the odd results in each set of experiments with the same results from another set .In order to be more accurate we consider the average of the results.

Also, The detail information about the network configuration and network measuring results have been discussed and presented in my colleague's thesis report [49].

5.4.1 Collecting quantitative data experiment

Right after finishing two sets of experiments, two set of files are generated. One set is related to measure execution time and another is related to measuring the power consumption. All raw data related to the experiments are collected automatically by the script or manually from devices. Figure 16 depicts that collecting data from different devices for remote service execution experiment after finishing the experiment. Also, Figure 17 shows the collecting data from different devices for local service execution experiment after finishing the experiment.
The output of the execution time in local and remote service experiment is CSV files that are little bit different with each other. Basically the remote service execution CSV file contains more information such as marshaling, un-marshaling time, sending, receiving time of the data over the network, execution time on the server, which assists us to find out which parts of the solution requires improving. The improvement in the solution led to reduce the execution time in remote service execution experiment. The column names in remote execution CSV file can be seen in table 9. Also the column names of the CSV file for local execution experiment is shown in table 10. The file contains the data regarding the running experiments for specific complexity and payload.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Index of the table</td>
</tr>
<tr>
<td>app_before_app_transact</td>
<td>Time before starting calling service</td>
</tr>
<tr>
<td>Time_before_marshall</td>
<td>Time before start marshaling</td>
</tr>
<tr>
<td>Byte_Lenght</td>
<td>Length of marshaled data in byte</td>
</tr>
<tr>
<td>Time_after_marshall</td>
<td>Time after marshaling data</td>
</tr>
<tr>
<td>Time_after_sent</td>
<td>Time after sending data through the socket</td>
</tr>
<tr>
<td>Time_after_read</td>
<td>Time after reading data from the socket on the server side</td>
</tr>
<tr>
<td>Time_after_unmarshall</td>
<td>Time after un-marshaling data on the phone</td>
</tr>
<tr>
<td>s_after_read_from_socket</td>
<td>Time after reading from socket on server side</td>
</tr>
<tr>
<td>s_after_unmarshal</td>
<td>Time after un-marshaling data on the server</td>
</tr>
<tr>
<td>s_after_transact</td>
<td>Time after finishing calling method of the server</td>
</tr>
<tr>
<td>after_write_to_socket</td>
<td>Time after writing to socket on the server</td>
</tr>
<tr>
<td>app_after_app_transact</td>
<td>Time after receiving result of the service method calling</td>
</tr>
</tbody>
</table>

Table 9 Remote execution CSV columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Index of the table</td>
</tr>
<tr>
<td>app_before_app_transact</td>
<td>Time before starting calling service</td>
</tr>
<tr>
<td>app_after_app_transact</td>
<td>Time after receiving result of the service method calling</td>
</tr>
</tbody>
</table>

Table 10 Local execution CSV columns

There are two files are generated in the experiment to log the power trace of the phone as described in following:
Power supplier log:

Matlab program is connected to power supplier and use it to record voltage fluctuation of the phone. The phone uses this voltage during the execution. This voltage is changing during the course of experiment. Matlab’s log file includes two columns; first column is the log time in millisecond and the second column is the power usage of the phone in voltage.

Power trace log:

The Kernel activity monitoring script, logs the activity of the ARM, Modem and memory of the phone by monitoring the Android kernel activity during the experiment. The content of the log file includes data regarding when the Modem, ARM, memory become active or inactive and also the OPP for each arguments. There is a python interpreter has been developed to parse the log file and extract the specific information to a CSV file. The columns and their description of the generated CSV file illustrated in table 11.

<table>
<thead>
<tr>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Subtraction between the current and previous event</td>
</tr>
<tr>
<td>b2r2_mcde</td>
<td>Screen activity. Value 1 for active and 0 for inactive</td>
</tr>
<tr>
<td>arm_opp</td>
<td>Arm OPP value which can be 7 for opp200, 3 for OPP400, 2 for OPP800, 4 for OPP1000</td>
</tr>
<tr>
<td>ddr_opp</td>
<td>DDR OPP value which can be 7 for opp200, 3 for OPP400, 2 for OPP800, 4 for OPP1000</td>
</tr>
<tr>
<td>ape_opp</td>
<td>APE OPP value which can be 7 for opp200,3 for OPP400,2 for OPP800,4 for OPP1000</td>
</tr>
<tr>
<td>modem_Wake/Sleep</td>
<td>1 if the modem is active .0 if the modem is inactive</td>
</tr>
</tbody>
</table>

Table 11 Power trace CSV column name

5.4.1.1 Execution time

The total execution time of each request functionality with specific payload and complexity for the local and remote experiment data are measured by subtracting the time right after execution from time before the service execution based on formula in Equation 1:

\[
\text{Execution time} = \text{Time after service execution} - \text{Time before service execution}
\]

Equation 1 Execution time

Each function with the specific payload and complexity is invoked five times. Therefore there are five execution times per functionality. The reason behind that is to have more accurate data. Basically due to network latency some of the first requests of functionalities take longer times to be delivered to the server in remote execution. Subsequently, the average of five invokes for each functionality with the same payload and complexity is more accurate. The qualitative data in chart collected after executing. The experiment execution can be seen in figure 18. The horizontal Axis indicates the payload and complexity and the vertical Axis indicates the execution time in millisecond, which has been rounded. In horizontal Axis, the figures on the left side of the parentheses represent the payload, which starts, from 0 for one byte payload and 1 for ten byes payloads etc. and also, the complexity figures on the right side of
parentheses, which starts from 0 indicates complexity one (ten times sorting repetition), and 1 indicates complexity two (one hundred times sorting repetition) etc. The total local and remote execution times for each payload and complexity is shown in pair together with different bar colors. The pair of remote and local execution for each functionality with specific payload and complexity is represented together with blue and green bars to facilitate the comparison and interpretation.

It’s apparent that the execution times for the functionalities, which are executed in the remote service, encompass marshaling and un-marshaling data on the phone and the server side as well as traveling data on the network.

![Descriptive chart for Execution time](image)

Figure 18 Descriptive chart for Execution time

Based on figure 18 the total local execution time is always less than total remote execution time for functionalities with payload from 0 to 8 for different complexities from 0 to 3. When the complexity become 4,5 or 6 for all functionalities in different payloads, the total local execution time of functionalities increase gradually for complexity 4 but experience a drastic increase for functionalitites with complexity 5 and 6. For instance total local execution time for the functionality (3,1) that has been represented with green bar is less than the same remote execution functionality that is shown with blue bar prior to the green bar. However, the local service execution time for the functionality (3,6) that is represented in green bar, is much greater than the remote service execution of same functionality that is presented in blue bar prior to the green bar.

The outliner of data was checked to find out whether the experiment contains inconsistent data point. In order to confirm whether or not data point is an outliner the plotted data of the experiment regarding the execution time are considered to figure out the inconsistent data. An execution time for functionalities are considered as an outliner as long as the difference between the data with same functionality in similar
former experiment results is high. The outliers handled by substituting rational data with our former experiments data that is reasonable.

5.4.1.2 Energy consumption

Energy consumption in mobile phones refers to the amount of energy that is spent by a mobile phone to execute functionalities with specific payload and complexity locally or remotely. In order to calculate the energy consumption for each functionality with specific payload and complexity, the power supplier data collected from Matlab program and plotted for the remote and local service execution separately with using Octave[34]. Additionally, following steps have been performed to calculate the energy that has been consumed by executing functionalities on the phone. The steps have been considered for the local and remote service execution separately:

- Math total power that has been consumed for each method invocation window
- Math total power that has been consumed for each invoke
- Math total power without background energy
- Math energy without background per invoke

During the local and remote execution experiments there was 20-second wait and a screen blink considered in the experiment application before each request. The reason behind having the blink and the wait was to identify begin and end of each method invocation. In this case, distinguish between each service method execution in the plotted data is become simpler.

