University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

BACHELOR THESIS

Offloading in a mobile environment using Bluetooth Low Energy

Scientific Adviser:
S.l. dr. ing. Laura Gheorghe

Author:
Antonel-George Dobre

Bucharest, 2015

Maecenas elementum venenatis dui, sit amet vehicula ipsum molestie vitae. Sed porttitor urna vel ipsum tincidunt venenatis. Aenean adipiscing porttitor nibh a ultricies. Curabitur vehicula semper lacus a rutrum.

Quisque ac feugiat libero. Fusce dui tortor, luctus a convallis sed, lacinia sed ligula. Integer arcu metus, lacinia vitae posuere ut, tempor ut ante.

Abstract

Here goes the abstract about MySuperProject. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean aliquam lectus vel orci malesuada accumsan. Sed lacinia egestas tortor, eget tristique dolor congue sit amet. Curabitur ut nisl a nisi consequat mollis sit amet quis nisl. Vestibulum hendrerit velit at odio sodales pretium. Nam quis tortor sed ante varius sodales. Etiam lacus arcu, placerat sed laoreet a, facilisis sed nunc. Nam gravida fringilla ligula, eu congue lorem feugiat eu.

Contents

List of Figures

List of Tables

Notations and Abbreviations

BLE – Bluetooth Low Energy
BLEOffloadingFramework – Bluetooth Low Energy Offloading Framework

Chapter 1

Introduction

In past years, mobile devices have encountered a widespread use among technical and regular consumers world wide. This increase in popularity is based on the advent of the Internet and social media, together with easy to use, user-friendly interfaces and applications for hand held devices worldwide. A mobile device is not defined anymore as a strict communication device, but as a hand held computer that acts as an access point to content and information sharing.

Although this growth spark of recent years has lead to over the top technological advances, mobile devices have their limits which is mostly reflected in their size and battery life. The key for a successful device is to provide a user-friendly interface with a rich set of features, ranging from on-the-go connectivity to playing media. The main issue that arises in such a rich environment is the constraints on battery life and the fact that producers need to maintain a balance between usability and efficiency. Most smart phones today run on a typical battery of 1500 mAh [5], mainly because this is a limitation in size. Unlike smart phone technology that has developed drastically in the past few years, battery technology has been evolving for the past century and such no large breakthrough has been discovered in the last 15 years[6].

Developers for and of the smart phone platforms have since realized that they need to bypass the hardware constraints and create either power efficient chipsets and components or create efficient software that provides the selected features that are in demand. A method that can achieve a more efficient energy consumption on smart phone devices would be a mixed approach of power efficient hardware and the ability to communicate with other devices and software, called code offloading.

Computational offloading is a technique used to share the processing power of several devices between each other in order to achieve better performance. Code offloading is a technique that has gained a lot of interest recently due to the possibility of using the *System as a Service* architecture of cloud computing in order to offload resource intensive operations to cloud-based surrogates[12], especially in the case of low-power devices such as smartphones. This technique usually occurs at the code level, where a mobile application may be partitioned such that some of the more process-intensive tasks or algorithms would be run on separate machines. The partitioning can be done either by the developer, statically, or it can be determined at runtime, dynamically, by a simple linear algorithm that measures the cost of data transfer and the potential offloading gain that can be achieved for certain tasks. Because the cost might outweigh the benefit gained, offloading should be considered an optional process preferred in mobile operations that require high amounts of processing time and low amounts of data transferred between devices[9]. Previous works have also identified that computational offloading is required mainly by applications that implement graphical rendering, image and video processing techniques[2] [1].

TODO:
task delegation

Following the notion of computational offloading and the need for extended battery life, this paper proposes a system through which the process of code offloading and task delegation can be done efficiently over a low-energy imprint communication channel: Bluetooth Low Energy.

Bluetooth (IEEE 802.15.1) is a technology based on a wireless radio system designed for short-range and low-cost devices in order to replace cables for computer peripherals, such as mice, keyboards, printers, etc. Since it's conception, this standard has seen a wide variety of use and has evolved from it's main purpose of interconnecting peripherals to creating small, wireless, personal area networks (WPAN) that permit advance data transfer such as: file sharing, transmitting TCP/IP packets, data streaming over simulated Serial Ports and other uses.

