Code Offloading in Opportunistic Computing

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Alan Ferrari

under the supervision of
Prof. Luca Maria Gambardella and Prof. Silvia Giordano

November 2017

Dissertation Committee

Prof. Mario Gerla	University of California, Los Angeles, USA
Prof. Bernhard Plattner	Swiss Federal Institute of Technology ETH, Zurich, CH
Prof. Mehdi Jazayeri	University of Lugano, Lugano CH
Prof. Cesare Pautasso	University of Lugano, Lugano CH

Dissertation accepted on 28 November 2017

Research Advisor

Prof. Luca Maria Gambardella

Co-Advisor

Prof. Silvia Giordano

PhD Program Director

Walter Binder and Olaf Schenk

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Alan Ferrari
Lugano, 28 November 2017

To my beloved wife Giada.

Science does not know its debt to imagination…

Ralph Waldo Emerson

Abstract

With the advent of cloud computing, applications are no longer tied to a single device, but they can be migrated to a high-performance machine located in a distant data center. The key advantage is the enhancement of performance and consequently, the users experience.

This activity is commonly referred as *computational offloading* and it has been strenuously investigated in the past years.

The natural candidate for computational offloading is the cloud, but recent results point out the hidden costs of cloud reliance in terms of latency and energy; Cuervo et. al. illustrates the limitations on cloud-based computational offloading based on WANs latency times. The dissertation confirms the results of Cuervo et. al. and illustrates more use cases where the cloud may not be the right choice.

This dissertation addresses the following question: is it possible to build a novel approach for offloading the computation that overcomes the limitations of the state-of-the-art? In other words, is it possible to create a computational offloading solution that is able to use local resources when the Cloud is not usable, and remove the strong bond with the local infrastructure?

To this extent, I propose a novel paradigm for computation offloading named *anyrun computing*, whose goal is to use any piece of higher-end hardware (locally or remotely accessible) to offloading a portion of the application.

With anyrun computing I removed the boundaries that tie the solution to an infrastructure by adding locally available devices to augment the chances to succeed in offloading.

To achieve the goals of the dissertation it is fundamental to have a clear view of all the steps that take part in the offloading process. To this extent, I firstly provided a categorization of such activities combined with their interactions and assessed the impact on the system.

The outcome of the analysis is the mapping to the problem to a combinatorial optimization problem that is notoriously known to be NP-Hard. There are a set of well-known approaches to solving such kind of problems, but in this scenario, they cannot be used because they require a global view that can be only maintained by a centralized infrastructure. Thus, local solutions are needed.

Moving further, to empirically tackle the anyrun computing paradigm, I propose the anyrun computing framework (ARC), a novel software framework whose objective is to decide whether to offload or not to any resource-rich device willing to lend assistance is advantageous compared to local execution with respect to a rich array of performance dimensions.

The core of ARC is the *inference nodel* which receives a rich set of information about the available remote devices from the SCAMPI opportunistic computing framework developed within the European project SCAMPI, and employs the information to profile a given device, in other words, it decides whether offloading is advantageous compared to local execution, i.e. whether it can reduce the local footprint compared to local execution in the dimensions of interest (CPU and RAM usage, execution time, and energy consumption).

To empirically evaluate ARC I presented a set of experimental results on the cloud, cloudlet, and opportunistic domain. In the cloud domain, I used the state of the art in cloud solutions over a set of significant benchmark problems and with three WANs access technologies (i.e. 3G, 4G, and high-speed WAN). The main outcome is that the cloud is an appealing solution for a wide variety of problems, but there is a set of circumstances where the cloud performs poorly.

Moreover, I have empirically shown the limitations of cloud-based approaches, specifically, In some circumstances, problems with high transmission costs tend to perform poorly, unless they have high computational needs.

The second part of the evaluation is done in opportunistic/cloudlet scenarios where I used my custom-made testbed to compare ARC and MAUI, the state of the art in computation offloading. To this extent, I have performed two distinct experiments: the first with a cloudlet environment and the second with an opportunistic environment. The key outcome is that ARC virtually matches the performances of MAUI (in terms of energy savings) in cloudlet environment, but it improves them by a 50% to 60% in the opportunistic domain.

Acknowledgements

My deepest gratitude is to Dr. Daniele Puccinelli, without his guidance, encouragement and practical advice this achievement would not have been possibile.

My co-advisor, Prof. Silvia Giordano, has been always there to listen and give advice. I am deeply grateful to her for the support I have received in the past years.

I would like to express my gratitude to my Advisor Prof. Luca Maria Gambardella and my thesis committee members for all of their guidance through this process; your discussion, ideas, and feedback have been absolutely invaluable

I am also indebted to the members, friends and research colleagues at the Institute for Information Systems and Networking (ISIN) at the University of Applied Sciences and Arts of Southern Switzerland.

Most importantly, none of this would have been possible without the love and patience of my wife Giada to whom this dissertation is dedicated to, she has been a constant source of love, concern, support and strength all these years.

Contents

Figures

Tables

Chapter 1

Introduction

The disappearing computer is a concept introduced by Mark Weiser to indicate the trend in computer evolution where the computer is embodied in other devices. Smartphones and tablets are the best examples of this evolution allowing the users to access a rich set of functionalities with novel and natural interaction mechanisms (e.g. voice recognition) given the enormous number of applications that are currently available on the dedicated stores.

With the advent of cloud computing, applications are no longer tied to a single device, but they can be migrated to a high-performance machine located in a distant data center. The key advantage is the enhancement of performance and consequently the user experience.

This activity is commonly referred as computation offloading and it has been strenuously investigated in the past years. The cloud is the natural candidate for offloading, but there are circumstances where the cloud may not perform well; for instance, Cuervo et al. [2010] illustrates the limitations on cloud-based computational offloading based on WANs latency times. The dissertation confirms the results of Cuervo et al. and illustrates more use cases where the cloud may not be the right choice.

1.1 Motivation

Smartphones are continuously evolving and are becoming increasingly powerful day-by-day, but their structural requirements (especially portability) still pose some limitations in their computational capabilities, memory size, and battery capabilities. To overcome these limitations that affect the current smart devices several techniques have been proposed. One of them consists in the offloading of smartphone's computational needs to another device to reduce their local footprint.

With computational offloading, a set of programmatic instructions, the entire program, or even a virtual machine are run onto a remote device.

The Internet connectivity is becoming truly ubiquitous, and at the same time, the Internet is becoming increasingly cloud-centric. In many circumstances, It is far more cost-effective to outsource resource-intensive tasks to a powerful, dedicated, high-performance computing infrastructure than to run them locally. Consequently, the cloud is the natural place to offload, as it offers virtually unlimited resources and computing power.

Nevertheless, in some circumstances, the cloud is not the right choice and four scenarios have been identified where the cloud should be avoided:

- *Latency times:* when the high-speed mobile networks are not fast enough to offer a sensible gain in offloading.

- *Roaming:* costs are exacerbated due to the usage of a mobile operator in a different country.

- *Mobile Plan Costs:* the majority of mobile plans usually tend to offer only a certain amount of free data.

- *High-Speed Network Unreachable and Overloaded:* the high-speed mobile connectivity is not always available.

- *Privacy:* users want to maintain the control over their personal information.

Cuervo et al. [2010] have investigated the first scenario (Latency times), they have reconsidered the usage of 3G network and have decided to use a solution where the cloud is located in the proximity (cloudlet) to test their Computational offloading framework. In the dissertation, I show that the observations made by Cuervo et al. are still valid in modern network scenarios.

In recent years, several approaches (more or less) to cloud-free offloading have been proposed, and their details are provided in Section 2. Though these cloud-free offloading schemes are extremely valuable, they are not very flexible because they all require dedicated computing resources to run the offloaded code. Using a remote machine over WAN results in increased latency and suboptimal energy consumption, as shown by Cuervo et al. [2010], for best results the dedicated computing hardware should be within the same LAN as the devices that need to offload their code. Thus, state-of-the-art cloud-free offloading generally lacks flexibility because dedicated resources need to be known prior, and the offloading device is bound to share a LAN link with such resources. The unicast addressing methodology refers to this flavor of computational offloading as unirun computing.

1.2 Problem Statement

Moving the cloud to the edge of the network is the traditional solution for such kind of problem, and a full set of solutions are available in the literature (e.g. Cuervo et al. [2010]; Satyanarayanan et al. [2004]; Kosta et al. [2012]). The main limitation of the current solution is the lack of flexibility because dedicated resources need to be known beforehand, and the offloading device is bound to share a LAN link with such resources.

Is it possible to build a novel approach for offloading the computation that overcomes the limitations of the state-of-the-art approaches? In other words, is it possible to create a computational offloading solution that is able to use local resources when the Cloud is not usable, and remove the strong bond with the local infrastructure?

More in details, let us assume that a given device has a portion of an application that needs to be executed (the granularity of that portion may vary from method, thread, or even an entire virtual machine). The device can choose between a set of options; executing the code locally or asking to another device to take care of the job. The offloading is usually performed into cloud or cloudlet, but why not extend this option to other local devices? The core of the following dissertation is to provide a smart and dynamic solution to exploit the unused local resources (e.g. any laptop, computer or smart device).

Therefore, the problem approached in the thesis can be simplified as follows (a detailed categorization is offered in Section 3) :

- a is the portion of the application that must be executed (method, thread ...).
- N is the set of devices that are willing to take care of the offloading (they may be any device from the current device up to the Cloud)
- $C(n \in N, a)$ is a function that expresses the cost of executing that portion of code a in the device n (e.g. the energy footprint of the current device, WAN access costs, ...), given the current context.

The problem can be formalised as:

$$\underset{n \in N:}{\text{minimize}} \quad C(n, a)$$

In other words, given the current context (expressed in C) what is the best host to execute a?

The C function expresses all the possible vagaries in the execution process such as the local cost, the network vagaries in WAN communication, or the unpredictable contacts due to user mobility in local scenarios. In Section 3 there is a complete categorization of the limitation and a complete analytical definition of the problem.

The problem is an ILP (integer linear programming) problem that is notoriously known to be NP-Hard. There are a set of well-known approaches to solving such kind of problem, but in this scenario, they cannot be used because they require a global view that can be maintained only by a centralized infrastructure.

Thus, the previous problem has been re-formulated as follow: is it possible to find the best $n \in N$ with only a partial view of the network? The view is partial because I tackle the problem from the device viewpoint (a better categorization of the element viewed from the local device will be provided in Section 3).

This dissertation proposes anyrun computing; a novel paradigm for computational offloading that aims to solve this problem. To define anyrun Computing inspiration has been taken from the recent advances in computational offloading and opportunistic computing research areas to build a solution that can be defined as follows:
Rather than assuming the existence of a dedicated piece of higher-end hardware and being rigidly tied to it, anyrun may elect to use any piece of hardware (locally or remotely accessible) that can do the job better they can (this concept will be further clarified in the roaming of the dissertation).

The anyrun computing paradigm can be solved by dynamically deciding if a portion of an application a that is under execution in given device n should be offloaded or run locally, and on which device it should be offloaded. To take this decision on resource-constrained devices, I propose to use the bayesian approach, which is known to be very efficient and lightweight in terms of resources usage. However, specific focus is on the NaïveBayes approach, which requires relatively little computing power and provides an excellent inference quality compared to more sophisticated approaches. The NaïveBayes inference model is used to select the best device on which to forward the offloading of computation; in this sense, offloading is considered advantageous when there is a reduction of the energy footprint of the smart device.