In the first step, after plotting the power supplier data, we defined border of each invoke for each functionality. A small Matlab script that had been developed for this purpose performed this action. We name each functionality a window. The window includes the screen blink and waiting time as well as background power.

In the second step, we exclude the blink and waiting power from each window. This process is performed manually by looking and examining each window to find the beginning and end of each invoke. At the final of this step, we collect the power consumption for each invoke that includes the background power.

In the third step, the total energy consumption per functionality with a specific payload and complexity is calculated by using below formula in Equation2.

\[
\text{Total energy consumption for each functionality} = \frac{\text{Total without background energy}}{5}
\]

Equation 2 Total energy consumption in each functionality

The comparison between the local and remote service execution energy consumption can be seen in figure 19,20. Figure 19 shows functionalities with complexity 0,1,2,3 and figure 20 illustrates functionalities with complexity 4,5,6. The horizontal Axis indicate the time in millisecond and the vertical Axis indicates the functionalities with specific payload and complexity. In vertical Axis, the figures on the left side of the parentheses represent the payload, which start, from 0 for one byte payload and 1 for ten bytes payloads etc. and also the complexity figures on the right side of parentheses, which starts from 0 indicates complexity one (ten times sorting repetition), and 1 indicates complexity two (one hundred times sorting repetition) etc.
Also, the total energy consumption of functionalities is shown with identical colors in pair bars. The pair of remote and local execution energy consumption for each functionality with specific payload and complexity is represented together with red and blue bars to facilitate the comparison and interpretation.

As seen in figure 19 total local service execution energy consumption for functionalities with complexity 0,2,3 and payloads 0,1,2,3,4,5,6,7,8 are much less than total remote service execution for same functionalities. The reason behind it is that the energy that is consumed for sending the intensive computing functionality to the server as well as execution on the remote service and receives the results back from server consumes more energy than execute the service locally for the same functionalities. For instance total local service execution energy consumption for the functionality (7,3) that has been represented with blue bar is less than the same remote service execution functionality that is shown with red bar.

However, in figure 20, total remote service execution energy consumption for functionalities with complexity 4 and payload 0,1,2,3,4,5,6,7,8 have been decreased noticeably. This reduction becomes much more noticeable for complexity 5,6. The reason behind it is that, The energy that has been consumed for sending the intensive computing functionality to the server as well as execution in the remote service and receives the results back from server to the activity component consumes less energy than execute the service locally for the same functionalities. For instance, the local service execution time for the functionality (1,6) that is represented in blue bar is much greater than the remote service execution of same functionality that is presented in red bar.
Figure 19 Descriptive data for Local VS. Remote service energy consumption for complexity 0,1,2,3
Figure 20 Descriptive data local Vs. remote service execution energy consumption for complexity 4,5,6

The outliner of data was checked to find out if the experiment contains inconsistent data point. In order to confirm whether or not data point is an outliner the plotted data of the experiment regarding the energy consumption are considered to figure out the inconsistent data. Energy consumption for functionality considers as an outliner if the difference between the data with same functionality in similar former experiment results is high. The outliners handled by substituting rational data with our former experiments data that is reasonable.

5.5 Data set reduction

In a experiment result, some experiment data point are too large or too small than expected in comparison with other values because of outliners[40].

In this experiment, each of the functionalities with specific payload and complexity were executed 5 times. The final result for every functionality was the average values of the executions. Those data that was suspected to be the outliner are checked again to assure that they are correct. Therefore no data are excluded from the experiments because of being outlier.

5.6 Hypothesis testing

The objective of hypotheses testing is to reject the null hypotheses according to data set analysis[40]. The aim of this section is to test the entire formulated hypothesis based on the data set.
### 5.6.1 Execution time

According to the data that has been collected and plotted in figure 18 for local and remote service total execution time, the execution time for the functionalities with the identified payload and complexity is not the same for the local and remote service execution. Total execution time for local service is greater in comparison with remote service for all functionalities with complexity 4,5,6 than functionalities with complexity 1,2,3 and visa versa. Therefore the null hypotheses are rejected. Hence, all alternative hypotheses are rejected.

Based on the statistics presented in earlier sections if the complexity is high in a functionality, it might be better to execute it in a remote service because the execution time is much more faster in remote service execution for these kind of functionalities. On the other hand if the functionality has less complexity, it is performed faster as long as it is executed in a local service. In our experiment functionalities with complexity 4,5,6 are executed faster in the remote service and functionalities with complexity 0,1,2,3 are executed faster in the local service.

### 5.6.2 Energy consumption

According to the data that has been collected and plotted in figure 19,20 for local and remote service total energy consumption for the functionalities with identified payload and complexity the energy consumption is not the same for the local and remote service execution. Total energy consumption for local service is greater in comparison with remote service for all functionalities with complexity 4,5 and 6 than functionalities with complexity 0,1,2 and 3 and visa versa. Therefore the null hypothesis is rejected. Hence all alternative hypotheses are rejected.

Based on the statistics presented in earlier sections if the complexity is high in a functionality, it might be better to execute it in a remote service because the energy consumption time is much more less in remote service execution for these kind of functionalities. On the other hand if a functionality has less complexity it consume less energy as long as it is executed in a local service. In our experiment, functionalities with complexity 4,5,6 have consumed less energy on the phone when they are executed in the remote service and functionalities with complexity 0,1,2,3 have consumed less energy when they are executed in the local service.

### 5.7 Discussion & Analysis

In order to analysis, represent the results of experiments and explain them, we decided to use Heap map. HeapMaps provide a simple visual summary of information which allows the spectator understand complex data set[33].It represents the value of the corresponding elements of data matrix with a shaded tile on a color scale. The motivation behind using the heap is to observe variation of data regarding the execution time and energy consumption on the phone by variation in colors.
5.7.1 Local vs. remote execution time

Total execution time in our experiment is the subtraction of request and response time for a service method.

Figure 21, as a heap map depicts the ratio of total execution time of a service remotely to the total execution of the same service locally for the functionalities with specific payload and complexity. Each distinctive color in the heat map reflects the ratio between local and remote execution. The Ratio calculated by equation 3.

\[
\text{Ratio} = \frac{\text{Total Local Time execution}}{\text{Total Remote Time execution}} \times 100\%
\]

Equation 3 Ratio formula for local vs. remote service execution time

The heap map consists of two Axis. Horizontal Axis divided into seven different equal stages. The stages represent the complexity of the functionalities that are executed in local and remote services. The complexity has been ordered from left to right from less complexity to more complexity. In other words, by growing the complexity from left to right in each stage, the computation of the service for each requests increases ten times from the previous stage.

The Vertical Axis represents the data size of data travelling between the phone to the server and visa versa that named payload. The payload has the same size in both directions. As shown in the figure 21 the Payload Axis is divided into 9 stages. The payload size grows in the logarithmic order from 1 byte to 1,000,000 Bytes from left to right. The 1 MB is the limitation for local execution in AIDL because the data size more than 1MB is not handling by the service and Android throw the TransactionTooLargeexception[50].
Figure 21 Ratio of total execution time of the service remotely vs. locally

The vertical legend on the right side of the chart indicates ratio with different unique color varies from 0 to 2000. The area with deep blue color on the heat map indicates that the ratio is less than 100% which express this fact that execution the service locally take less time for those payload than sending them and execute them remotely on the server and send the result back to the smart phone. In other words, the total execution time for this sort of functionalities is faster on the phone locally than remotely. Hence it would be beneficial if they execute locally.

As far as colors on the heap map turned toward lighter colors such as yellow or red, the ratio becomes more than 100% as seen in the legend. The lighter colors on the heat map indicates that local execution of the service for that specific payload and complexity take more time as it sends to the server, get executed out there and the result sends back to the phone. In other words, for the functionalities with the ratio greater than 100% or better to say blocks with lighter color with specific payload and complexity total execution time is faster on remote execution. Hence it's more beneficial if the data sends to the server, executed and the result sends back to the phone. For instance, the red color block on the heat map, which represents the ratio greater than 1800%, the total execution time is more than 18 times faster for that functionality with specific payload and complexity in comparison with executing it locally.