Together with Bluetooth 4.0 specification a new design was proposed for low-energy devices, which represents a trade-off between energy consumption, latency and throughput. This new specification has been dubbed Bluetooth Low Energy and since 2010 it has been implemented in most hand held devices along side Bluetooth, in a setup called dual-mode.

1.1 Project Description

One of the important factors when discussing computational offloading is the communication channel between devices. Most work in this field has been in done with the objective of using the powerful computational model of Cloud Computing[9] in order to establish an offloading system for mobile devices that transfers data and code through Internet connectivity, either through Wi-fi or 3GR wireless technologies. Usually this type of system implies that offloading is possible only when the right connectivity makes possible a certain gain of computational power, which is not always the case.

In order to mitigate the drawbacks of the aforementioned model, this paper proposes a new framework that uses the advantages of Bluetooth Low Energy in order to lower the cost of transferring the data between devices, thus maximizing the gain that a system can achieve when offloading.

As such, we present the BLEOffloadingFramework (Bluetooth Low Energy Offloading Framework) , a framework designed for developers of mobile applications that uses low energy communication channels in order to transfer data and code between hand held devices, such as smartphones or tablets, with the specific objective of saving battery life of the desired embedded system.

One of the most energy consuming part in mobile embedded systems is the main Processing Unit. A correlation exists between the battery life of smartphones and the amount of time the CPU is doing work [5], especially in modern day devices, where processors have a high potential for computational power, but are not especially power efficient. Smartphone operating systems such as Android have additional protections against unnecessary CPU usage such as the Power Wake Lock, in which it creates a link between the screen of the device and the processor and forces the system to go into a sleep mode when the screen is turned off, thus preserving battery [3].

As such, one of the motivation behind offloading and the project presented in this paper would be **creating an efficient and easy way to offload computational tasks in order to preserve battery life, with the key objective of extending user experience**.

For this purpose, the BLEOffloadingFramework uses the low-energy specification of Bluetooth in order to transfer data between devices. This technology has been proved to be more efficient

in transferring small chunks of data between devices [10] and helping the connection setup of the standard BR/EDR Bluetooth.

The framework follows a typical offloading system architecture, but instead of relying on Internet connectivity of the device it uses a wireless personal area network established through Bluetooth Low Energy. In a general offloading scenario (such as the one described in 1.1). When the right conditions are met the compute part of an algorithm can be transferred to the cloud, sending back a result of said computation that can be presented to the user or used in other parts of the application.