1.3 Methodology

To remove the bound between the device that needs to offload and the device that provides the service (respectively seeker and provider from now on), due to the usage of solutions with fixed infrastructure. Several communication paradigms have been introduced. The one investigated in the dissertation is named opportunistic computing and, according to Conti et al. [2010b], it can be described as: "when two devices come into contact, opportunistically, it is also a great opportunity to share and exploit each other (software and hardware) re- sources, exchange information, cyber forage, and execute tasks remotely. This opens a new computing era: the opportunistic computing era". The key idea is to extend the concept of opportunism that is translated in the possibility to choose the best option to perform the offloading between cloud, cloudlet and any other devices that are locally available down to the local device (in case there are no other options that provide a sensible gain). The selection of the remote host is crucial and a key challenge in anyrun Computing; a system that approaches the paradigm must be able to detect which high-performance host (located into cloud, cloudlet, or proximity) is able to perform better compared to the others (the current host is also included in the list).

Therefore, I defined a methodology based on the Naive Bayes inference model to select such best remote host as formally introduced above.

To empirically study the anyrun Computing solution I have proposed I developed The AnyRun computing framework (ARC) a software solution that approaches the anyrun Computing paradigm with the scheme presented above.

In anyrun computing, it is impossible to maintain a global fairness because the usage of opportunistic solutions removes the global view of the network (it could be maintained only with cloud-based solutions). For this reason, the AnyRun Computing paradigm cannot be translated into an optimization problem.

The removal of a global view poses also other challenges; for instance, by removing the cloud we also remove the "trusted host" (thus Security and Privacy become two key issues to be solved in the opportunistic domain.

1.4 Contributions

The main focus of this thesis consists of providing a solution to the offloading problem. The main objective is to provide a solution for computation offloading that is not only tied to physical infrastructure but it can also exploit high-performance devices via opportunistic access.

Several challenges must be addressed: the first one consists in having a strategy to select the best host (that can be located in a cloud, cloudlet, or accessed opportunistically) who will take care of the offloading. This is a key problem because selecting the wrong host can lead to low performances. Other aspects to investigate include the management of security and privacy that are crucial given the distributed nature of the solution. Further, the evaluation of the proposed solution is highly challenging: being the problem interleaved with many real and physical aspects, i.e. power consumption or real mobility of users, it cannot be evaluated analytically. As traditionally done in literature, I, therefore, decided to empirically evaluate my solution. However, differently, from what has been done till now, I realized that was paramount of having the real measurement of power consumption and not simply the measurement numbers given by the devices operating systems. In fact, such numbers are affected by the whole context (e.g. the other code currently running on the device) and do not allow to correctly reproduce the experiments, and thus fair comparison. As I explain later, this required to develop a new testbed environment.

After providing a set of use-cases that illustrate the limitation of cloud and cloudlet approaches to better contextualize the problem, I introduce an analytical formulation of the anyrun Computing problem. A simplistic view of the key components and their interaction is: the process starts in the host that is interested in offloading a portion of its application (seeker), at the beginning, it has to select the best remote host to satisfy its request (provider). When the provider has been selected the seeker forwards the data and eventually the code that must be executed and goes to sleep. It will be woken up from the selected provider when it forwards the results of the offloading or eventually, it will be woken up by a timer indicating that the process takes too long to return the results thus the execution is recovered locally.

To better approach and study the problem, I propose another key contribution of this thesis that consists of the implementation of a software solution to approach the anyrun computing paradigm. The solution provides a host-selection mechanism that is able to provide accurate selection but at the same time avoid to impact the computation cost, the key challenge is that the solution only has a partial and incomplete view of the network because a complete view requires a global view that could be maintained only with the cloud-based solution. The solution also provides mechanisms to maintain security and privacy in such hybrid domain where in many circumstances there is no central authority. Thus, the solution is able to:

- Match the performance of state-of-the-art solution in traditional and stationary network conditions.

- Outperform the state-of-the-art in a more dynamic and realistic environment to a better host-selection policy and the augmented reachability in terms of devices.

To make a host selection decision on resource-constrained devices, reference is made to the bayesian approach, which is known to be very efficient and lightweight in terms of resources usage. Specific focus on the NaïveBayes approach, which requires relatively little computing power and provides an excellent inference quality compared to more sophisticated approaches.

Ferrari et al. [2015a] provides a comparison of three machine learning techniques applied to activity recognition problem. The authors consider two key measurements: the accuracy of the results and e the energy spent in the process. The main outcome of the analysis is that NaïveBayes generates good performances (up to 85% accuracy with the proposed problem) and it has a very limited impact on the energy consumption due to its linear complexity. The work justifies the choice to use Bayes in the Offloading domain because the other approaches require a prohibitive computational effort; in fact, a neural network has been used as a benchmark problem in the dissertation.

To empirically study my solution I propose ARC (anyrun computing framework) a software solution whose goal is to dynamically decide whether any run of a method should be offloaded or run locally and on which device it should be offloaded.

We note ARC does not ensure global fairness because it would require a global view that could be maintained only with cloud-based solutions.

The host selection process made in ARC seeker is crucial and the most challenging part. It implements a NaïveBayes as proposed in the dissertation as inference mechanisms to select the adequate remote host. The decisional process receives a rich set of information about the available remote devices to decide whether offloading is advantageous compared to local execution, whether it can reduce the local footprint compared to local execution in the dimensions of interest (CPU and RAM usage, execution time, and energy consumption).

ARC is evaluated in my testing environment that is composed of two key elements:

- Energy profiling: energy profiling has been performed with success for decades, however, novel smart devices have a set of peculiarities that require new solutions. The first problem is in low-granularity of reading frequencies of the battery on modern devices, and the second is a dynamic environment where smart devices operate (mainly dictated by device mobility). A solution that has been used to test ARC is presented and evaluated with a simple example in the next section.

- Experiment replication: offering a fair comparison between different solutions in the anyrun domain requires that the same environment is replicated in different experiments. To approach this issue I propose DroidLab; a testbed that offers realistic emulation of a dynamic environment but it also offers full control over it.

In the evaluation, I have shown that the main limitations in adopting a cloud-based approach are still actual and strictly connected to the difference between computation and transmission costs. After, I demonstrate that ARC virtually matches the performances of the state of the art in cloudlet environment, but the performances are improved by a 50% to 60% in opportunistic domain compared with the state of the art.

Concluding, the results of the work extend the state-of-the-art in several areas ranging from opportunist computing to power measurements on smart devices, and it also opens the door to novel application scenarios. For instance, a possible use-case may be to deploy a set of resources in specific locations and offer them as an incentive to go there (e.g. a restaurant may be interested in offering it as a service to its client). Another interesting use-case is the use of local resources like the laptop to improve the user experience in virtual reality applications; it is notoriously known that virtual reality is computationally expensive and impacts heavily on the smartphone battery lifetime. With the proposed solution it is possible to offload the computation to another device located in proximity to improve the quality of the results and reduce the energy footprint of the smartphone.

1.5 Thesis Organization

The rest of the thesis is organized as follow:

In chapter 2 I present the state of the art that is mainly centred in two research areas; the first, named opportunistic computing is an appealing solution to a wide range of problems where the cloud cannot be used, and the second one is mobile code/computation offloading a solution to lighten the use of resources in smart devices and at the same time augment the performances. In this section, the state of the art of the two research areas is deeply illustrated from the point of view of functionalities and security. The chapter 2 concludes by illustrating a set of paradigms that have a connection with the work presented in the dissertation.

The connection between them and the work presented in the dissertation will be clarified further in the thesis.

In chapter 3 I present an analytical definition of the anyrun Computing paradigm. In conjunction ARC framework is presented, a software solution specifically created to work in the anyrun scenarios. At the end of the chapter the reasons behind the AnyRun paradigm are deeply investigated; the chapter concludes by illustrating the main architectural characteristics of ARC.

In chapter 4 I illustrate a novel environment specifically created to measure anyrun-like solutions. The environment is composed of two main elements: an accurate energy profiling, and a testbed that offers a realistic emulation of a dynamic environment.

Chapter 5 shows the empirical investigation performed for both scenarios. At first, the benchmarks used to perform the evaluation are presented and compared. Then, the testing environment is described in the previous chapter, so the setup and the results of the evaluation are presented.

The final chapter 6 summarizes the work described in this thesis with an overview of the key results. The chapter will also provide the future directions derived from this work and the results of the scientific publications on which the results of the research has been disseminated.

Chapter 2

Related Work

2.1 Introduction

In the past years, the computer has evolved into a wide variety of smart devices. This mutation has been predicted by Mark Weiser more than two decades ago in his remarkable work "Computer of the 21st century" (Weiser [1991]), in this publication the term *ubiquitous computing* has been proposed to indicate a set of novel devices that are small, inexpensive and are part of our daily lives in various form. Devices that follow such paradigm are nowadays available on the market and offer a full range of services to their users.

Starting from the vision of Mark Weiser, many researchers have created novel research areas. The work proposed in the following dissertation took inspiration from two of them; the first, is Opportunistic Computing an appealing solution to a wide range of problems where the cloud cannot be used, and the second is mobile code/computation offloading a solution to lighten the use of resources in smart devices and at the same time increase the performances. In this section, the state of the art of the two research areas is deeply illustrated from the point of view of functionalities and security.

At the end, a set of paradigms that have a connection with the work presented in the dissertation are illustrated. The connection between them and the work presented in the dissertation will be made clear in Section 3 where the core solution of the dissertation is deeply investigated.

2.2 Mobile Ad Hoc Networks (MANETs)

Before explaining the opportunistic computing paradigm it is necessary to start from the mobile ad hoc networks (MANETs) paradigm, then move into the opportunistic networks. After, the opportunistic computing paradigm can be introduced.

A lot of research efforts has been dedicated in the field of MANETs during the past years; the key elements that distinguish them compared to traditional network paradigms are the self-forming and self-healing concept that enables peer-level communication between mobile devices (also called nodes) without relying on any centralized resources. bluetooth or WiFi (nowadays in the form of WiFI direct) are the most common communication solutions providing a device ranging from 3 to 30 meters for Bluetooth to 25 to 120 meters for WIFi. The main contribution offered by MANET-based solutions compared to traditional network solutions is the fast deployment and maintenance of a network in ad-hoc mode, features that could be exploited in many fields (e.g. military and emergency services).

Routing is a key element in MANETs and has attracted the majority of scientific research in the past years. The central challenges are dictated by the mobility of the devices that creates unstable connections, thus the optimal routes change over time.

Dozen of protocols have been proposed such as OLSR (Clausen et al. [2003]), AODV Perkins et al. [2003], and many more (a good survey is provided by Liu and Kaiser [2005]. Routing algorithms can be distinguished in three main categories: **proactive**, **reactive**, and *hybrid*.

- *Proactive Protocols:* In Proactive protocols nodes continuously sample the network to maintain the routing tables always updated. These protocols offer high performances in terms of network setup and message delivery rate, however, they requires extra costs to maintain the table always up-to-date.

- *Reactive Protocols:* Nodes set-up the route on-demand. They offer high scalability compared to proactive protocols, however, they may require long delays to discover the route.