The white line on the heat map represents the border of functionalities that are beneficial whether executed remotely or locally. Basically, the white line can be considered as the decision line. Practically, functionalities to the left side of the border are executed faster and are more profitable if they are executed on local services and the functionalities on the right side of border are executed faster as long as they are performed in the remote service.
By examine the heap map, it’s understandable that the amount of ratio varies and is influenced by changing the amount of the complexity and payload. Changing amount of the ratio on the heap map for some certain payload and complexity is interesting to be considered and explained. As shown, when the complexity is around 6,7 which means in the highest stages and the payload is low (around 3) the color of conjunction block turns to red. In this case, executing the functionalities on the remote service is faster than it is executed locally. For instance, functionality with payload stage 3 and complexity 7 achieve around 18 times execution time faster on the remote service in comparison with executing the same in local service. Hence it would be more beneficial in terms of execution time as long as these kinds of functionalities run on the server side. Generally, according to the heap map we can admit that for the functionalities with lowest complexity and highest payload which represent in the left up corner blocks of the heat map, ratio is lowest because marshaling and transferring the high payload size data to the server over the network take more time than executing them in local service on the same phone. Unlikely, the functionalities with highest complexity and lowest payload (right down corner) are executed faster on the remote service subsequently. This occurs because the server side has more powerful computing resource than the phone to handle complex functionalities. Furthermore transferring small payload size to the server and receiving the small size of results is faster which result in faster execution time. On the other hand, executing the functionalities with high payload and low computing is faster on the local service and execution of functionalities with low payload and high complexity is faster by transferring the functionalities to the remote service.

On the heap map, unlike the expectation to see highest ratio (dark red) in the right down corner where the payload is the lowest and complexity is the highest, the highest ratio is in blocks with payload 3 and complexity 7. Because the sending and receiving time of data, results on the phone and server side for 1 or 10 bytes is longer than the same for 100 or 1000 bytes which result in slower execution time for those functionalities. Furthermore the TCP/IP overhead is higher for 1 or 10 byte in comparison with 100 or 1000 bytes.

As outlined in figure 21, The decision line (white line) is not a straight line and tends to be a curve in complexity 4 with payload 2,3,4 and 5 then again it returns back to the straight line in complexity 5 with payload 6. We investigated on this phenomenon from different angels to understand why it has been occurred. During the experiment the size of socket buffer for sending and receiving data set to 2048K in smartphone and server code. According to [51][52] TCP/IP’s latency and processing overheads should be considered in using TCP/IP protocol. Also, Processing overheads depend on the message size. Large or small data sizes that send or are received through TCP protocol over network have direct effect on the latency and throughputs. Also we figured out that control amount of buffering is done to have a better performance in socket and less traffic on the network. It tries to send full data segment by waiting. In our experiment the optimized situation for low-latency and high throughputs occurs in payload 3,4,5 for complexity.

5.7.2 Local vs. Remote energy consumption

The energy consumption of the phone in the experiment diverse due to the complexity and the payload of the functionalities that are executed in local service or send to the server to be executed in remote service.
Figure 22 as the energy consumption heat map depicts the percentage ratio between the total energy consumption of functionalities execution with specific payloads and complexity locally to the total phone energy consumption of the same functionalities in the remote service. The ratio is calculated with Equation 4:

\[
\text{Ratio} = \frac{\text{Amount of phone energy consumption by local service execution}}{\text{Amount of phone energy consumption by remote service execution}} \times 100\
\]

Equation 4 Energy consumption Ratio

The Vertical Axis represents the data size of data travelling between the phone to the server and visa versa, which named payload. It divides into 9 stages with the same size that varies from 0 byte to 1,000,000 bytes. Also Horizontal Axis represent the complexity of the executed process handled by the service remotely and locally. The complexity has been ordered from less complexity to more complexity. In other words, by growing the complexity from left to right in each stage, the computation of the service for each requests increases ten times from the previous stage. The horizontal Axis divides into 7 stages.

The legend on the right side of the heat map depicts the ratio with unique different colors.

Based on the heat map the energy consumption by the phone for functionalities with high complexity and the low payload is less than when the same functionalities are executed locally. The ratio of these blocks are more than 100% .

When the complexity is lower and payload is higher, the ratio is less than 100%. The blocks with ratio less than 100% are colored by dark blue. This means that the phone consumes less energy for executing functionalities with higher payload and less complexity in comparison to functionalities with less payload and high complexity.

The phone energy consumption amount is the same for executing functionalities on the white borderline, remotely or locally. On the other hand, the same energy is consumed in the phone whether functionality are transferred and executed on the server or executed in the phone. The functionalities with the specified payload and complexity to the left side of the border, consumes less energy if they are executed in the remote service and functionalities on the right consume less energy if they are executed locally in local services.

By comparing the heat maps of energy consumption and total execution time in figure 21 and 22, it’s observable that with rising the execution time the energy consumption is increasing subsequently and vice a versa. This is noticeable by matching colors and ratio in both heat maps. It’s apparent that functionality with longer execution time consumes more energy than functionality with shorter execution time.
Also, we studied CPU, modem and memory activities on the total phone energy consumption to figure out the possibility coloration between them and the total energy that is used by the phone. To do this purpose, the power trace log and power supplier data are studied together. The power trace log data file records the CPU, modem and memory activity. The log file is collected after finishing the experiments from the phone. A python script is developed and used to interpret the data in the log file and convert data to the CSV format, which is more appropriate for plotting. A Matlab script is developed and used to plot the power trace content as well as power supplier data. The two graphs aligned on each other with helping wait times and screen blink as well as cpu and modem activities which were recognizable on graphs. We studied the fluctuations of the power consumption for each functionality with the specific payload and complexity. The results showed the correlation between the CPU and modem usage of the phone with the total phone power consumption. Based on the plotted charts the arm(CPU) and modem become active when the invoke is initiated and data is submitted to the server and stay active till the result is received from the server side. After examining the fluctuation in all of functionalities in the local and remote execution, it can be concluded that there is a correlation between the CPU and modem usage on the phone and the total phone energy consumption. Basically CPU usage is hand in hand with total energy consumption on the phone. In other words, when the CPU, modem or both become active the total energy consumption rises drastically.

The explanation for wiggling the decision line (white line) on the energy consumption heat map is same as the execution time heap map. In this heatmap the total energy consumption for remote execution of functionality with complexity 5 and payload 3,4,5 is less than the total local energy consumption for the same complexity and payload.
Figure 23 shows the phone energy consumption for execution of functionalities with the specific payload. The vertical Axis indicates the energy in joule (J) and the horizontal Axis indicates the specific payload. The figure 0 for payload indicates one byte and figure 1 indicates ten bytes etc. The energy consumption for each payload from 0 to 3 is approximately consistent with minor fluctuation. The reason behind it is that the size of the packets for payload 0 to 3 is not bigger than the 3G bandwidth therefore the OS doesn’t need to split up the data into smaller packets and send them to the server. From payload 4 to 8 (from 10,000 bytes to 1MB) the energy consumption for each payload consistently increases. Basically, with increasing the size of the payload more energy has been consumed to send the data including the payload to the server side because the size of the data become more than the bandwidth size. Therefore the Android OS requires breaking the packets into smaller size and send them to the server. This results in spending more energy from the battery.