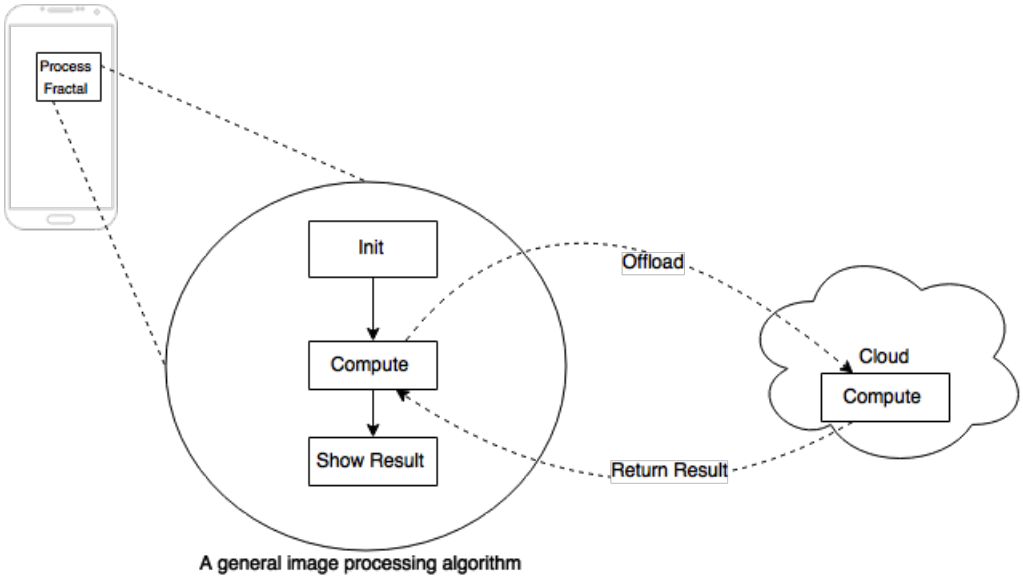

Figure 1.1: A general view on offloading

The BLEOffloadingFramework takes this concept and applies it to a much smaller range network. An example of how the framework works is depicted in appendix A.

As you can see from the appendix, the framework consists of an Offloading Server, that can be any Bluetooth enabled device, such as a Personal Computer, a laptop or even an embedded device, such as a Raspberry PI [1]. This server always runs a low-energy advertising technique (described in section 2.1) in which it promotes its existence to other devices.

If a device picks up on such a server and wants to perform offloading it will create and send a request to the server with the desired method that it wants to offload. The server then decides if it can process that request or not. It will send back to the device a Accept/Reject type of answer, in which Accept states that it can begin the offloading and Reject means that it is too busy at the moment.

Once the request is accepted, the server will compute the selected method and offer a response back to the application.

Experimental results demonstrate that this typical scenario presents an increase in battery life, because the mobile device does relatively low processing on itself.

Because of its lightweight infrastructure and use of low-energy technology, the BLEOffloadingFramework presents a series of advantages over other offloading scenarios. In this paper, the method and implementation of this framework will be presented, together with the context and

[1] Raspberry PI - a small ARM-based System on a Chip embedded device that can run a lightweight Linux distribution, used mainly for education and small embedded projects

motivation behind this type of offloading and also the experimental setup used to demonstrate the principles of the framework.

Chapter 2

Background and Related Work

This chapter presents the technical background of the technologies used in constructing the BLEOffloadingFramework. As previously stated, Bluetooth wireless technology is used as a communication channel between offloading devices, as such the key terms of this standard are presented in section 2.1.

The main consideration of the offloading framework is extending the battery life of embedded devices, with a special focus on smartphone mobile devices. In section 2.2 the Android smartphone operating system is presented and the motivation for choosing this type of devices as the beneficiary for the framework.

In section 2.3 several offloading systems are presented and the main advantages of BLEOffloadingFramework over other systems is detailed.

2.1 Bluetooth Low Energy

Bluetooth has been a long standing standard for small area wireless communications. Most mobile devices, ranging from PDAs to mobile phones and other gadgets, use this technology in order to communicate effortlessly over short distances, making possible file transfers, contact sharing, wireless audio and video streaming and much more. With the progress of Internet and the Cloud, though, the need for small Personal Area Networks has been reduced, as its drawbacks became more and more obvious - battery life of mobile devices has been reduced and the added overhead of Bluetooth communications is not sustainable, has a low throughput and a small range.

Together with the specification of Bluetooth 4.0, Bluetooth SIG has also announced the standard for Bluetooth Low Energy[7]. This standard focuses on a trade-off between energy consumption, latency, piconet size and throughput. The advent of this standard, versus other similar wireless solutions such as ZigBee, is due to the fact that it is applicable in a larger variety of use cases: healthcare devices, small electronics, low power devices, Internet of Things or security measures.

This standard also offers full backwards compatibility, as the added benefit of low-energy transmissions can be used in parallel with the normal Bluetooth 4.0 specification. This is applicable because BLE mainly relies on parameter configuration and short, but consistent, device discovery.

In classic BT applications, when two devices needed to communicate they had to be set in Discoverable mode, identify each other and create a secure connection in a process referred to as pairing and then follow the specifications of certain Profiles. We can compare this wireless

connection capability to the OSI stack, where instead of protocols, we have profiles that specify how to interact with different devices. For example, in order to connect to a Bluetooth enabled Mouse or keyboard and use its facilities the device needs to follow the guidelines of the Human Interface Device (HID) profile.

TODO:
insert BLE stack picture here.