- *Hybrid Protocols:* The key idea consists in forming clusters of nodes using proactive algorithms,then exploit such structures to help increase the performance of reactive algorithms.

There are different types of MANETs that strongly depend on the end-to-end device used. For instance, VANETs (Vehicular Ad Hoc Network) are MANET applied to vehicles that are the core component of the future intelligent transportation systems.

The key limitation of the MANET paradigm is that whenever a node is not reachable messages cannot be properly delivered. Even if this seems a trivial conclusion, it will be explored in the next sub-section that it is one of the distinctive element among MANET and Opportunistic Network

2.2.1 Service Discovery and Selection in MANET

There are several applications that can take advantage of a MANET architecture ranging from network-recovery in a post-apocalyptical world up to the temporary group collaboration to share some kind of content or give access to functionalities that only a small set of nodes own. The concept of Service Discovery is the bottom layer to build system that offer services into a MANET domain. One of the early work on this domain is proposed by Kozat and Tassiulas [2003] where authors propose a service discovery mechanism for Ad-Hoc networks where the protocol builds a virtual backbone and assumes that all the nodes with distance at least 1-hop are part of the backbone. The system does not provide any solution for service selection in case multiple instances are available. This problem is approached by Helal et al. [2003] where authors build a SOAP-like protocol for MANET to be built on top of a multicast protocol for service-discovery.

A key element that distinguishes the previously mentioned approaches is that they do not use any routing algorithm. Koodli and Perkins [2002] propose an extended version of AODV that supports service discovery. The approach has been shown not to be very effective because it generates a tremendous amount of traffic to satisfy all the client's request.

Mechanism that do not rely on multicast or flooding are currently under investigation. Sailhan and Issarny [2005] propose a scalable service discovery protocol that uses a dynamic deployment of directories within the network. To address the problem of flooding and service differentiation Lenders et al. [2005] propose a decentralized mechanism that uses an analogy with electrostatic fields as heuristic. Each service is modelled as positive charge and the service requests are modelled as negative charge, which are attracted to service instance. Routing request are directed towards the high potential service allowing an implicit optimization. I Artail et al. [2008] propose the Minimum Distance Packet Forwarding (MDPF) algorithm to select the best possible service considering the distance between nodes as central metric, a set of nodes act as clusters headers and stores such metrics in conjunction to other device/service information.

Service discovery in MANET continues to be a hot topic and with the advent of novel smart devices in the domain of Mobile Cloud Computing (MCC) more information can be used to determine the best service that can fulfil a given need. Zhou et al. [2015] propose a context-aware algorithm that uses both network and remote device usage statistics to associate the best host for a given service need.

2.2.2 Security in MANET

The particular characteristics of MANETs (lack of centralized authorities, dynamic topologies, ...) generate free space for novel Security threats. They can be categorized in two distinct groups:

- *Availability Threats:* where resources (like bandwidth or even battery) are disturbed (e.g. abused) such that other nodes cannot get access to them.

- *Data Theft Threats:* where devices can access sensible data without getting the consent of the owner.

In the first case (Availability Thread) one of the most common attack is known as *Blackhole* Al-Shurman et al. [2004] that occurs when a node advertises that it is a good candidate for a given route, but it stops the packets. GrayHoles (Shanmuganathan and Anand [2012]) is similar to Blackholes, but the malicious node stops only a set of selected packets, so it is much harder to find out who is responsible. In Bizantine Attacks (Awerbuch et al. [2002]) a node creates a non-optimal route, but it advertises it as optimal; this attack is very hard to detect because from the viewpoint of the device, the MANET seems to work properly. Finally, DoS attacks are a very destructive element in MANET, as well as in other domains. Such kind of attacks may be performed in many forms. Probably the most challenging to detect is known as Dynamic DoS (DDOS) (Shanmuganathan and Anand [2012]) where in MANET it is able to exploit node mobility to create a devastating attack.

In the second case (Data Theft Threats) several possible threats were found like the Spoofing Attack where an attacker masquerades itself as a legitimate node to bypass the security controls and gain access to sensible resources. In the Sybil attackDouceur [2002] a user creates multiple fake identities, when the number of nodes become significant compared to the total number of nodes in the network, the fake nodes can impact the overall behaviour of the MANET in all tasks that require collaboration (e.g. Routing).

Countermeasures in MANET are divided in two categories: cryptographic-based and trust-based. In cryptographic-based authentication encryption is at the center of the solution. One of the first protocol that addresses this challenge is SAODV (Lu et al. [2009]) that is an extension to standard AODV meant to support security. The key concept of SAODV is that each node has a signature key generated by an asymmetric crypto-system. The algorithm requires that each node is able to verify the key-address pair and the public key they offer. On the other hand, in a trust-based system the remote node reputation is used as a feature to provide a form of identification in the network. For instance TAODV (Uddin et al. [2012]) is an extended version of AODV that uses subjective logic as a tool to estimate the reputation.

2.3 Opportunistic Networking

In a sparse system where nodes have a high degree of mobility and the connection falls very frequently the solutions proposed in MANET domain will not work properly. The reachability limitation needs to be lifted so that a message can be delivered even if there is no path at the time it is sent. This is the key concept of *opportunistic network* that corresponds to **disconnected** MANET where nodes exchange messages *opportunistically*.

Key elements of such paradigm are denoted as islands (a.k.a regions) that can be defined as a set of isolated hosts. The islands may be created, destroyed, or modified in several situations. A wide variety of possible algorithms have been proposed.

The approach named *store, carry and forward* (Fall [2003]), is usually adopted in the DTN, (Delay Tolerant Networking), and may also be adopted in opportunistic networks. This approach uses nodes denoted as *mules* that move across different islands to propagate the messages between the two groups.

Even though "store, carry and forward" is a very promising approach the unpredictability of the time where a message is delivered makes it almost impossible to know beforehand the amount of memory needed to carry all the messages between islands. Therefore, the routing schemes should seriously consider these issues if they want to be effective.

During past years the research community has provided a wide variety of protocols for opportunistic networks. Following the taxonomy proposed by Huang et al. [2008], there are two main approaches to build such algorithms:

- **forwarding based**, where a single copy of the message exists at a given time;

- **flooding-based**, where multiple copies of the message are carried around at the same time.

The first approach is clearly more conservative in terms of memory used with, the second one attempts to reach the destination node with as many intermediate nodes as needed. A good survey of the challenge that has to be faced in opportunistic networks and routing in opportunistic networks is provided by Conti et al. [2009] and Pelusi et al. [2006].

One of the first approaches for forwarding-based schemes has been proposed by Spyropoulos et al. [2004] and is called *direct transmission*, where a node stores the message until it meets the final destination. This approach offers advantages in terms of memory consumption, but it might require an infinite time to reach the final destination. Other solutions exploit the prediction of future locations to determine the set of devices that are or will be close to the destination before sending the message. *MoVe* proposed by LeBrun et al. [2005] uses the knowledge of the relative velocities of the nodes to predict their future location.

Using a variegated set of information about the other nodes might be beneficial in such algorithms. For instance, the Context-Aware Routing (CAR) solution proposed by Musolesi and Mascolo [2009] chooses the node to forward the message by computing the so-called delivering-probability that is computed using prediction techniques based on Kalman filters and utility theory.

In flooding-based approaches, each node stores the message until it finds a way to reach the destination. *Epidemic Routing* by Vahdat et al. [2000] that is a variant of the Epidemic Algorithm (a well-known solution to synchronize replicated databases) forwards the message to each reachable node. For this reason, it requires a tremendous amount of memory. To overcome this limitation *Spray And Wait* is proposed by Spyropoulos et al. [2005]. The algorithm controls the level of flooding by limiting the number of copies that are sent across the network.

Similarly to the forwarding-based scheme, to filter off nodes that will not be able to deliver the message prediction-bases approaches are proposed in flooding based scheme. One of the first is PROPHET by Lindgren et al. [2003]. At first, it computes the probability of success in delivering the message by looking at the contact. PROPICMAN by Nguyen et al. [2007] extends even further PROPHET by looking at the context of the node and it computes the probability to successfully deliver the message if sent to a given intermediary-node by looking at the device context and the contact history.

Nguyen and Giordano [2012] propose a novel context-aware routing scheme (CIPRO) that includes the spatial and temporal dimension of the activities performed inside mobile nodes to predict future mobility patterns

Services discovery and service selection functionalities have also been studied in opportunistic networks. Passarella et al. [2011] develop an analytical model to study the behavior of service providers and seekers. The model considers multiple providers and requests with different delays and parallel execution. This study is a building block of the Opportunistic Computing paradigm that will be explained in the next subsection.

The first notable implementation of a framework that supports the Opportunistic Network paradigm has been proposed in the Haggle project (Scott et al. [2006a]) where windows mobile OS has been used as a development platform and the potential of this kind of architecture has been demonstrated with real mobile users. Similarly to Haggle, the Metrosense project[1] implements a similar architecture for Nokia's Symbian OS platforms to carry around information opportunistically (information can be air quality, weather conditions, ..).

More information about routing solutions in Opportunistic Network is provided by Woungang et al. [2013] and Conti et al. [2009].

2.3.1 Security in Opportunistic Network

This new communication paradigm creates a set of interesting security challenges that must be addressed. At first, opportunistic networks inherits all the security challenges from MANETs and extends them even further. For each forwarding mechanism a different security mechanism is required; for instance, epidemic forwarding suffers from DoS attacks and context-based algorithms by allowing the leakage of sensitive information.

Similarly to MANETs, reputation-based solutions have been proposed in opportunistic networks to cope with the lack of central authorities. The Sybil attacks are one of the major opponents to a full adoption of reputation-based schemes. Trifunovic et al. [2010] propose an extension of a traditional reputation-bases scheme that uses social network structure and establishes trust among devices dynamically.

The Haggle platform (Scott et al. [2006a]) uses a certificate exchange to control the access to sensitive information (i.e. in the context-exchange phase), with the certificate exchange nodes that prove that they belong to a certain community. The certificates must be pre-deployed by the device owner.

[1]http://metrosense.cs.dartmouth.edu/

2.4 Opportunistic Computing

A lot of work has been carried out in recent years in opportunistic networking. One of the key questions that have been raised is "how to extend the opportunistic concept beyond communication?". This is the case of **Opportunistic Computing**.

According to Conti et al. [2010b], "when two devices come into contact, opportunistically, it is also a great opportunity to share and exploit each others (software and hardware) resources, exchange information, cyber forage, and execute tasks remotely. This opens a new computing era: the opportunistic computing era".

With opportunistic computing, we have a new distributed computing paradigm that involves transient and unplanned interactions among mobile devices. It extends the concept of opportunistic networks allowing devices to tap into each other's resources in the form of services (exchange may include but is not limited to multimedia content, sensors or data ...). Challenges in such domain are an extension of the challenges presented in Service-Oriented Architecture (SOA) where node mobility plays a key role.

Similar paradigms have been adopted in SociableSense (Rachuri et al. [2011]) where authors present their approaches to quantify the user's sociability and illustrates how computationally intensive sensing tasks can be offloaded to the cloud in fixed networks. Shankar et al. [2012] proposes a system that collects and indexes updates in mobile social networks. The authors illustrated how user interaction can be used to reduce the services costs.