5.8 Discuss about the solution in real world applications

The conducted experiment mostly focused on simulating execution of the real world applications in the Android domain. The simulation was based on payload and the complexity exchanging between activity and service in Android applications. Figure 24 shows the use case of real world applications that were simulated by our experiment. According to the results of our experiment, the applications on the right side of the border line mostly with high complexity and less payload are more profitable in terms of power consumption and total execution time if they are executed in the remote service and the application on the left side of the border are more beneficial whether they are executed on the phone. For instance in video editing industry, user who uses our solution in their Android mobile video editing application would be able to obtain benefit of executing intensive computing the whole process of application in a remote server which lead to reduce execution time for processes. Furthermore, application developers would be able to execute video editing application that require intensive computing and use our solution to run them on an Android smartphone without worrying about the low execution time for intensive processes.
However, sometimes execution of an application is not profitable from energy consumption and execution time perspective whether the whole processes are executed on the server side or on the smartphone. However, execution of the same application is profitable if just intensive computing processes run on a remote powerful machine and the others execute locally on the smartphone. This idea allows the application developer to design complicated games, which require intensive computing for executing. For instance, some games are not able to be executed on smartphones due to high graphic processing. In this case, game developers would be able to use our solution to execute the high-intensive process on a remote server and the other processes with less processes on the smartphone. This results in executing more complicated games on the smartphone as well as profitable from the execution time and performance.

5.9 Future experiments

For the future experiment, we will do these both execution time and energy consumption experiments with LTE phones and compare the results from 3G and LTE together to figure out how much improvement we will achieve by using LTE phones over 3G phones.

5.10 Validity threats

5.10.1 Internal validity

Some times during conducting an experiment the dependable variables may be affected with some factors, which ultimately lead to affect the final result of the experiment. Observing these factors is the concern of Internal validity[40].
The threats regarding the number of execution, payload and complexity of each functionality of this study considered low since the number of execution are the same for all of functionalities with different payload and complexity. Also, they are monitored and recorded during the experiment to be checked at time of collecting data.

Execution environment is another important threat that is associated with internal validity. The network communication is one of the main factors that was involved in this study. There was no control on the network outside of the Ericsson network Lab environment. This high rate is not expected to affect the outcome since the set of tests are done more than four times in different time of daylight and the result were the quite similar to each other. There was the reason behind executing functionalities for specific payload and the complexity nine times as well as taking the average of the results.

5.10.2 External validity

External validity refers to the ability to extent inferences the result of the experiment to the other situations. First threat is related to the operating system constrain. The prototype of the solution is implemented in Android framework as a built-in functionality. Also, the experiment conducted on this prototype. Therefore, it is not possible to generalize it into other mobile operating system in this family such as Apple iOS[51] for Apple iPhone. For this reason, we limited the results of the experiment just to Android operating system. Second threat is related to the Android framework version that is used in this study. As mentioned, the experiment is conducted on the customized version of Android framework from ST-Ericsson and patched according to Android 4.04, which was the last version at the time of starting this study, and never tested on other prior versions. To eradicate this we just changed those files and functionalities, which were in common in all Android versions from 2.5 till 4.04. Therefore we can be sure that the mechanism works on the prior versions in all Android phones from different brands. The type of phone that is used in this experiment can be a threat to the external validity because it’s not look like the regular phones. But from functionality point of view it’s the same as the regular Android phones.

5.10.3 Construct validity

Construct validity refers to relationship between the experiment and theories behind the experiment in terms of what is measured and affected[40].The threat is related to the application that is used for this experiment. Unlike the other researches in this domain such as [12][15][26] the regular well-known applications such as Suduko or Android face detector are not used to evaluate the proposed prototype in terms of the total execution time and the energy consumption. To nullify this threat, we developed an application which simulated the real situations in real world application with different cases in terms of the size of payload and complexity .The metrics strongly correlated with each other properly which can explain the relation between the execution time and energy consumption on the phone for real world applications.

5.10.4 Conclusion validity

Conclusion validity refers the possibility to draw correct or reasonable conclusions regarding the relationship between treatments and the result of an experiment[40].From the experiment data perspective, a potential validity threat is the
reliability of energy consumption statistic data collection. The energy consumption data partially extracted manually because the octave script that implemented for this purpose were not accurate. To minimize this risk, we double checked the collected data and also compare it with other results from the same experiment to assure the related data are collected properly. Another threat to the conclusion validity is the concern about adopting appropriate instrument for the measurement. To eliminate this concern we discussed with the industry advisors who confirmed that the instruments are appropriate for this purpose.

5.11 Conclusion and Future work

In this study, we propose a framework model aiming to remove constrains of Android devices in terms of execution time and energy consumption regarding Android applications. It enables the Android phone device to execute the Android services on a remote server as an enhanced functionality of Android OS not as an application that can be deployed on the application layer of Android framework. The design decisions are driven by intention to modify the underlying Android platform without modifying the application source code. In addition, we have implemented a prototype of the solution and evaluate and examine the application execution time and energy consumption on a Android mobile device.

There are a set of directions that need further research efforts. First of all, our solution is limited in some respect by its inability to convert the images to byte code in underlying level of the Android OS. Consequently, if one were to send an image at a point in the execution to the service on the remote server, the image object on the destination wouldn’t have the entire content same as source object. Further, taking the optimized decision regarding the execution of service remotely on the server or locally on the device is challenging. As another issue, data security of the services running on the remote server is essential problem for many available solutions. Furthermore, execute the service on the cloud service in order to achieve the advantage of the cloud computing is demanded. We have provided a version of server side solution to deploy on the Amazon EC2. However we couldn’t deploy it on the cloud due to the shortage of time. Therefore, it can be considered as an open issue. Also, we conducted our experiment with 3G phone that is significantly slower in terms of data transfer speed in comparison with LTE phones. Therefore conducting the experiment with a LTE phone and involve other factors such as network bandwidth can reveal the more beneficial of executing the services remotely in LTE phones in terms of application execution time and phone energy consumption.
6 ANSWERS TO RESEARCH QUESTIONS

During the course of this study, a new model proposed for executing an Android application services on a remote server. Additionally, the prototype of the model implemented to be evaluated. The case study is performed to explore and enhance the Android functionality to execute the Android application services remotely as well as locally along with presenting the prototype. Moreover, an experiment conducted on the prototype of the model to explore the key factors that make to execute computation locally on mobile devices as well as remotely on the remote server.

In this section, each research question is answered concisely by mapping the relevant results to the questions. The description of each answer provided based on the work has done in this research.

6.1.1 Research question 1

What modifications and extensions to the Android OS framework are necessary to enable applications to partially run on a remote server?

In the course of case study, research presents the details of proposed model as well as the packages; classes and methods that require to be modified and enhanced on the Android OS to enable Android application services are executed on the remote server without code modification. The AIDL mechanism that is used for communicating two or more process together through IPC takes the major role in the solution. The transact method in Binder class in java API wrapper layer of Android OS as the key method of IPC has been enhanced for this purpose. The enhanced transact method uses a socket mechanism based on TCP/IP protocol to communicate with the remote server. This communication includes delivering the computation to the server and receives the results after the execution. Of course there are some other classes, methods added and modified in Android OS to accomplish this that are addressed in the case study.

In the server side, our server program installed on an enhanced version of Android x86 OS that uses socket to communicate with the mobile device and AIDL mechanism to execute the computation in the service and return back the results to the mobile device.

In order to use this mechanism the Android application developer just requires to use a special tag in the Manifest xml file that represent the specific service(s) as the remote service(s).

6.1.2 Research question 2

What is the effect on resource consumption, energy consumption and performance on mobile phone when part of the application is running on remote server in comparison with running locally?