Just as in classic Bluetooth[8], the BLE protocol stack consists of two main parts: The Host and the Controller. The Controller contains the Physical Layer and the Link Layer and is integrated on a small System on A Chip with radio capabilities. The Host part of BLE usually runs on an application processor (such as the processor in a mobile phone) and contains upper layers of functionality such as:

- Logical Link Control and Adaptation Protocol (L2CAP) - this layer is an optimized and simplified protocol based on the Bluetooth L2CAP, with the main goal of multiplexing data of other higher layer protocols.
- Attribute Protocol (ATT)- this protocol defines the communication between two devices playing the role of client and server. The server maintains a set of attributes, where an attribute is a data structure with information managed by GATT. The client can access the server's attributes by sending requests, while the server responds with response messages. These type of messages follow a stop-and-wait scheme.
- Generic Attribute Profile (GATT) - this profile defines a framework that uses the ATT for the discovery of services and the exchange of characteristics between devices. A characteristic can be defined by a set of items with values and properties.
- Security Manager Protocol (SMP) - BLE offers various security services for protecting information exchanged between devices, but most of the supported security services can be categorized in LE Security Mode 1 and LE Security Mode 2, which provide security at the Link Layer and ATT layer, respectively.
- Generic Access Profile (GAP) - This profile specifies device roles, modes and procedures for the discovery of devices and services, the management of connection establishment and security.

It is worth mentioning that the GAP profile permits several operating modes, through which several techniques can be established. One such technique is called Advertising. In this technique, a device assumes the role of GAP Broadcaster, which sends small packets of data constantly. These packets contain a string of bytes that are used for identification of the device, the service used and also vendor-specific bytes.

Devices can act as Observers in BLE and such pick up on notifications and packets transmitted by a device in Broadcaster role. This technique is called scanning, and most devices can efficiently scan for advertisement packets by applying filters at the Link Layer and such only receive notifications if there is a specific Broadcaster device in range.

The BLEOffloadingFramework uses this technique in order to identify servers. If a mobile device desires to offload, it will start a scan over Bluetooth Low Energy and waits to see if it picks up any packets from a server. If such a packet exists, then the offloading framework on the client side becomes active and when an application wants to use this system a request will be generated.

When the mobile devices exceeds the server range, it will loose contact with the server (no more advertising packets detected) and such it will shutdown the framework until a new server comes in range.

Even though the main drawback would be the small range of BLE, which is around 10 meters for most devices and is dependent on the hardware, using this model of server detection and data transfer represents a tradeoff between the latency caused by a network connection to a distant cloud server in other offloading systems and the availability of such systems.

2.2 Android

Android is an operating system designed for smartphones, with a focus on usability, touch input and efficiency, both in power and computational abilities. It was first defined as a "software stack for mobile devices that includes an operating system, middleware and key applications" [4]. Today, Android exists on numerous devices, including smart watches, TVs and even cars [13].

This operating system is designed on top of a modified Linux Kernel[1] with a specific stack designed with user applications on the very top. This model permits the enabling of security protocols on the lower levels (closer to the hardware), while providing a feature rich environment for third party developers that deliver content to this specific ecosystem.

Because of the availability of this code, the plethora of devices it runs on an the simple interface that it provides, the BLEOffloadingFramework is constructed mainly for the Android Operating System.

TODO:
NOTE ON FIRST DRAFT: should I add more to this part, such as the Android framework and SDK?

2.3 Related Work

This article is based primary on the works of [11] in which an offloading mechanism based on the application life cycle is proposed.

In this model, an application has several states such as interaction with an user via the GUI, processing multiplayer commands, simulations, graphic pipe rendering or terrain generation and some of them are done cyclically by the application. The research is based on the fact that certain states of the above loop can be offloaded completely on other devices or on servers in the cloud. One such application that fits this pattern is OpenTTD.

An example regarding OpenTTD is the offloading of the Artificial Intelligence agents that act as players throughout a game. These agents give out commands and take decisions like a real player and are a core part of the application infrastructure, that use up a lot of processing power, depending on the complexity of the algorithms used. One way to improve on this technique is to search for other states that can be offloaded, besides the agent scripts, or to apply a fine-grain distributed technique.