One of the first studies on the impact of resource availability of human mobility is by Ferrari et al. [2012] where metrics named Expected Resource Availability (ERA) are presented. The goal of the metric is to evaluate the quality of a given mobility-related scenario in the presence of a given set of services in the context of opportunistic computing. Within this work, some preliminary simulation results confirm the validity of the ERA approach to find a possible service allocation.

The first framework that supports the opportunistic computing paradigm is presented in the EU FP7 SCAMPI[2] project and is named the SCAMPI framework. To tackle the project the SCAMPI framework (Kärkkäinen et al. [2012]). The figure 2.1 shows the rationale of the framework performs three distinct tasks:

- Selecting an appropriate service sequence set out of available services.

[2]http://www.ict-scampi.eu

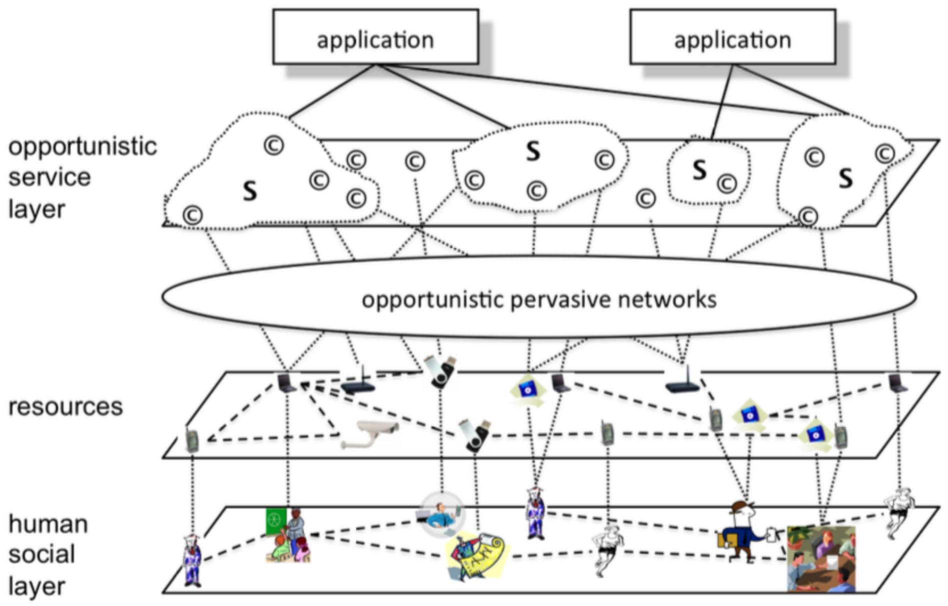


Figure 2.1. Rationale of the SCAMPI framework

- Forwarding service inputs to the device hosting the next service in the composition.

- Routing final service outcomes back to the requester.

The framework has been implemented and tested into Android-based platforms.

2.4.1 Security in opportunistic computing

To empirically study the opportunistic computing paradigm, the Twimight application (Hossmann et al. [2011]) has been developed within the SCAMPI project. Twimight is an open source Twitter client for Android phones that includes a disaster mode. In other words, it enables opportunistic communication between users in case they lose the direct connectivity to the Internet. In the disaster mode, tweets are not sent to the Twitter server, but stored on the phone, carried around as people move, and forwarded via Bluetooth when in proximity with other phones.

This passage to a decentralized mode requires a security architecture that protects Twimight from basic attacks. The security architecture of Twimight is proposed by Hossmann et al. [2011] and is based on the assumption that security elements (certificates and keys) can be established before a disaster happens.

When connectivity is lost and the disaster mode is activated, the protocols can no longer rely on a central authority. This hybrid security scheme simplifies the design of the architecture considerably compared to a fully distributed security architecture, but the key drawback is that when in disaster-mode new users cannot join the network.

The key elements of the Twimight security architecture are illustrated in Figure 2.2. The core element is the Twimight disaster server that manages the security primitives in terms of keys and certificates. Once the device goes into disaster mode, the communication with this element is compromised and the disaster-mode communication is started. This architecture has been validated in a realistic scenario.

Security and Privacy in opportunistic computing have also studied from a physical-layer perspective. Muntwyler et al. [2012] uses random spreading sequences in order to obfuscate the communication in the IEEE 802.15.4 physical layer.

2.5 Mobile Code/Computation Offloading

Smartphones or tablets are very attractive to consumers for a wide variety of reasons. To maintain this attractiveness they require a trade-off between their size and resources (in terms of battery, memory and computational capabilities). In parallel, applications like gaming with complex AI functionalities or augmented reality are created and deployed in the application market. Those novel applications heavily impact on the device resources and the user experience.

The rapid growth of capabilities and availability of cloud computing [3] has opened new doors for mobile developers. Massive scalable and high reliable hardware is made available at a very competitive price; this solution has allowed developers to adopt the so called **cloud assisted execution** where computation is distributed across both mobile and cloud devices. For instance Chun et al. [2011] propose *CloneCloud*, a framework that uses a static analysis and dynamic analysis of the application code to dynamically partition the application and migrate a thread at runtime whenever needed. Therefore, results are integrated into the current execution when and if the thread comes back to the mobile device.

The rationale behind this approach is that the offloading system transforms a single application into a distributed execution that can leverage both local and remote resources to enrich the user experience.

[3]http://www.forbes.com/sites/louiscolumbus/2016/03/13/roundup-of-cloud-computing-forecasts-and-market-estimates-2016/

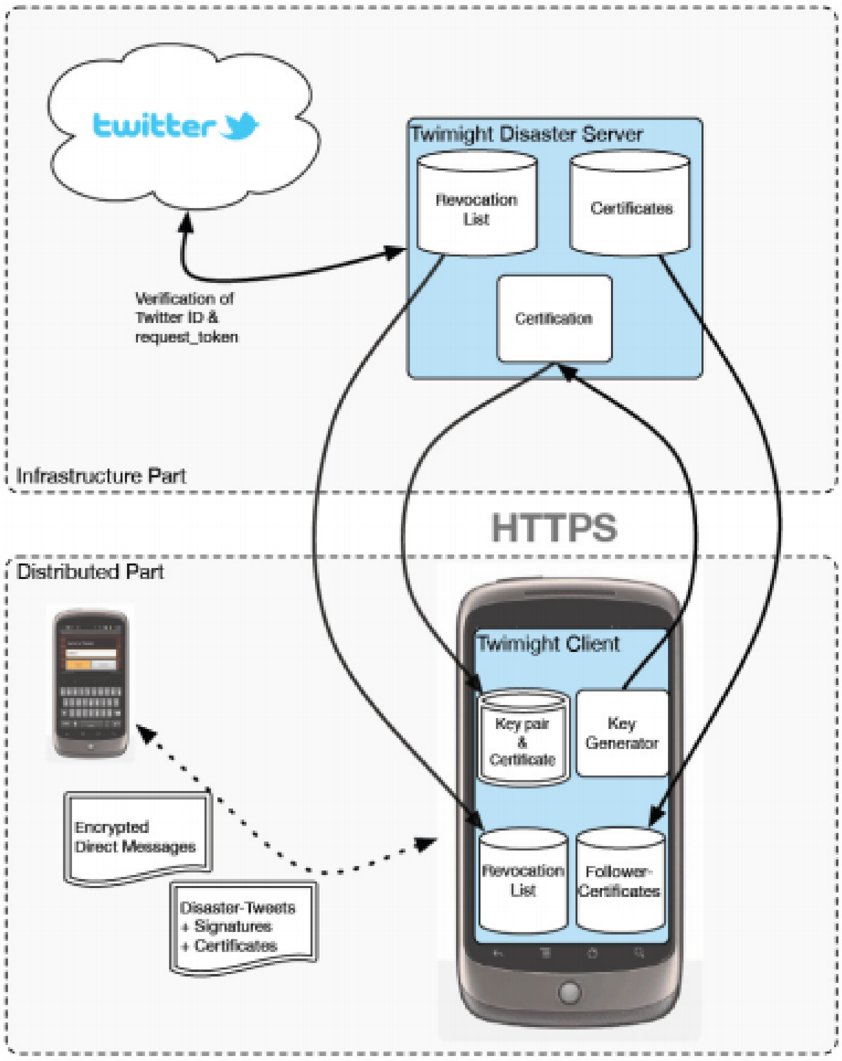


Figure 2.2. Hybrid security infrastructure (courtesy of Hossmann et. al.)

2.5.1 Mobility of the Computation

As defined by Carzaniga et al. [1997] mobile code languages (MCL) are languages that offer tools to support mobility of the execution and of the code by itself at runtime. MCL can offer two types of computational mobility:

- **Weak Mobility:** where the MCL binds dynamically with the code coming from a different site, but no execution states (values from the stack and registers) are transferred.
- **Strong Mobility:** where code and execution states are moved into a different site.

By analyzing the current state of the art in computation offloading a third case must be added:

- **Semi-Strong Mobility:** where only the execution states (in the form of data from stack and registers) are moved.

In many operating system (e.g. iOS [4]) this solution is blocked to maintain an adeguate level of security.

On Android platforms, the support of code/computation mobility is offered in the form of semi-atrong mobility, but with the appropriate permission, strong mobility can be performed (See Section 3). The Dalvik class loader and Java object serialization allow a graph of objects to be transformed into a binary stream and vice versa. The serialization extract data from the memory stack allows the object instance to be fully replicated in the remote site. Other operating systems (iOS and Windows Mobile) provide a similar solution.

The key element to run the program remotely is the **reflection** mechanism that consists in the ability to manipulate and examine the structure of an object at runtime (e.g. check all the possible method of a given object and even call them). Reflection is a basic mechanism on each Java-based virtual machine thus is part of the Dalvik/ART as well. Similarly iOS and Windows Mobile provide similar features.

2.5.2 From Cloud to Cloudlet

Nowadays, the situation in mobile communication is defined by Cisco as the "mobile data tsunami" [5]. This definition is dictated by the fact that in recent

[4]https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[5]http://blogs.cisco.com/news/the-mobile-data-tsunami-2

years the number of mobile users has increased by 50% and the mobile traffic has grown 62 times from 2011 to 2016, but only 36% of network coverage has been migrated to 4G.

The most usual circumstance where the network bandwidth cannot offer adequate support is when it cannot be reachable or the bandwidth is strongly used, causing delays. Other circumstances might be due to communication costs (i.e. roaming) or by limitations of the current mobile-network plans. Nowadays, in Switzerland, the standard unlimited plans come with a network usage limit; if it is reached the network bandwidth is reduced.

The common circumstance is that transmission time takes so long that it impacts the overall performance of the offloading to a distant cloud. Empirical evidence of this problem has been shown by Cuervo et al. [2010].

However, due to the evolution of communication technologies, Cuervo's experiments have been replicated with a modern networking paradigm in Section 5; the main finding is that the transmission time is still a bottleneck and the problem illustrated by Cuervo et. al. still exists.

To address the inherent resource-poverty of mobile terminals along with the setbacks of relying on distant clouds, the *cloudlet* model (Satyanarayanan et al. [2004]) has been proposed by Satyanarayanan et. al., who empirically shows the limitations of WAN-based cloud solutions and proposes a novel approach based on accessing high-resource devices located in close proximity.