As mentioned in research question 1 answer, an Android application that uses our model and follow the assumption to develop the application be able to execute service component on the server side. Based on the experiment results, when the service component of an application is executed in the remote server, mobile phone requires to send the computing to the server application. The results of our experiment represents
that in the course of sending the computing data to the server mobile consume energy to deliver the computing data to the server side. The CPU and modem on the phone become active and stay active till the result is received back on the phone from the server. Also, we figured out that there is a tight correlation between CPU activity and mobile energy consumption. We realized that the more CPU activity result in the more energy consumption on the phone. According to the results by increasing the payload from x to x+1 the energy consumption increases meaningfully. This growth become more noticeable when the size of the data that is supposed to be delivered to the server becomes bigger than a specific size. In this case, energy consumption on the phone increases gradually by increasing the size. The reason behind it is that phone requires to marshal /unmarshal the bigger data in size and also send more data packets to the server. This results in increasing the energy consumption on the phone.

However, when the service component is executed locally on the phone, almost all of the power consumes by the CPU to execute the functionalities. By increasing the complexity, the execution time increases drastically that lead to grow energy consumption on the smartphone significantly.

6.1.3 Research question 3

What principles govern whether parts of an application should run locally or on a remote server?

The results of the experiment also evaluate execution time as well as energy consumption for executing the computation in local service in comparison executing the same service on a remote server. The result analysis of the experiments indicates that there is a strong correlation between the payload and complexity as well as energy consumption and execution time. The execution time and energy consumption for the functionalities are fluctuating by different payload and complexity. Therefore it’s reasonable to conclude that offloading the computation to be executed on the remote server is beneficial as long as large amounts of computation (complexity) are required with relatively small amounts communication (less payload).
7 REFERENCES


[37] T. Schreider, “Android Binder–Android Interprocess Communication,” Ruhr-
[44] “Novathor STE-8500 photo.” [Online]. Available: http://www.google.se/search?q=STE-8500&source=lnms&tbm=isch&sa=X&ei=TPSHUdOHEsuO4gTM0YGwAQ&ved=0CAcQ_AuOAQ&biw=1440&bih=382#tbm=isch&sa=1&q=Novathor+STE-8500+photo&oq=Novathor+STE-8500+photo&gs_l=img.3...10755.19007.2.19399.13.11.2.0.0.0.80.542.11.10.0...1c.112.img.hl7EG14xvw&bav=on.2,or.r_qf&bvm=bv.45960087,d.BGE&fp=d562e5ef4e18504a&biw=1440&bih=382&imgrc=nlAWzcue9mm0A%3A%3BUBFUDzWzzjv-IM%3Bhttp%253A%252F%252Fimages.vidimg.popscreen.com%252Foriginal%252F10%252FeG1pd3v2MTI%252D_o_st-ericsson-novathor-u8500-with-ice-cream-sandwich-clipping.jpg%3Bhttp%253A%252F%252Fwww.popscreen.com%252Fv%252F6vSpr%252FSST Erie son-NovaTh or-U8500-with-Ice-Cream-Sandwich-Clip-1%3B640%3B480. [Accessed: 06-May-2013].


8 APPENDIX I

8.1 Android

Android is an open source Linux based framework under Apache licensed designed primarily for mobile devices, including an operating system, middleware and core applications. The Android developed by Google incorporation with Open Handset Alliance. The first version of Android released on fall 2008 and become very quickly expanded and used in industry because of being open source nature and its architecture [53][54].

8.2 Android architecture

The Android system architecture is illustrated in figure 25. It contains five layers:

- **Application Layer**: A set of core applications shipping with Android including SMS program, Calendar, maps, browser, Camera, music player and many more. All of these applications are written using the java programing language [56].

- **Application framework**: The same framework APIs are used to develop core applications are fully available for developers. The purpose of the application architecture is simplifying the reusing of all components. This design provides a facility for the user to reuse the components of an application, which is developed and published earlier (with considering security constraint enforced by framework)[56].

- **Libraries**: There is a set of C/C++ core libraries in Android that are used by different component of Android systems. These libraries are accessible can be used by developers through the Android application framework[56].

![Figure 25 Android System architecture (Adopted from[55])](image)
• **Android Runtime:** Android Runtime consist of Core java libraries and Dalvik virtual machine: core libraries include functionalities that are presented in essential libraries of java programing language. Dalvik virtual machine (DVM) is a one of the major component in Android framework. It runs java program apps. Every Android application runs on its own process and takes own instance of Dalvik virtual machine. Dalvik virtual machine is designed to allow running multiple instance at the same time efficiently in term of memory and CPU usage. As a register-based VM, the Dalvik VM runs specific file format which called .dex(Dalvik Executable).The .dex format is generated at compile time by “dx” tool that comes with the SDK. Also, Dalvik VM relies on Linux kernel functionalities such as threading and low-level memory management to be able to run in multiple instances [56].

• **Linux Kernel:** Android uses Linux kernel (standard Kernel version 2.6), which enhanced with new extensions for mobile requirements. It has several modules such as logger, Alarm, Binder, power management, security, driver model, and kernel debugger. In addition, the modules play as hardware abstraction layer between the hardware and the app[56].

### 8.3 Android application anatomy

Running apps is a main task of operating systems. In order to do this purpose, Android uses different components or building blocks. Android defines four different types of building blocks. Each building block exists as a unique entity and plays a certain role with a distinct goal in an application but can depend on one another. Despite may not an application use all of them; still some of them can be combined get used by the user.

The four Android building blocks are described as following:

#### 8.3.1 Activity

An Activity is an Android component that provides user interface to interact with end user in Android application. Sometimes the application includes more than one activity components to for different purposes. They can return result values to the invoking components once the execution completed[57].

#### 8.3.2 Service

Android offers service for long-lived run operations in the background without user interaction[57]. Background of Music player or file downloader can be operated in the service. In music player user select and play music through the activity screen. In the meanwhile user may work with other application by navigating to new screen while music keeps playing. It is the same in file downloader, while the file is downloading user may work with other application(s).
8.3.3 Broadcast Receivers

It receives the intents that broadcasted to multiple applications at a same time. The receivers are started running in the background to perform a specific event that is expressed in the intent. It may just alert user by displaying notifications if something interesting has occurred e.g. Low battery level alert. Usually receivers are not long-lived; they regularly convey messages to services or activities of the applications. There are three type of broadcast intent available in Android:

- Normal: They are vanished, once they sent to all registered receivers.
- Sticky: They stay available even after they have been delivered to receivers, to be able to re-broadcast to upcoming receivers.
- Ordered: Only one receiver at a time becomes active by the broadcaster[57].

8.3.4 Content provider

Provide a facility for applications to store their data in a persistent data storage (e.g. SQLite) and may share it with other applications [57].

Figure 26 shows hierarchical java Class diagram of Android components.

![Hierarchical Class Diagram of Android Components]

8.4 Intent

Intent is the Android high-level message passing system to link applications together [57]. Basically "It is an abstract representation of an operation to be performed." [58].
Intent can be considered as a self-contained object that the performer of desired operation is not defined in that and includes just specification of the remote procedure aim to be invoked along with the associated arguments. Intents are used by applications for the inter application communication and communication in between two or more applications. Moreover there is a mechanism in Android to broadcast intent as event notification. This messaging mechanism is called system broadcast intents [57].

There are two different forms of intents:

- **Explicit intent**: the delivered component in the intent is explicitly specified. In other words, the intended recipient identified by the name in the intent[57].

- **Implicit intent**: It delivers to any component that supports the desired operation. To be simple, in implicit intent Android OS determine which address the receiver component[57].

Intent can be sent to building blocks in Android. They can be used to start activities, start, stop and bind Services, and broadcast information to Broadcast receivers implicit or explicitly [57].

Intent holds two most important piece of information: Action and data (URI) to act upon. The URI uniquely addresses the application component and action addresses the operation to be performed.

### 8.5 Inter-process Communication (IPC)

Inter-process communication is a mechanism that uses a set of specific methods that provide a communication platform in Linux to exchanging data between multiple threads in one or more processes. The processes are on the same or multiple machines connected together through a network[37]. There are several methods for IPC in Linux systems such as pipes, message passing, shared memory, signals, sockets and semaphores however, just two of them are used in Android. Below we states and briefly discuss the IPC mechanisms which is used in Android:

- **Shared Memory**: A location in system memory can be simultaneously accessible by two processes. This is a very efficient way of communication as opposed to other mechanism (e.g. transferring images between processes).