As an example we can either use the same technique in the GenerateWorld state, which is an initialization state. We can send the settings used to generate over a network communication to a cloud service and generate the world there, the result being a large amount of already processed data that the application can use. While for small worlds this method might bring a very small improvement, or none at all, for large maps can benefit from this technique. By applying fine-grain distributed technique we can mark the methods or parts of code that can be offloaded and offload them. This means that we have to create an abstraction that encapsulates

[1] Linux Kernel - http://www.kernel.org

the code, which is usually process-intensive, and offload that segment to a server that knows how to interpret it and simulate results. This technique resembles the Java Remote Method Invocation or Remote Procedure Calls.

The framework presented in this paper proposes a more generic approach to offloading, in that the focus will be more on optimizing the data path between devices together with the capability of applying this framework on almost any type of application.

In [1] a more general offloading solution is proposed. In this method, using the high performance and mobility of cloud technology a virtual machine is created that simulates the exact environment of an application from a mobile device. With such a medium, code translating from mobile device to another machine is straightforward and less error prone. The only drawback that may occur is the fact that data usually has to be transferred across multiple points in order to be processed and that the mobile device has to always be online.

The BLEOffloadingFramework will try to handle these problems by reducing the amount of time spent on data communication, while maintaining the generic aspect of the code that can be offloaded.

Chapter 3

General Architecture

This section presents the main guidelines of the architecture used in developing the BLEOffloadingFramework and the two main offloading solutions. The first topic of discussion is the environment in which the framework is run, followed by the presentation of the techniques used and finally a general description of the data flow.

3.0.1 Offload Environment

When talking about offloading in a mobile context, we mainly refer to transferring code and data from hand held devices. These devices have evolved in recent years from the mobile phones used to communicate between two or more people to personal computers that permit you to connect to the Internet, socialize, play videos, listen to songs and much more. On the current market, the lead operating systems for smart phones are Android, open source project maintained and controlled by Google, and iOS, proprietary software owned by Apple.

Because of its open source nature and the fact that it is present on different devices with different specifications, the BLEOffloadingFramework was built with Android as the primary client for this architecture. Although, the generic approach devised here accounts for any smart phone that has the capability of Bluetooth Low Energy technology.

Android is an operating system based on the Linux kernel [1] with enhancements regarding embedded systems, such as a well defined sleep/idle time, better power management, paranoid security and a well rounded framework that acts as a layer between underlying hardware and software applications. The Android Framework is mainly written in Java and has a client-server approach to inter process communications and event managing.

This approach permits applications developed by third parties to be created and run inside their own virtual machine and whenever they need a system functionality, such as access to the camera hardware or to other applications a request needs to be sent through the framework. An overview of Android and its functionality can be viewed at [**?**].

As such, the environment that the offloading framework proposed in this paper is defined as the Android mobile applications and one of the main objectives of the project is to help developers in creating power saving applications using this system.

[1] www.kernel.org

3.0.2 Offloading Solutions

After defining the objective and the clients of this framework, it is time to define the general offloading methods through which the system achieves its purpose.

1. *Remote Procedure Call/Remote Method Invocation*

   RPC[?] is not a new protocol. With the advent of the Internet, which has become the largest distributed system ever built, several methods of accessing code across different platforms have been described for different uses: from database interrogation across networks to accessing the output of proprietary code in a secure way. The basic definition of RPC is that it's similar to local procedure call in the way that when a section of code desires to call a procedure it would need an address to jump to. In Remote Procedure Call, that address can be on another machine altogether and the system can either wait for the procedure to finish, or, in more complex systems, continue it's execution and receive an event when the call is done.

   For offloading purposes, RPC or RMI (the equivalent in object oriented languages) means that the device can effectively use a system that has been implemented and stabilized for some time now in order to execute very processor intensive code. This would be the main advantage regarding this offload technique.