2.5.3 CloudLet-base approaches

One of the first notable examples is the ThinkAir framework from Kosta et al. [2012] that proposes a novel computational offloading architecture based on smartphone virtualization in the Cloud. The authors provide method-level computational offloading. The offloading strategy is chosen based on a method's energy footprint and device status in terms of resource usage and network connectivity. They also show that the offloading gain in terms of energy consumption is one order of magnitude greater compared to local execution.

The most notable work on such domain is the Mobile Assistance Using Infrastructure (MAUI) system (Cuervo et al. [2010]) that has been recently proposed by Cuervo et al. to enable the fine-grained energy-aware offload of code from a mobile device to a MAUI node, *i.e.* a nearby piece of infrastructure connected to the mobile device by a high-performance WLAN link. MAUI aims to reduce the energy footprint of mobile devices by delegating code execution to remote devices; it dynamically selects the function to be offloaded depending on the expected transmission costs of the network and provides an easy way for the developers to use the framework in their code. Mobile terminals can leverage cloudlets of nearby infrastructure that can be accessed over Wi-Fi. This is certainly a promising strategy, especially given the recent results on the advantages of augmenting 3G with Wi-Fi Balasubramanian et al. [2010]. It is shown in Cuervo et al. [2010] that "the cost of 3G for computational offloads is prohibitive compared to Wi-Fi, and that the energy consumption of computational offloads grows almost linearly with the Round Trip Time (RTT)": using a nearby server is much more beneficial and energy-efficient than using a distant cloud, which confirms the conclusions of Cuervo et al. [2010].

The webOS 2.02 proposed by Ostrowski [2012] provides a Node.js3 runtime environment where developers have the possibility to write event-based network services using Javascript that can run on both server-side and on webOS 2.0 devices.

Fernando et al. [2011] explore the feasibility of using a mobile cloud computing framework to perform job-sharing in the ad-hoc and opportunistic network. Authors prove the usefulness of such approaches by a set of experiments via Bluetooth and they conclude by proposing an analytical model that determines where and if is convening to perform the offloading. The key limitation of such approach is that they do not consider the vagaries of the network.

Mavromoustakis et al. [2014] demonstrated, thanks to a simulation, the possibility to improve the quality of offloading thanks to the learning of the user's social context.

Chen et al. [2012] explore the design of a system able to automatically partition Android applications without modifying the underlying system. Basically, the application extracts the android-services and redeploy them inside a "virtual-smartphone" in the cloud. They also provide a set of decision metrics do decide whether offload or not, the metrics have to be computed before and they consider the local execution costs vs. the transmission costs. This approach is clearly flawless, first because of the network changes during time thus the metrics are not available, secondly because services are composed by a whole set of activity and many of them require accessing the local context (i.e. access to location) thus they cannot be remotely deployed.

Mtibaa et al. [2013] propose the idea of **mobile device cloud**, where the union of mobile devices located in proximity may be used to perform computationally intensive tasks. The authors built a framework and performed a set of experiments to show the feasibility to use a mobile device cloud to perform offloading. The authors also showed by means of simulation that the contact-history and the relationship between the users can be used to predict future contacts and thus used in an offloading scheme. A similar scheme can also be found

Framework	Granularity	Offloading Decision
μCloud by March et al. [2011]	Component	Fix
MACS by Kovachev et al. [2012] 9	Component	Dynamic
CloneCloud by Chun et al. [2011]	Thread	Dynamic
COMET by Gordon et al. [2012]	Thread	Dynamic
MAUI by Cuervo et al. [2010]	Method	Dynamic
ThinkAir by Kosta et al. [2012]	Method	Dynamic
Cuckoo by Kemp et al. [2012]	Method	Dynamic
WebOs 2.02 by Ostrowski [2012]	Component	Fix

Table 2.1. List of computation offloading frameworks. The granularity indicates the level at which the framework offloads the application. The offloading-decision indicates if there is a certain level of self-configuration in taking the offloading decision given the status of the environment.

in Mtibaa et al. [2015].

Chen et al. [2015] propose a theoretical formulation for mobility assisted computational offloading. They formulate the *optimal computational offloading* problem as a convex optimization problem with is NP-Hard. The key limitation of this approach is that it requires the global view of the network. Chen et al. [2015] propose a similar scheme and they also include the design of a software architecture that supports this paradigm.

A significant amount of research has been dedicated to the study of the offloading framework. Table 2.1 compares the most notable software frameworks available in this area according to the architectural decision authors have made.

Recently, offloading techniques have been successfully applied to collaborative rendering. Cuervo et al. [2015] provides Kahawaii, a system able to provide high-quality gaming on smart devices by offloading a portion of the GPU computations to server-side infrastructure. The distinctive element is that Kahawai uses collaborative rendering to combine the output of a mobile GPU and a server-side GPU into the displayed output.

2.5.4 Privacy and Security in D2D Offloading

Privacy and Security in the new domain of D2D offloading is not a trivial task. In the context of privacy, Langheinrich [2001] extensively defines the concept of privacy by design in ubiquitous computing and divides the challenges into seven main areas:

- Principle of openness or simply notice: users should be aware of the nature of the data shared.
- Choice and consent: where users can choose to offer that information to the requester and they have to give explicit consent.
- Anonymity and pseudonymity: anonymization can be defined as the state of being not identifiable within a set of subjects. The larger the set of subjects is, the stronger is the anonymity according to Pfitzmann and Köhntopp [2001]. Instead, pseudonymity allows more control of anonymity e.g. adding an ID to a name of a person.
- Proximity and Locality: in essence, it expresses the fact that information must not be disseminated indefinitely, not even across a larger geographic boundary.
- Adequate security: Network and disk security are fundamental.
- Access and recourse: provide mechanisms for access and regulation. Eventually, also penalties if someone breaks the rules.

Starting from the previously mentioned areas a lot of effort has been devoted in that area. An important example is a current research in differential privacy (a good survey is provided by Dwork [2008]) that is currently implemented natively into modern devices[6]. The key goal of differential privacy consists in maintaining anonymization in a single data-entry but allowing to use statistical inference on top of the whole set of data.

Another key aspect is security in offloading. At the time of writing, there are very few solutions that approach security in D2D offloading. Traditional applications simply consider the fact that a remote device is willing to cooperate and does not act selfishly. Unfortunately, this is not always the case. To address this issue Anderegg and Eidenbenz [2003] propose a payment-based mechanism controlled by a central authority. Michiardi and Molva [2002] provide a reputation mechanism where reputation is still managed by a central entity. As mentioned before Trifunovic et al. [2010] proposes a similar scheme in a distributed environment. Chatzopoulos et al. [2016] uses the principle of hidden marked design to build a framework that integrates an incentive scheme and a reputation-based mechanism. The authors have validated their approach with simulation but they also clearly depict the limitations of this solutions. They are:

[6]https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/

- *Malicious behaviour of users:* users can use a wide variety of attacks to impact the overall system performance. For instance, if a group of users join resources, they can virtually augment their reputation.
- *Privacy of the Information:* In D2D offloading, personal information is shared among different devices and can be easily stolen.

In conclusion, I would like to reiterate that privacy and security do not play a key role in the following dissertation, but some preliminary solutions (especially in the privacy domain) are presented. To address the challenges that have been raised in the dissertation, the project CHIST-ERA2017 UPRISE-IOT[7] has been initiated.

2.6 Other paradigms

The goal of the following section is to mention the paradigms that have been taken as inspiration and that can also benefits from the work presented in the dissertation.

2.6.1 Public resource computing

Public resource computing (also known as global computing or peer-to-peer computing) uses unused resources located in the user's devices to do scientific supercomputing. It enables previously infeasible research and it also encourages public awareness of their results.

The first system supporting such paradigm has been developed in 1996 in the Great Internet Mersenne Prime Search project [8] with the goal to find the as large prime number as possible. Many other projects followed the trend, the most notable one was SETU@Home [9] carried out in 1999 with the goal in helping the SETI project finding extraterrestrial life by analyzing a multitude of radio frequencies coming from space.

2.6.2 Elastic computing

In their seminar work Herbst et al. [2013] defines elasticity as " .... degree to which a system is able to adapt to workload changes by provisioning and de-

[7]http://uprise-iot.supsi.ch/
[8]http://www.mersenne.org/
[9]https://setiathome.berkeley.edu/

provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible."

There are different architectural solutions to implement elasticity that can work on different levels. The key element is a resource that can be scaled up or down. In case the resource is a Virtual Machine elasticity can scale as a function of RAM and CPU cores assigned to it. In other domains a resource can be a physical node or a single unit like a CPU core or the RAM consumption.

A notable Elastic Computing implementation is the Amazon's Elastic Computing Cloud (EC2) [10] that is the core part of the Amazon's Cloud Computing Platform. EC2 is used as cloud instance solution this dissertation.

2.6.3 Internet of Things and Fog Computing

The term Internet of Things (IoT), was first introduced by Ashton [2009]. This new paradigm enables novel applications that extend the traditional Ubiquitous/Pervasive computing scenarios.

According to Atzori et al. [2010] there are three main IoT usage scenarios; they are illustrated in Figure 2.3 and they can be defined as:

- *Things Oriented:* where the network is constituted with very simple devices (i.e. RFID tags) and the main goal is to improve object visibility (i.e. the traceability of an object and the awareness of its status, current location, etc.).

- *Internet-oriented visions*: it is essentially the vision provided by the IPSO (IP for Smart Objects) alliance [11] and consists in using Internet Protocol as the main technology to interconnect things (i.e. in 6LowPAN group [12] they provide a standard for IPv6 over Low Power Area Network).

- *Semantic-Oriented vision*: the key idea behind this approach is to promote the usage of semantic technologies to describe and organize the information generated by IoT. The necessity of this organization may be dictated by the fact that the number of devices involved in the IoT may become extremely high thus making it really challenging to access the necessary piece of information.

[10]https://aws.amazon.com/ec2/
[11]http://www.ipso-alliance.org/
[12]https://datatracker.ietf.org/wg/6lowpan/documents/

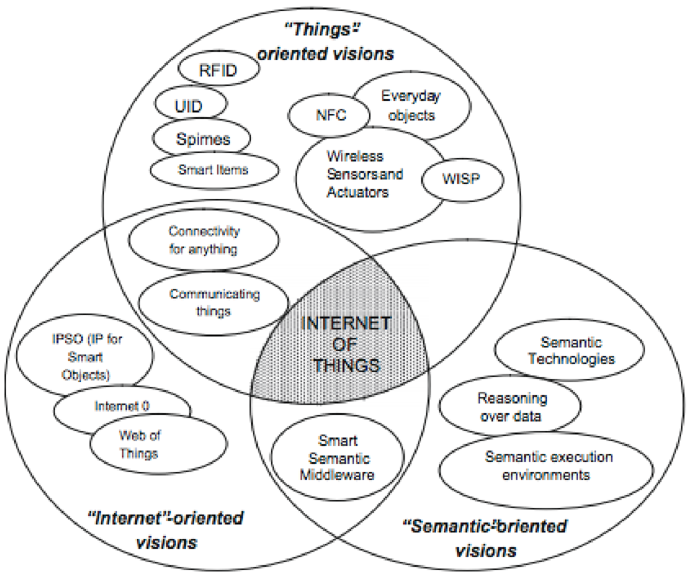

"Things"-oriented visions
RFID
UID
Spimes
Smart Items
NFC
Everyday objects
Wireless Sensors and Actuators
WISP
Connectivity for anything
Communicating things
INTERNET OF THINGS
IPSO (IP for Smart Objects)
Internet 0
Web of Things
Smart Semantic Middleware
Semantic Technologies
Reasoning over data
Semantic execution environments
"Internet"-oriented visions
"Semantic"-oriented visions

Figure 2.3. Internet of Things paradigm as a result of the convergence of different visions. (Courtesy of Atzori et. al.)