- **Signals**: An asynchronous notification submits to a process with same user id and group id or to a specific thread within a same process to notify it an event.

### 8.6 Android component communication

Android components belongs to separate process communicate and exchanging data with each other through IPC and if they are in the same process, inter component communication is used for this purpose[37]. The communication is initiated by Intents .The address of destination component and operation that need to be performed are loaded up in URI and action of the Intent and submitted to the destination through the inter-component communication or IPC.
The different form of communication between Android components are described as follows:

Figure 27. Shows, the interaction of two activities; an activity start another activity by submitting an intent to it.

Figure 27 Starting an activity

Figure 28. Illustrates the interaction between an activity and a service; an Activity can start, stop and bound to a service through IPC. Invoking the methods in the bounded service and returning the results to the activity is handled by IPC as well.

Figure 28 Activity and service Interaction

The communication between an activity and a content provider is handled by IPC. This communication includes querying and results (see figure 29)

Figure 29 content provider interactions

Submitting all intents intent from subscribed component to the Broadcast receiver is done through IPC (see figure 30).

Figure 30 Interaction with Broadcast receiver
8.7 Binder

Binder is a remote procedure call (RPC) IPC mechanism that allows a process to remotely invoke and execute a subroutine or a function in an address space of another process in Android. It uses the shared memory mechanism in Linux kernel API to transfer payload to the target process[59].

The communication among the processes and components in Android is done through the Binder e.g. Activity to activity, activity to service and etc. The high level APIs for using the Binder are available in the application framework layer of the Android OS and also accessible for the developers. For instance, Binder can be used to start/stop, bind/unbind a service as well as invoking the method of the Bind service using high level Android APIs in application framework for this purpose.

8.7.1 Binder Terminology

To understand the Binder framework, some specific terminologies are stated to term components and other parts[37]. Here in the sub-section the significant terms are briefly explained.

- **Binder**: “The” Binder represents the overall architecture however; “a” Binder represents a specific implementation of a Binder interface.

- **Binder Object**: is an entity or generally an instance of a class, which implement the IBinder (Binder interface). An object Binder might derives from multiple binders [60].

- **IBinder interface**: It’s an interface, which includes definitions of events, properties and methods designed for high performance lightweight remote procedure call mechanism. It’s suggested not to be implemented directly but instead extend from Binder[60].

- **Binder Protocol**: It’s a very low level protocol that is used by IPC to communicate with the Android Linux kernel driver[60].

8.7.2 Main Binder facility and feature

Those facilities that binder framework has been offered is more than a simple messaging system mechanism.

The main advantage of the using binder is that the remote methods can be invoked, as they are local methods. Client invokes the remote method synchronously or asynchronously and receives the response from the remote method. In synchronous method call client sends the message to the server and wait for response. Basically client process is blocked till the answer is delivered to client from server. In asynchronous method call, the client no needs to wait and assign a thread for return message from the server.

With this facility an application doesn’t require being aware that service is executing in its own process or attached to a local activity process. This facility is presented as a feature of AIDL that will be described in following sections.
8.7.3 Interaction Model in Binder

Binder framework follows client–server model interaction. In client-server model client sends request to the server and wait for the response. The binder framework on client connects to the server through a proxy object. Once the request is received from client on the server-side, server assigns thread(s) from thread pool to handle the request and send back the response.

The communication model of the binder is shown in figure 31. Process A is the client that uses a proxy object, which perform interaction with kernel binder driver. Process B is assumed as the server with the Binder thread pool. The binder framework assigns threads to handle requests as long as the free threads are available in the thread pool. Proxy object as an interface interact with Binder driver to handle communication with the target object. Binder driver uses shared memory mechanism to pass messages between threads and process[37][54]. Shared memory mechanism shown in figure 32.
8.7.4 Transactions, marshaling, un-marshaling

Delivering a unit of data from one process to another is called transaction. The Payload of the data, which is submitted in this operation, named transaction data[37]. Basically in Android this transaction data is called parcel. Parcel is the container of data that also maintains some meta-data about its contents. Parcel can contain data and references to live IBinder object, which will be moved to the other side of IPC across processes. Parcel is not a general-purpose serialization mechanism and is designed as a high IPC transport.

Transmission data contains target of the destination node and ID of the sender as shown in figure 33. Also it includes serialized data array of commands that are passed through the Binder. In order to send an object over the IPC or network to the destination, it requires to be converted to the byte. The procedure of converting an object to the byte is called serialization. The data bytes on destination are converted to the object again, which is called deserialization. In Android all objects that requires to be converted to byte code must implement Parcelable interface. Therefore, they consider as serializable objects.

In Android, In procedure of sending an object over Binder, all information of the object breaks down into simpler data types are understandable to the system such string, float, Boolean. This data flattens into a parcel object and is converted to byte. The procedure of transforming an object to byte is called marshaling. Inverse; the procedure of reversing an object from the byte is called un-marshaling. The marshaling and un-marshaling is done by Android framework. The byte code data generated from marshaling is not appropriate to place into persistent storage because modification in the underlying implementation of any of data in the parcel can make the older data unreadable.

![Figure 33 Transmission data (Adopted from[37])](image)

8.8 AIDL

Android framework has its own implementation of Common Object Request Broker (CORBA) and Component Object Model (COM), which consider as a lightweight RPC mechanism[54]. Similar to other IDLs, It defines a programing interface, which provides a mechanism for client and service be able to communicate with each over IPC. This mechanism is called AIDL (Android Interface definition language)[61].
In Android, a process normally cannot access the memory of another process. The key purpose of the AIDL is to facilitate the implementation of Android remote services.

It allows developers to define an interface with method signatures of the remote services that the both client and server agree upon that in order to interact with each other through IPC. The interface file is in aidl format.

The AIDL follows java syntax. The AIDL parser generates automatically a java interface from the aidl file that itself consists of a proxy and stub class and is available for the client as well as the server. The proxy class facilitates the client access to the service. The stub is an inner abstract class that derived from and implements the method from the AIDL interface. The developer derives the Stub class and implements the methods in the remote service.

As the communication between the client and the server is over the Binder, the data of the object requires to be decomposed to simpler data type such as integer, string, array before marshaling in client and server side. The generated AIDL java interface handles decomposing data into primitive data types, flatten into the parcels , sending them through IPC by calling Binder transact method and rebuild the object on the client and server side as well as invoking the service method on the service and send back the result.

\textit{transact()} in Android method is the key of IBinder API stitched to \textit{Binder.onTransact()}. these two methods allow the application to invoke a call bound in a parcel to an IBinder object and receive a call coming to a binder , respectively. When an IPC invoke is performed from process A to process B , a specific thread in process A sends the transaction data to process B through the transact() method . Incoming transact is received by a thread from the thread pool in process B .The transaction calls Binder.onTransact() on the target object and send back the parcel result to process A. once the results is received by the thread in Process A, the execution continues. Transaction API is synchronous; meaning that a call to transact () return result as long as target has returned from Binder.onTransact().in other words, the specified thread in process A is blocked unless the result is received from process B[62].

The source code of stub and proxy both are generated in the same file located in gen folder, which is accessible, by remote service and client (e.g Activity) files.

8.9 AndroidManifest.xml file

Having AndroidManifest.xml is an obligation for every Android application. The manifest holds fundamental information about the application and some other essential information which system need to know before running the application. This information includes naming java packages, component of the application; activities, services, broadcast receiver, permission to interact with other applications or protected parts[63].

8.10 Android APK

Android applications use the specific file format to be able to be installed on Android operating system, which is called application package file or APK in abbreviation. The Android programs are package into apk file format .apk files are
type of zip format file with .apk extension, which includes the programs codes in dex files, resources, assets, certificates and manifest files.