   The main disadvantage would be that it is not a very scalable solution. A developer that uses Remote Procedure Calls would have to plan its application in advance and select the exact methods or procedure that he wishes to offload. This sometimes results in duplicate code, as the framework would not be available some times, and would be very hard to maintain.

2. *Loose coupling of systems*

   If the previous method relies solely on implementation, this method requires structuring the application in such a way that certain parts are independent from the rest of the system, as shown in figure **??**.

   A good example is an application that processes images in order to obtain certain information from it, such as faces or objects. If we are to take into consideration hand held devices without a specialized Graphical Processing Unit, then this algorithm is costly on the CPU. In these cases, it would be much more efficient to send the picture to another device and use its processing power or even GPU.

   The drawback of this method is that not all applications are suited for this type of design (such an application that rely solely on device hardware), while the advantage is that it offers a scalable approach to offloading.

3.0.3 Data flow and main agents

In a general Offloading Framework three components can be correctly identified: the device that needs to offload its work, the communication channel and the the device that helps boost performance and receives the work to be offloaded.

The system proposed in this paper uses for communication a low power consumption implementation of the Bluetooth technology, called Bluetooth Low Energy [?]. This technology permits a high throughput of data to be transferred wireless with relatively low energy consumption over short distances (below 10 meters). The benefit the BLE communication brings is that the channel used has a very low impact on the system itself [?], but it has an innate drawback in the fact that it has a short range.

In order to mitigate this drawback, in this framework the device that provides the processing power is a small System on a Chip device that can either lend its power to mobile devices or act a gateway to other more powerful systems. The benefits for adding an intermediary for offloading solutions is that it keeps the simple goal of preserving battery power on the mobile device. The SoC is a small computer, without a battery that can be positioned almost anywhere: in a home, public transportation stations or work environments. By offering the possibility to access different processing power nodes would significantly reduce the mobile devices battery consumption.

Moreover, the implementation of the system permits interchangeability between the client and server devices. As such, if a device permits it, it can act as a server for another device, and thus managing to share some of the resources. The model proposed here will assume that the client is a mobile device based on the Android OS, while the server is a SoC, specifically a RaspberryPI (An ARM GNU/Linux box for $25).

In a first stage, when a piece of code that can be offloaded is detected, the client device will scan for potential offloading nodes. If such a node is detected, it will try to connect and create a network between the two devices. Once the connection is established, the client will send it's offloading request, through either of the aforementioned methods. The server will respond with either request acknowledge or busy.

The server can accept a number of connections and requests per time frame, in order to not burden itself and thus becoming a bottle neck for the system as a whole. Once the request has been acknowledged and processed, the results will be sent back to the mobile device, depending on the offload method that is used. An exemplification of this data flow can be view in figure ??.

Chapter 4

Project Implementation

In order to achieve the desired performance and low energy consumption, an offloading software framework is proposed. The BLE Offloading Framework is structured as a client-server architecture that uses Bluetooth Low Energy technology in order to communicate.

4.0.4 The client - Android Framework

The BLEOffloadingFramework offers an Application Programmable Interface (API) for developers of applications on Android devices. Using the Bluetooth Low Energy framework available since Android version 4.2 these devices can scan and connect to other devices, without an impact on performance or power consumption.

At the current state of the project the framework represents a test application that connects through BLE to the server side program and sends small packets of data that mimics data transfer over a period of time.

4.0.5 The server - Linux Embedded System

Because of the easy to use interface and availability of source code, the server is conceptualized on a Linux Operating System and uses the BlueZ[**?**] open source Bluetooth stack. For connectivity, a Bluetooth 4.0 USB dongle is used. This permits a generality for the system in the sense that it is not hardware specific - any Bluetooth chipset that abides to the standard can be used, even if it is directly embedded on the system, communicating through the UART interface, or through the USB protocol.

In order to facilitate development, the server is written in the C language. The basic server functionality is handling Bluetooth connections and responding in an efficient way to requests from clients as decided in the protocol mention in section 3.0.3.

The server starts off by advertising its availability using BLE Advertising[**?**] in connectable mode. This permits clients to automatically connect to the server and create an L2CAP socket that becomes available for use in transmitting and receiving data. After a connection is established, the server application waits a predefined period of time for clients to send a request header, that contains the type of offloading and data type that the client expects to receive after the invocation of the methods.