Integration between the cloud and the IoT has proven to be really beneficial. Botta et al. [2014] highlights several advantages in this integration between various domains (e.g. healthcare or smart cities), but the key idea is that IoT devices can take advantage of the capabilities that cloud computing can offer (e.g. to compensate the technological constraints).

The integration of the IoT with the cloud creates a new paradigm that has been named *Fog Computing* 8Bonomi et al. [2012]). It can be defined as a virtualized platform that provides computing, storage and networking service between devices located in the edge (things) and the Cloud. Fog Computing has been designed to address applications that tolerate high latencies. The main features of fog computing are:

- *Location awareness and low latency capabilities:* rich services shall be supported at the edge of the network.
- *Geographical distribution:* applications are distributed across different geographical locations.
- *Support for Mobility:* host identification and location must be decoupled.
- *Heterogeneity of Resources:* since devices are located into "things" there is an infinite kind of devices that can be part of the fog.

- *Interoperability:* in order to provide a given service previously mentioned requires interoperability among different providers.

It is important to clarify the difference between fog computing and opportunistic computing. They both consider mobility as an element. In opportunistic computing, mobility augments the reachability of the devices, while in fog computing it is a problem to be addressed through unique identification. The reason is that the fog still relies on the cloud to accessing the desired service while in opportunistic computing services settle for the closest available device.

2.7 Conclusions

This section has illustrated the principles that introduce the solution presented in the dissertation. It has started by introducing the concept of opportunistic computing that is a communication paradigm where services can be accessed opportunistically among devices located in the closest proximity. The FP7-SCAMPI Project has deeply investigated this paradigm and has produced a ready-to-use framework that will be the core of the following dissertation.

Moving further, computation offloading techniques have been introduced. The core idea of these techniques is to move part of the computation into a different device (usually into the cloud) to reduce the runtime and save energy.

The chapter concludes by introducing a set of complementary paradigms that may take advantage of the work proposed in the dissertation. The first is named public resource computing and consists in using the unused resources in personal devices to perform scientific calculation. The second one is elastic computing and consists in allocating resources and moving computation dynamically depending on the application needs. The section concludes by introducing the internet of things and fog computing; the first is a paradigm to add connectivity to everyday objects and the second is a solution to merge the IoT and cloud computing to offer high computation capabilities to the edge of the network.

The dissertation continues by illustrating a new paradigm named AnyRun Computing, a novel solution to augment the performance of smart devices by selectively offloading the computation in different hosts.

Chapter 3

AnyRun Computing

3.1 AnyRun Problem Definition and Modelling

Nowadays, applications are no longer tied to a single device. The cloud has allowed novel application paradigms to enhance the performance and the quality of the user's experience. The reliability of cloud-based solutions is out of discussion but there are situations where a given app cannot rely on it. Those situations require a different approach that is able to leverage different resources (e.g. located in the proximity). In this chapter I propose a paradigm that approaches this problem, named anyrun Computing. In conjunction I present the anyrun computing framework (ARC), a software solution specifically created to work in the anyrun scenarios. The chapter starts with the motivations behind the anyrun Computing paradigm. After, a solution to the host-selection problem is presented. The chapter concludes by illustrating the main architectural characteristics of the ARC problem.

3.1.1 Motivation

Nowadays, the access of high-performance systems anytime and anywhere have drastically changed the way people interact with their personal smart devices. This novel paradigm (namely cloud or cloudlet) makes smart devices able to perform computationally prohibitive tasks thanks to the offloading of computations. This service is made available thanks to recent advances in communication technologies. However, there are situations where high performance systems cannot be used. The following use cases present several examples:

- **Latency times:** in some circumstances, even high-speed mobile networks are not fast enough to offer a sensible gain when performing the offloading.

This is due to the amount of data needed to be transferred back and forth during the offloading process that in some cases might be huge.

- **Roaming:** probably one of the biggest sources of extra costs in the current mobile plans are the roaming costs. Usually, in such kind of situations, costs are exacerbated due to the usage of a mobile operator in a different country; this translates into a double-cost, one for the foreign operator and one for the national one. Even though in some cases there are special fares (e.g. E.U. Roaming convention[1]) such costs continue to be prohibitive for most of the users.

- **Mobile plan costs:** the majority of mobile plans usually tend to offer only a certain amount of free data. One may argue that there are unlimited plans; it is true that they offer unlimited data but it is common that they slow down the connectivity (usually down to EDGE speed) after a predefined amount of data has been used.

- **High speed network unreachable and overloaded:** the high-speed mobile connectivity is not always available. It depends on the network coverage that a given operator offers and to the morphology of the terrain where the network is located. (Figure 3.1 shows the 4G network coverage of one of the main Phone Operator in Switzerland Sunrise AG).

The previously mentioned limitations only apply to the cloud. With *cloudlet* devices are no longer restricted to WAN technology limitations because the hardware is located in close proximity. However, the cloudlet approach has a huge limitation due to the fact that hardware must be pre-deployed and known a priori; thus the cost to give full Cloudlet coverage is definitely prohibitive.

To overcome the previously mentioned limitations I propose the *anyrun computing* paradigm that can be described as follow:

Rather than assuming the existence of a dedicated piece of higher-end hardware and being rigidly tied to it, with a device anyrun may elect to use any piece of hardware (locally or remotely accessible) that can do the job better it can (this concept will be clarified later in the dissertation).

Figure 3.2 shows the dimensions where anyrun operates compared to traditional paradigms. The key idea is to extend the concept of **opportunism**, that

[1]https://ec.europa.eu/digital-single-market/en/roaming-tariffs

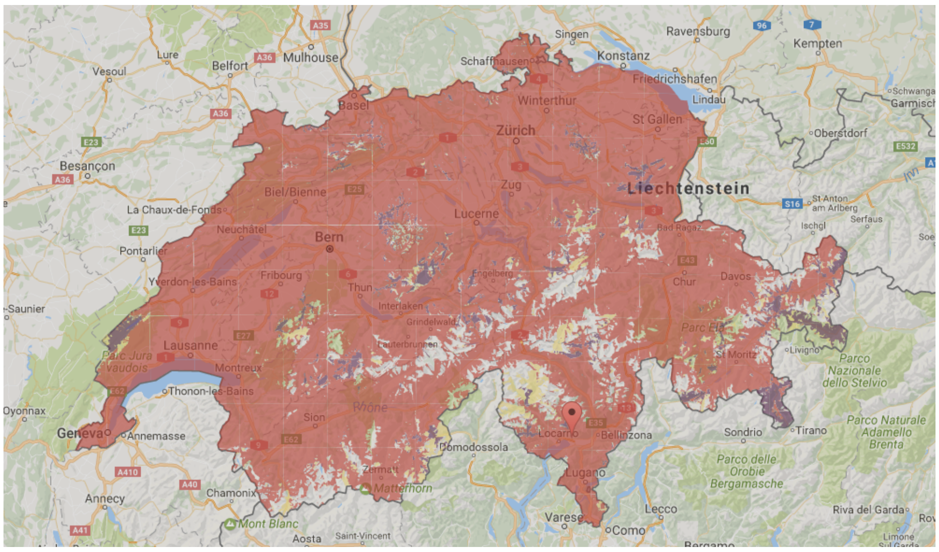


Figure 3.1. Sunrise 4G network coverage in Switzerland (courtesy of Sunrise AG), we can clearly identify several areas without 4G coverage, consequently, many services that rely on computation offloading may perform poorly.

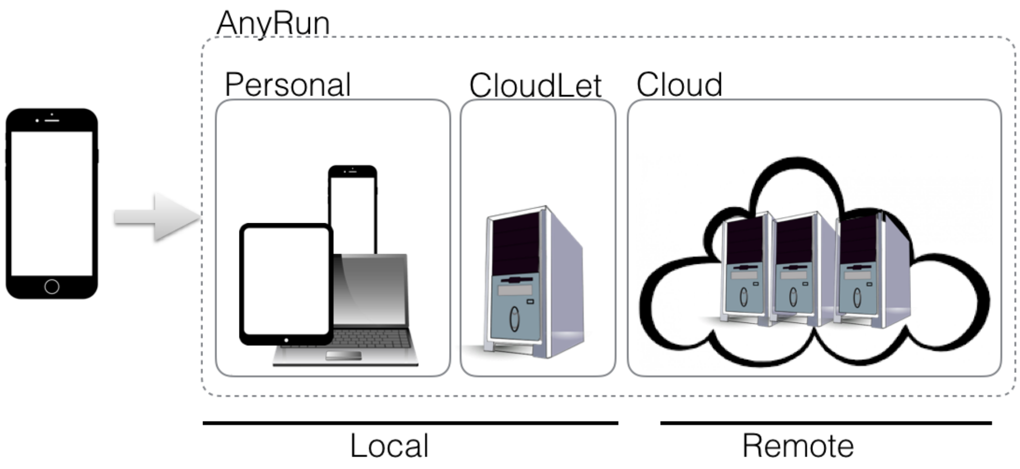


Figure 3.2. AnyRun dimensions compared to cloud and cloudlet.

in this case is translated as the possibility to choose the best option to perform the offloading between Cloud, Cloudlet and any other devices that are locally available down to the the local device (in case there are no other options that provide a sensible gain).

This section continues by providing a detailed definition of the anyrun paradigm and of the key challenges that must be addressed. After the ARC framework is described, ARC is the first framework that approaches the anyrun computing paradigm.

3.1.2 AnyRun Problem Definition

The first action that must be performed to characterize the anyrun paradigm consists in the definition of the key actors and their characteristics. Thus, following the scheme in Figure 3.2, the key actors are:

- *Personal smart devices:* portable and small devices (i.e. smartphone, tablet, ...). Due to the required portability, those devices usually have a scarce set of resources (in terms of computational capabilities, memory, and battery) compared to their desktop counterpart.
- *Personal Stationary devices:* they move rarely or not at all (i.e. Desktop computer or laptop). Some of them are powered by a battery but, compared to smart devices, the power source lasts longer.
- *Local and remote shared devices (cloud/cloudlet):* they are devices that do not move at all. The cloud is deployed in a remote location and accessible through WANs. The cloudlet is only accessible if the user is physically in its proximity. The core difference between remote devices and stationary devices is that the last is directly accessible by users while remote devices offer a set of services accessible through the network.
- *Seeker Device:* is the device that has the computational needs and is willing to find the best way to satisfy them. Usually, it is a smart device; the explanation is in the fact that those devices offer a limited set of resources.