8.11 Start/Stop and bind/unbind a Service in Android

In Android, start or stopping the service is performed by sending Intent to the target service. The intent can contain complete class name of the service or abstract definition through the action and other filed that requires to start/stop the service. bind/unbind a service is also performed by the Intent. When a service binds to an activity component it makes an activity can invoke the method of the service. Once a service is unbind the activity disconnect from the service and can not call the methods of the service any longer. The mechanism for start/stop, bind/unbind a service(s) is a little bit different than invoking the method of the service because they don’t work based on AIDL mechanism. Figure 34 depicts the sequence diagram of start/stop and bind/unbind an Android service from an activity. By executing each of startService()/stopService() or bindService()/unbindService() methods in activity, methods with the same names are performed in ContextImpl object and then ActivityManagerProxy object. The request sends to the destination through the transact method and the Binder. When the service is started/stopped, bind/unbind the result sends back to the activity in the same way as delivered. The stop/start, bind/unbind the service is performed in a new assigned thread. Therefore, the application won’t get blocked during the start/stop, bind/unbind the service.

![Figure 34 Start/Stop, bind/unbind the service](image-url)
9 APPENDIX II

9.1 Solution Implementation

In this chapter, we describe implementation details; include class diagram and modified or added methods, classes to the Android operating system to extend the functionality of the OS in order to be able to execute a service on a remote machine.

All development in the Android including code modifications and adding new codes to handle new functionality are done in Android 4.0.4. This version of Android OS checked out from the head of Android code repository when we started conducting this research on May 2012. The check out process is done based on the Android developer official website instruction in [64]. In the beginning of this chapter we start with describing the binder framework implementation in Android and then focus on describing the modifications that we have done in Android framework in order to execute the Android services remotely.

9.1.1 Development Environment

In this section we describe the development environment and tools that are used to develop the prototype.

In software development, having a proper working environment increases the performance of the development team. The hardware and software details for developing the proposed architecture prototype have been described in following.

Table 12. Depicts the Hardware and software that are used in this thesis. It includes Development machine and server machine specification, phone hardware specification, operating systems and also includes software tools used in development environment.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Development machine &amp; OS</th>
<th>Server &amp; OS</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>X86 machine with 8GB RAM, 16 dual core CPU 2.6M and Ubuntu 12.4 for OS</td>
<td>X86 machine with 4GB RAM, 4 dual core CPU</td>
<td>ST-Ericsson 8500</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software&amp;tools</th>
<th>Code Repository</th>
<th>Repo Git</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Intellij IDEA 11.00 &amp; Eclipse</td>
<td></td>
</tr>
<tr>
<td>Script tool</td>
<td>Apache Ant 1.8</td>
<td></td>
</tr>
<tr>
<td>Test tool</td>
<td>Android emulator</td>
<td></td>
</tr>
<tr>
<td>Developed OS</td>
<td>Android OS 4.04</td>
<td></td>
</tr>
</tbody>
</table>

Table 12 Development software and hardware

In the first step, the Android SDK was downloaded from Android site and installed according on instruction in[65].

The source code of the Android 4.04 downloaded to the Ubuntu development machine from Android code repository. Also development machine is prepared for compiling the Android source according the instruction in [66].
Intellij IDEA[67] & Eclipse[68] as the IDE are installed on Ubuntu development machine and is accessible through the terminal for client and server development. The IDE provided some facilities to expedite development and modification of Android OS.

On client Side, When the Android OS is compiled the output file copied to a mac machine to be executed by an Android emulator. The emulator is used for doing test on client side because it was faster in comparison with running the test on the phone.

On the server side, there is an customized version of Android x86 OS[38, p. 86] installed inside the oracle virtual Box[39] on a Linux Ubuntu 10.4. Once the Android started it took a static IP address automatically from the network.

The emulator and the Android x86 OS are communicating over socket on port 32000. For this purpose, the firewalls between the client and server machine are configured to allow the traffic on the port for the static IP address.

After the development and test of each feature, the code is committed to the Git code repository to keep all history and full revision tracking.

The final test for each feature is done on the phone. The phone is connected through Telia network to the server machine over the port 32000.

The phone is used its own customized Android OS which is developed by ST-Ericsson. The ST-Ericsson Android OS customized version includes some additional program, which leads it to work with the ST-Ericsson, phones e.g. modem driver. To apply the modified and new codes on ST-Ericsson Android source code and use our modified OS, we took a patch from newest commit in Git and apply the patches on the ST-Ericsson code. Normally, compiling the ST-Ericsson source code took 30 minutes. Finally, the compiled code pushed to the phone in order to be tested with the test application in connection to the server through the Network and configured port 32000. The development environment is shown in figure 35.

![Figure 35 Development environment](image-url)
9.1.2 Binder Framework implementation

This sub-section, presents an overview regarding the implementation of binder framework and the purposes of each layer. The Binder framework implemented in three layers in Android operating system. Each layer does the specific task. Figure 36. Illustrates all three layers of binder framework implemented methods.

![Figure 36: IPC Binder System](image)

The first and top layer provides API that user can incorporate them into their applications. The middleware layer provides different services and that are executing quickly cause the included libraries are compiled to machine language. The third and last layer is the kernel driver. In following we elaborate about mission and functionalities of each layer in IPC.

- **Java API wrapper**: Java API wrapper is the first layer of the Binder IPC system. This layer is implemented which responsible to provide the wrapper APIs for the functions in middleware layer. These wrapper APIs allow the Android application can use Binder communication. Also it provides facilities to the Binder framework to use intent. The java classes, interfaces and dependencies of Java API wrapper layer are shown in figure 37.
Middleware layer: The Java API layer that provides APIs for apps that use middle layer. The Java APIs on the upper layer mapped to the implemented C++ functions in middle layer. Middle layer is developed in Java Native interface programing (JNI). The JNI provides a facility for java APIs to invoke native function and libraries, which developed in C++ [37]. The frameworks/base/core/jni/android_util_Binder.cpp file includes the map of java layer functionalities implemented in C++ such as transact(), marshaling and unmarshaling. Moreover, Thread management and handling threads and controlling process are done in this layer. Also the implementation of marshaling and unmarshaling the parcel and interaction with the Binder Kernel is done in this layer.

The implemented C++, cpp and header files of middleware layer our mainly located under below paths:

“frameworks/base/include/utils”
“frameworks/base/libs/utils/”

Kernel Driver: Binder kernel driver is the key part of binder framework, which
guarantees the secure and reliable interaction between the processes in Android. It’s developed in C and includes some simple functionalities establish a connection with binder, map binder memory and provide the facility for upper layers to submit and receive information and messages[37].

In order to extend the functionality of the Android framework to be able to execute and handle the communication with the service on the remote machine, we modified the files and added new functionalities to the Java APIs wrapper layer as well as middleware layer.

9.2 Android framework Modifications

This section describes the modifications have been done in Android framework to allow it be able to handle communication with the services on remote machine includes start/stop,bind/unbind service as well as remote method call.

9.2.1 Package Manager

In Android, all information regarding the application packages, which are currently, is installed on the device including all data defined in the AndroidManifest.xml are available and accessible via PackageManager class. This class is accessible through getPackageManager().

We used getPackageManager() to retrieve the data defined in AndroidManifest.xml .This data is used to identify between the local and remote service in modified version of Android framework .The identification is done based on the Android local service tag convention and our remote service tag convention which defined for this purpose.

9.2.2 ContextImpl

It is a Common implementation of Context API, which provides the base context object for Activity and other application components. The ContextImpl is used not just for each internal process communication through the Binder but also for installed application on device. extractMetaData method in ContextImpl is responsible for extracting the data of current running Android application includes all data of services in AndroidManifest.xml and populate the MetaData class through the PackageManager when the service is method invoked or started. This MetaData is checked in our implemented transact() to check whether the invoked method is in local or remote service.