Chapter 5

Experimental Results

In order to validate the offloading system, a series of performance and stress tests are the determining factor. In this chapter, the testing methodologies are described for the BLEOffloadFramework.

5.0.6 Test setup

In order to offer conclusive data, the test setup contains different types of mobile devices that can benefit from the Offloading Framework, in this case, two smart phone devices from different producers with different specifications. Both devices are using the Android Operating System, version 4.4, in order to benefit from the Bluetooth Low Energy technology.

Both devices are charged to maximum capacity, as indicated by the Android notification system and run the same applications. Example applications include simple programs that are computational intensive, such as image processing applications or route calculating algorithms, which are common algorithms among mobile devices.

The idea is to expose the device to a series of tests, conducted using UIAutomator, a testing tool for Android that emulates user behavior. The test sequence repeats a pattern of user touch inputs until the devices receives a low battery notification, after which the time it took for the device to deplete it's battery is measured as the time between the start of the UIAutomator test case and the low power event.

Several test cases are distinguishable: The case where offloading is disabled and when offloading is enabled using different methods. The results of these test cases will roughly predict the power consumption of the devices in real life scenarios and the initial data can be extrapolated in order to predict the overall gain of using the offloading system.

These tests should reveal that using the framework described in section ?? will have an impact on energy consumption, in the sense that it takes a longer time to reach the low-battery notification. Exact empirical results will be available once the API for Android applications is completed.

Chapter 6

Conclusion and Future Development

Consuming less energy has been a prime focus in mobile communities since the development of high end smart phone devices has exceeded the technological advancement of batteries.

Offloading computational intensive code from one device to another is a method through which power saving can be achieved and the BLEOffloadingFramework is one of many solutions available to handle this problem. This system offers a complete solution for Android application developers to bring new value to their programs and to receive more out of the mobile environment. The solutions presented here are scalable, designed to be easy to use and offer a new dimension when it comes to programming for the Internet Of Things.

Improvements to the system can also bring a new form of distributed performance boost, because, even though the case for a stable server and mobile client was discussed, the framework can be made to work with other mobile devices as well, permitting users to share their power between them in a seamlessly and easy way.

Appendix A

BLEOffloadingFramework General Use Case

1. Advertising phase: An offloading server uses the technique known as BLE Advertising in order to promote it's whereabouts to nearby devices.
2. Enabling phase: A mobile device determines there is a server nearby through BLE Scan Filtering on a specific set of UUID. If an application desires to offload a certain part of its code to the server, it will generate a request to that server, which contains the offloading task, either pieces of code or specific task delegation.
3. Request phase: Once the server and mobile device know of each other and the mobile device generates a request, the server can respond with either Accept or Reject, stating that offloading is either possible or, respectively, not possible. If offloading is possible, the mobile device will wait for the results of the selected task.
4. Response phase: If the offloading was possible and successful, then the server will send back a response to the device containing the result of said computation