3.1.3 AnyRun analytical definition

Figure 3.3 shows the various activities that must be performed in order to have a successful offloading. When the local host has the need to offloading the computation it performs the following activities:

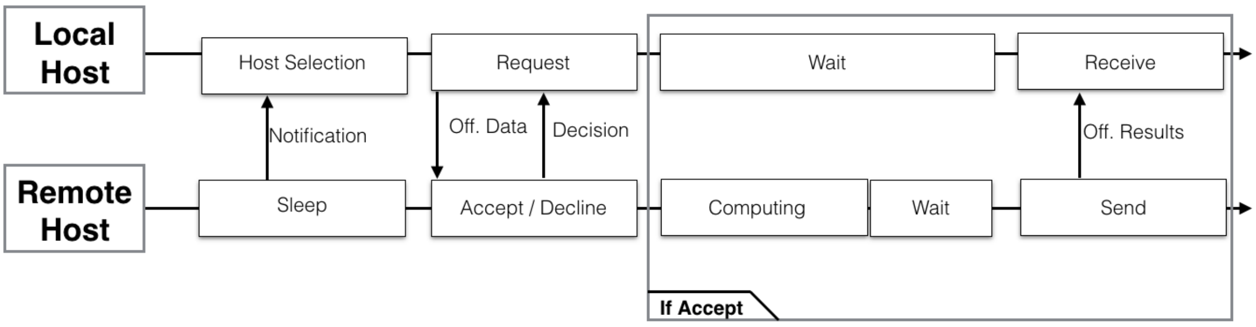


Figure 3.3. Sequence diagram that illustrates the various steps performed by two hosts when the offloading process is started.

1. Select an adequate remote host: in anyrun, selecting the wrong device may lead to extra costs (for instance the remote device may incur in the problem to send back the result of the computation, thus the local host must recover the job from the beginning).

2. Send the information needed to perform the computation: at this stage, the remote host has been selected, thus the local host sends the data and eventual the instruction needed to satisfy the request.

3. Sleep: the local host goes in sleep until the remote host sends back the results of the computation. Usually, timeouts are used to avoid infinite sleep.

4. Download the results: the local host receives the results of the offloading.

The remote host instead behaves as follows:

1. Accept or Decline: depending on the current context (e.g. status of the local resources) the remote host may decline the offloading request.

2. In case it accepts, it performs the computation.

3. It might happen that the two hosts are not in contact when the remote host ends the computation. In this situation, the remote host should wait until the two hosts return in contact again. Timeouts are used to avoid an unlimited wait.

4. Sends back the results.

It is important to point out that the connectivity between the two hosts may be lost due to users mobility if the devices are in the first two categories presented before (Smart Devices and Stationary Devices) but it may also be lost due to network vagaries in the third case (Remote Devices).

Another open point that arises is: why should a device which happens to be located in the proximity of a seeker and opportunistically accessed (thus without any form of infrastructure that the users usually pay for) be willing to take care of the computation? To answer this question I envision a set of use cases:

- The user owns the devices and it uses ARC between them to augment the performances of its smart devices (for instance to augment the lifetime of IoT devices).
- The users who own the devices know each other (personally, or belong to the same community, or ...), thus one of them allows the other to use its resources.
- The device who provide the computation is offered as a service in a specific location; for instance, a restaurant or a public transportation system may offer the offloading of computation as motivation mechanism.

The resources that a device is willing to offer depend on the device characteristics, the relationship between the owners of the devices, and the current load of the device.

Another approach may be to use incentive-based scheme like the solution presented by Chatzopoulos et al. [2016], but it poses severe security threats (as mentioned in Section 2). For this reason, I specifically avoided to use them as use-cases, but they will be part of my future studies.

Moving further, following the definition of Opportunistic Computing proposed by Ferrari et al. [2012] I can characterize anyrun computing with the following elements:

- N_t is the set that contains all the instances that are available to perform the computation at a given time t. Each $n \in N$ can be one of the devices presented before; it could be Cloud, Cloudlet or any other device that is willing to cooperate. The **local device** is a member of N_t. Depending on the time t, N_t changes its content because of the context changes (e.g. the network conditions change or different devices are available in the proximity).
- R is the set of all the offloading request from a given device. An application that needs to offload part of its computation must encapsulate the request into a request r. Each request contains:

Function	**Description**
$E_s(T_\delta(r,n)$	is the energy spent waiting for a good provider is accessibile.
$E_t(T_u(r,n))$	is the energy required to upload the data to the chosen provider.
$E_s(T_c(r,n))$	is the energy spent while waiting (sleep state) until the remote device has accomplished the computation .
$E_s(T_\omega(r,n))$	is the energy spent while waiting (sleep state) for the remote device to reach back the seeker.
$E_t(T_d(r,n))$	is the energy spent while downloading the results of the computation from the provider.
$E(r,n)$	is the overall energy spent during the offloading process.

Table 3.1. Characterization of the energy consumed considering the activity performed and the time required to accomplish it.

- The Information concerning the portion of the application that must be executed (if needed the software instructions can be shared)
- The data that the application must use to perform the job (usually this is a dump from the heap space).

Following the previously mentioned steps we can define the following function over time (we assume that the time T follows $T \in [0, ..., \inf]$):

- $T_\delta(r,n)$ is function that define the time to wait until a good provider (provider that is able to satisfy the request) is accessibile for a given request r executed on $n \in N_t$.
- $T_u(r,n)$ is function that define the time required to upload the data to the selected provider for a given request r executed on $n \in N_t$.
- $T_c(r,n)$ is function that define the time of the remote computation for a given for a given request r executed on $n \in N_t$.
- $T_\omega(r,n)$ is function that define the delay until the provider reach back the seeker for a givenfor a given request r executed on $n \in N_t$.
- $T_d(r,n)$ is function that define the time required to download the result of the computation for a givenfor a given request r executed on $n \in N_t$.

Moreover, we can characterize the energy spent in the various offloading activities as follow:

- E_s is a function that retrieves the energy consumed when sleep.
- E_t is a function that retrieves the energy consumed when transmitting or receiving.
- E_c is a function that retrieves the energy consumed when performing the computation.

We can therefore compute the overall energy consumption of the Offloading process as follow (a clarification for each elements is provided in Table 3.1).

$$E(r,n) = E_s(T_\delta(r,n)) + E_t(T_u(r,n)) + E_s(T_c(r,n)) + E_s(T_\omega(r,n)) + E_t(T_d(r,n)) \tag{3.1}$$

In case n is the local device we can shorten it as follow:

$$E(r,n) = E_c(T_c(r,n)) \tag{3.2}$$

All the other elements are zero.

AnyRun Problem Definition: Finally, we can define the anyrun Computing problem as follow:

$$\min E(r,n) \mid r \in R \text{ and } n \in N_t \tag{3.3}$$

That results in the selection of the best host n (local or remote) for each request r at a given time t such that it minimizes the energy consumed by the local device. This definition explains the "better" used in the anyrun definition at the beginning of the chapter.

Unfortunately, the accuracy of on-board energy measurement on smart devices is measured with a granularity too high to be effective in anyrun according to Ferrari et al. [2015b]. Luckily, we can use time as a proxy to indicate the energy consumption. In fact, there is a strict correlation between time spent doing a given job with the energy consumed in this job. Figure 3.4 illustrates the variation among different executions of the each energy-related scenario. The main finding here is that there are very few variations among different executions, thus since all the experiments have the same time granularity (1 second of activity) we can argue that it is reasonable to characterize the energy as a function of time and activity.

To this extent, the total time needed to perform the offloading can be computed as:

$$T(r,n) = T_\delta(r,n) + T_u(r,n) + T_c(r,n) + T_\omega(r,n) + T_d(r,n) \tag{3.4}$$

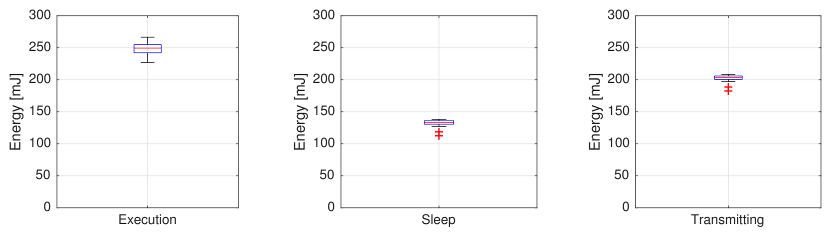


Figure 3.4. This figure illustrates three Boxplots that show the energy consumed for each energy-scenarios presented in the dissertation. The measurements are taken into a Samsung Nexus 3 Smartphone

If the n is the local device the previous expression si shorten as follow:

$$T(qrn) = T_c(r,n) \tag{3.5}$$

And the objective function became:

$$\min T(r,n) \mid r \in R \text{ and } n \in N_t \tag{3.6}$$

Given the dynamic and distributed nature of the anyrun, a deterministic approach cannot be used; in the remaining part of the dissertation I propose ARC a dynamic framework that adapt the offloading strategy considering the current context and learning from past experiences.

Note that anyrun does not ensure global fairness because it would require a global view that could be maintained only with centralized solutions. The approach draws inspiration from swarm-based algorithms where each device has a limited view (usually limited to its neighbors) and local optima are sought.

3.1.4 Remote Host Selection for Anyrun

Remote host selection is the most challenging and critical activity in anyrun. The method I am proposing to select the best remote host is based on the NaïveBayes classifier.

The key motivation that dictated the selection of that model between all the possible solutions presented in literature is that the computational costs may negatively impact the performances of the offloading process.

A preliminary analysis of the cost of different machine learning solutions is presented by Ferrari et al. [2015a] where the authors compare accuracy vs. energy consumption of three machine learning strategies applied to Activity Recognition. The authors have empirically demonstrated that the Naive Bayes has a small impact on the energy footprint of the device but they provide a decent classification accuracy.

Back to the anyrun computing timing function described before:

$$T(r,n) = T_\delta(r,n) + T_u(r,n) + T_c(r,n) + T_\omega(r,n) + T_d(r,n) \tag{3.7}$$

The selection of the provider directly impact $T_\delta(r,n)$, and the evaluation repeat its costs for each device that is evaluated (from cloud to locally available device) it is reasonable to assume that in many circumstances there will be a multitude of devices to evaluate, thus a solution that impact as less as possibile is needed.

I categorize Naive Bayes as follow: let Y represent a classification variable that defines mutually exclusive and collectively exhaustive classes, and let y be the value of Y. A classifier is a function that assigns a class label to a set of attributes $E = \{E_1..E_{N-1}\}$ (with $N \in \mathbb{N}$). If the attributes E_i are assumed to be independent, then I have a Naïve Bayesian model (Zhang [2004]), for which

$$P(E = e, Y = y) = \prod_{i=1}^{N} P(E_i = e_i, Y = y). \tag{3.8}$$

In our case, as the goal is to determine whether a portion of an application a should be offloaded, our classification variable Y may take on either one of two mutually exclusive and collectively exhaustive values: *offload* or *run locally*. I focus on $O =$ *offload*, and $P(Y = \textit{offload}, E = e)$ is to be interpreted as the probability that offloading is advantageous in the dimensions of interest given that the realization e of the set of attributes E has been observed. Since the two values that I assume Y may take on are mutually exclusive and collectively exhaustive, $P(Y = \textit{run locally}, E) = 1 - P(Y = \textit{offload}, E)$.