The class file of ContextImpl located in below path:

/framework/base/core/java/android/app/ContextImpl.class

9.2.3 SingletonMetaData

SingletonMetaData class is implemented by us to store the current data of the application regarding the services configuration. SingletonMetaData is a singleton class with getter and setter method which is initiated and populated in ContextImpl by calling getPackageManager() and accessible in our implemented transact(). The SingletonMetaData is developed as a static class to reduce the class initiation overhead and better performance. The SingletonMetaData is called in our implemented
transact() to check whether the invoked service is remote or local. For the local service the original transact method (transactOrigin()) is called to handle the request over the regular kernel Binder. For remote service, the ClientSocket class is initiated and populated with parcel through the setter method to send the request over the network to for executing the remote method and then receive the response.

ClientSocket class file is created in below path:

/framework/base/core/java/android/app/SingeltonMetaData.class

9.2.4 Transact

Transact method in Android OS Binder class is a native method that is implemented in middleware layer. We implemented our transact method which its definition is same as the original one. The motivation behind implementation of our transact() is identify remote method request invocation from local invocation coming to transact and also instantiate, populate ClientSocket class and the request to remote server as long as request is related to remote method invocation. we modified the name of the original transact() to originTransact(). The originTransact() is called in our implemented transact method as long as the request for invoking the method belongs to local service.

Binder.java file is located in below path:

/framework/base/core/java/android/os/Binder.class

9.2.5 ClientSocket

ClientSocket class is responsible for establishing connection with remote server through TCP/IP socket and marshaling/un-marshaling the request/response data. The ClientSocket is instantiated and populated by our implemented transact when the invocation is for the method of a remote service. It converts, the input parcel to byte code with the marshal method in the parcel. sentBytes() in ClientSocket submits the data over the available socket connection with using DataOutputStream[69] java API.

readBytes() is used to read byte result from socket which coming from the server with DataStreamInput[70] java API. Un-marshall method from parcel is called to convert received byte data result into parcel object.

To avoid thread interference and memory consistency errors, all marshaling/unmarshaling,readBytes and sendBytes method are done in a synchronized block.

Moreover, Client socket is executed in new separated thread in transact() because from Android OS 2.2 it is not possible to perform a network operation on main thread of Android framework[71].

ClientSocket java file is located in below path:

/framework/base/core/java/android/os/ClientSocket.class
9.2.6 ActivityManagerNative

In Android, starting, stopping and binding/unbinding a service is done through ContextImpl. startService method in ContextImpl calls startService method in ActivityManager class to form a parcel and invoke transact method to start the service through the Binder. In the modified Android framework, startRemoteService() implemented with the same description and similar content for starting the remote service on different machines. Just an integer is added in the parcel which defines the parcel as a start remote service command. The constant definition of remote service transaction defined in IActivityManager also same constants are added in the server side program to detect and identify start, stop, bind and unbind requests. Moreover, the part of code which was related to flatten binder in the parcel (writeStrongBinder) is removed in startRemoteService() because it’s not possible to marshal parcel with binder in Android.

The stopRemoteService() and bindRemoteService(), unbindRemoteService() implemented almost same as startRemoteService() which handles stop and bind and unbind the remote service in addition different constant description for stop, bind and unbind commands.

Here is the path of ActivityManager:
/framework/base/core/java/android/app/ActivityManagerNative.class

9.2.7 IActivityManager

IActivityManager is an interface that is implemented by ActivityManagerNative. For this reason definition of all new methods: startRemoteService(), stopRemoteService(), bindRemoteService() and unbindRemoteService() are expressed in IActivityManager. Additionally, integer constants definition that are used in above method are expressed in this interface. Therefore in modified Android we added the definition of the new implemented methods in IActivityManager.

The IActivityManager is located in below path:
/framework/base/core/java/android/app/IActivityManager.class

9.2.8 ActivityManagerService

ActivityManagerService class is derived from ActivityManagerNative. Hence all startRemoteService(), stopRemoteService(), bindRemoteService() and unbindRemoteService() are implemented here but with same content of startService, stopService(), bindservice() and unbindService() of as are in ActivityManagerService in the modified framework.

The ActivityManagerService class is located in below path:
/frameworks/base/services/java/com/android/server/am/ActivityManagerService.class
9.2.9 Android_util_Binder

As discussed in earlier sections, The Java APIs on the Java APIs wrapper layer mapped to the implemented C++ functions in middle layer. In Android OS, All implementation of native methods in Binder class are all mapped to the android_util_biner.cpp.

As discussed earlierdections, the transactOrigin(),new implementations of transact method developed in Binder to handle all of our new requirements regarding the executing Android services remotely. The transactOrigin() is still required for the regular Android IPC. The transactOrigin() is a native method and mapped to android_util_binder.cpp in middleware layer so, In order to handle current transaction functionality as well as the new functionality we changed the name of implemented transact method in android_util_binder.cpp modified to transactOrigin.

The android_util_Binder C++ file is located in below path:

/framework/base/core/jni/android_util_Binder.cpp

9.2.10 IBinder

IBinder is the key interface for interacting with a remotable object. It includes the definition of abstract methods that allow the remote procedure call in Android. Transact() is the key API in IBinder which allow a synchronuos call in IPC mechanism[62].

IBinder implemented in Binder class hence all the implemented methods in Binder class including transact() and transactOrigin() must be defined in IBinder.

The IBinder class path stated as following:

/framework/base/android/os/IBinder.class

9.2.11 AndroidManifest

As described in earlier sections androidManifest.xml keeps all configuration of the Android application which are used at the runtime. Based on [72] Android OS allows developers to add app configuration information to their application through define meta-data tag. Execute Android services on a remote machine is a new functionality that requires to be defined in a way that is identifiable from regular local services in our modified Android. To achieve this we defined the remote services definitions in androidManifest.xml with the specific meta data tag convention which is identifiable by transact function in our implemented transact() in binder class. This meta data tag contains two attributes; android:name and android:value. android:name indicates the service name which should follow a specific syntax. The first part of the name is always starts with remote_service_monexmo and the rest is a number which indicates the service number e.g remote_service_monexmo_1. The service number amount is need to be increased for executing each available services remotely. android:value keeps the name of auto generated AIDL java file related to the service. This meta-data tag needs to be added in between application tag. An example of the meta-data tag is shown in figure 38.

<meta-data android:name="remote_service_monexmo_1"
android:value="com.ericsson.monexmo.test.ItestService"/>
I the modified Android framework, an application with the remote service require establishing a connection to server side program in order to invoke the remote service. For this reason the developer must add *uses-permission* tag which allows the application can connect to Internet. This tag is shown as following:

```
<uses-permission android:name="android.permissionINTERNET" />
```

### 9.2.12 Modified Android OS

As discussed earlier the most modification to handle the execution of Android service on a remote machine are done on the Java API wrapper of the Android framework. The class diagram of modified Android OS illustrated in figure 39.
9.3 Server side

As mentioned in earlier sections server side contains a Broadcast receiver class, which is responsible for communication with client or the phone, and also identify between the requests received from the client to call the appropriate service through the proper service IBinder.

The configuration of the services is available in androidManifest.xml in the server side. The challenge here is to load the identified services from the available service classes, which is not installed on the server side. In order to make the service object from the service class, we used the reflection Class Loader mechanism. In Android DexClassLoader[73] API is used as the class loader. In Broadcast receiver DexClassLoader makes the service object instance from the specific service class.
Basically the service object is loaded in Dalvik virtual machine. Subsequently the onTransact() from the stub of the target service invoke the target method from the service instance object. The result again get marshaled and is submitted to the client by broadcast receiver. The mechanism for un-marshaling receiving and marshaling, sending is same as the client side that discussed earlier. Each service executed in a separated thread not in the main thread of Broadcast receiver class.