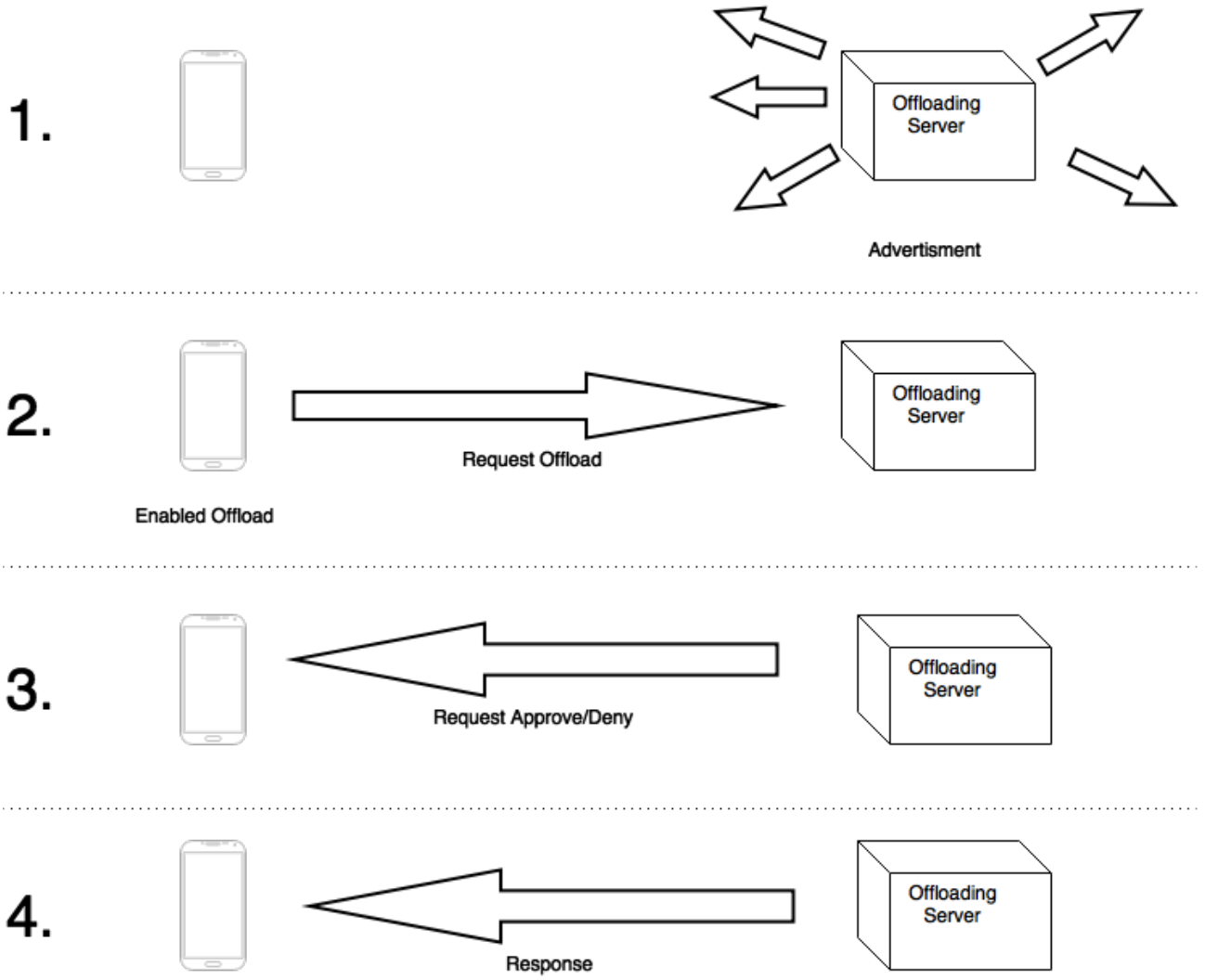

Figure A.1: Offloading Framework

Bibliography

[1] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[2] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[3] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. Android power management: Current and future trends. In *Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on*, pages 48–53. IEEE, 2012.

[4] Android Developers. What is android, 2011.

[5] Denzil Ferreira, Anind K Dey, and Vassilis Kostakos. Understanding human-smartphone concerns: a study of battery life. In *Pervasive computing*, pages 19–33. Springer, 2011.

[6] Megan Geuss. Why your smartphone battery sucks, 2015.

[7] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[8] Jaap C Haartsen. The bluetooth radio system. *Personal Communications, IEEE*, 7(1):28–36, 2000.

[9] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[10] Elke Mackensen, Matthias Lai, and Thomas M Wendt. Performance analysis of an bluetooth low energy sensor system. In *Wireless Systems (IDAACS-SWS), 2012 IEEE 1st International Symposium on*, pages 62–66. IEEE, 2012.

[11] Alexandru-Corneliu Olteanu, Nicolae Tapus, and Alexandru Iosup. Extending the capabilities of mobile devices for online social applications through cloud offloading. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 160–163. IEEE, 2013.

[12] Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3):1294–1313, 2013.

[13] Wikipedia. Android (operating system), 2015.

Index