By definition, the prior probability must be independent from E (Berry [1996]). I elect to compute the prior probability based on the state of the local node, while I populate E with attributes of the remote node I wish to offload code to. The first step is the computation of the prior probability that offloading is advantageous in the dimensions of interest based on the state of the local node. For each kth run of the request r, *either local or remote*, ARC estimates the *local* footprint as a function of the dimensions of interest (for more details, please refer to Section 3.2.3).

Let $l < k$ represents the current count of local runs; a moving average of the cost of the L most recent *local* runs is maintained by the offloading block and denoted as Λ_r, with $L = \min(l, L_{max})$. Each request r must be run locally at

Symbol	Description
r	incapsulate the portion of the application that must be executed in a request
r_k	Current run of r
L_{max}	Maximum size of the most recent executions.
L	Number of most recent runs considered.
Λ_a	Average estimate of the cost of running a.
$\Lambda_{r,k}$	Costs of the kth runs of a.
E	Most up-to-date observation.
λ	Probability Threshold.
T_{max}	Maximum allowed latency.

Table 3.2. List of symbols that describe the environment where ARC works.

least once to initialize Λ_r, which serves as our estimate of the average cost of running a locally under the recent conditions at the local node. In principle, it suffices to run r locally just once to initialize Λ_r. It is also possible to initialize Λ_r based on estimates of its CPU, RAM, time, and energy footprint. However, in a realistic scenario with unstable connectivitry, it is not always possible to offload a, so it is reasonable to expect that there will be a number of local runs and to leverage them to get an up-to-date estimate of Λ_r. Because the prior probability must be independent from the set of selected attributes (Berry [1996]), E is populated with remote node attributes. The set of attributes E is advertised by each remote device in its broadcast control messages (ARC presupposes the existence of a beacon-based neighbor discovery protocol). For the details of the attribute selection in my implementation, please refer to Section 3.2.3).

Whenever an offloading decision has to be made, ARC uses the most up-to-date observation of E, say $E_o \triangleq \{e_1 \dots e_N\}$, to compute the Bayesian inference, *i.e.*, the posterior probability that the observed set of attributes can be classified as c:

$$P(Y = y, E = E_o) = P(Y = y) \prod_{i=1}^{N} \frac{P(E_i = e_i, Y = y)}{P(E_i = e_i)} \tag{3.9}$$

For $C = \mathit{offload}$, Equation (3.9) represents the probability that offloading is advantageous under the conditions represented by the attribute observation E_o. To compute this probability, for each realization $e_i \in E_o$ of the random variable $E_i \in E$, I need to compute $P(E_i = e_i, Y = y)$ and $P(E_i = e_i)$. $P(E_i = e_i, Y = \mathit{offload})$ represents the probability that the ith attribute within E takes on the value e_i given that offloading is convenient. This probability is estimated based on the history of the runs of a given request r as the fraction of the runs with $\Lambda_{r,k} \leq \Lambda_r$ for which $E_i = e_i$ is observed. Likewise, $P(E_i = e_i)$ is estimated as the fraction of the runs of m for which $E_i = e_i$ is observed.

Having gauged the probability that offloading is advantageous in the dimensions of interest, I set a threshold $\lambda \in [0, 1]$ that captures how conservative I wish to be with the use of local resources. For instance, if $\lambda = 1$, everything will be run locally, while if $\lambda = 0$, every r will be offloaded. The specific value of the threshold λ must be set by the application whose requests are to be offloaded or run locally. For instance, if the application doesn't have particularly tight timing requirements, it is acceptable for the application to set λ to a lower value; conversely, the tighter the timing constraints of the application, the higher the value it ought to set λ to.

3.2 The ARC - AnyRun Computing Framework

To empirically study the anyrun Computing solution I developed ARC - the anyrun computing framework. The goal of ARC is to dynamically decide whether any kth run ($k \in \mathbb{N}$) of a method m that is part of a given Application A that is under execution in given device $d \in D$ should be offloaded or run locally (m is ten incapsulate in r following the definition from Section 3.1.3):

3.2.1 ARC Architecture

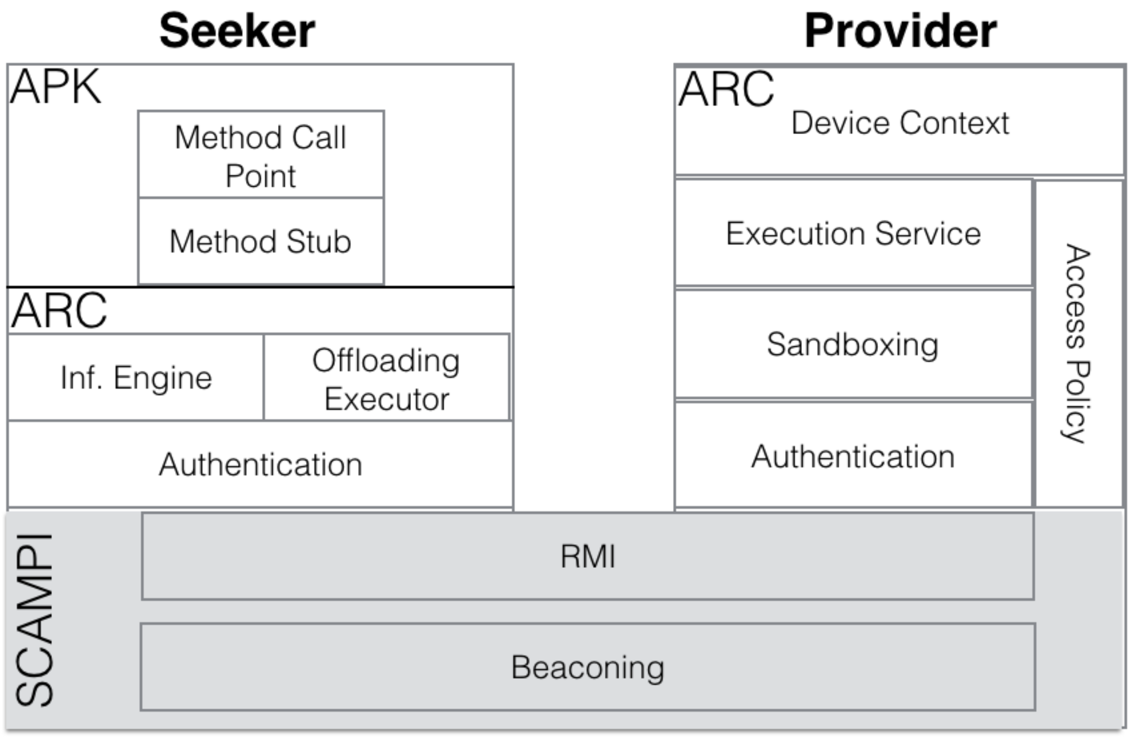


Figure 3.5. The components of the ARC architecture.

In the next subsection I will first deeply describe the ARC architecture, ARC inference engine, and ARC security mechanisms. The next section will discuss the tool I have built to accurately evaluated the paradigm and then I will conclude the dissertation by illustrating and explaining the deep evaluation I have to perform on top of ARC.

The system is built on top of the SCAMPI opportunistic computing framework Conti et al. [2010a]; Pitkänen et al. [2012] developed within the FP7 EC project SCAMPI (Service Platform for Socially Aware Mobile and Pervasive Computing)[2]. The key issue tackled in SCAMPI is the provision of services on top of the opportunistic computing paradigm; recent work in this space are offered from Hyytiä et al. [2015], and Li et al. [2015]; Liu et al. [2014]. SCAMPI offers a set of API to recognize devices in the proximity with a set of different technologies (from legacy WIFI to Bluetooth) and allows users to access services on such devices. Services may be run on multiple devices in parallel (multicast access) or on a single device (unicast). This work extends SCAMPI with an inference layers targeted to computational offloading.

There are two key actors that play a role in ARC, they are respectively:

- *ARC Seeker:* is the devices that is in needs to offload the computation, it generates the request (r in anyrun definition)
- *ARC Provider:* is a device that is willing to accept to take care of the Seeker's computation (that can be either cloud, cloudlet, or any other device in the closest proximity) .

Figure 3.5 shows the overall architecture of the two actors and the conjunction between them that is the SCAMPI Layer. The rest of the section is dedicated to deeply explaining both actors.

3.2.2 ARC - Seeker

The left side fo the Figure 3.5 illustrates the behavior of the main components of ARC Seeker. The entry point of the ARC framework is the *method stub*, which is a predefined class that allows developers to subscribe to the framework and access it easily. The stub uses the *Inference Engine* to assess the probability that offloading is advantageous compared to local execution in the dimensions of interest (CPU and RAM usage, execution time, and energy consumption). Depending on the response from the Inference Engine, the stub takes the appropriate course of action.

In case local execution is chosen, a dedicated architectural block (the *Local Execution Manager* in Figure 3.5) uses Java reflection[3]to call the method. In case

[2]http://www.ict-scampi.eu/

[3]I use Java as programming language for the implementation. Though the use of Java is not a fundamental requirement, certain features of the Java language (*i.e.*, reflection and serialization)

remote execution is chosen, a dedicated architectural block (the *Remote Execution Manager* in Figure 3.5) serializes the part of the Java heap reachable from the method, sends it to the remote machine through the SCAMPI framework , and awaits a response. A timeout is set based on the local execution time (I assume that each method is run locally at least once). If the remote execution exceeds the timeout, the Remote Execution Manager calls the Local Execution Manager so the method can be run locally.

The Seeker may decide for three execution strategies:

- *Offload to cloud/cloudlet*:
- *Offload opportunistically*
- *Local execution*

To this extend the first tentative is to check if the offload to cloud/cloudlet is available. To this extent to detect if the Cloud is available ARC first samples for WANs connection and if there is one periodically it checks if the cloud service is reachable. The continuous sampling is necessary because checking for the existence when is needed it add an extra delay that may lead to failure in offloading.

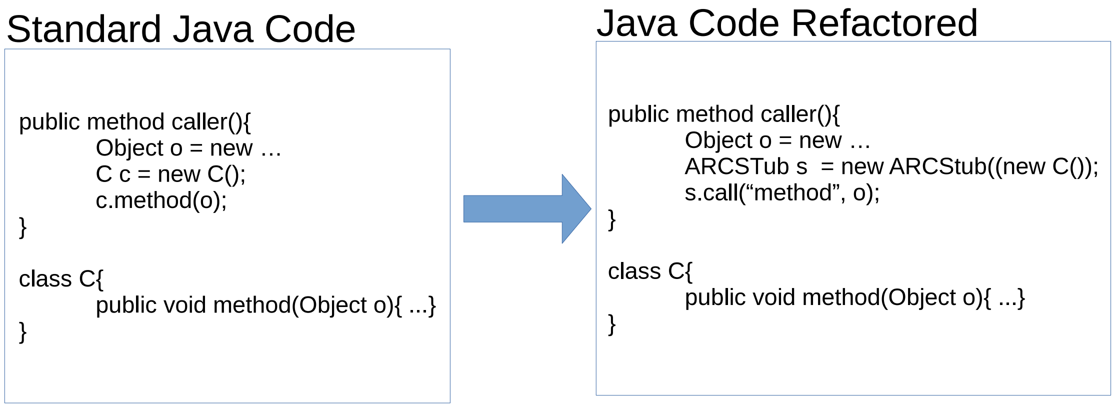
As said before, Cloudlet solutions requires being known a priory thus we can assume that ARC knows if a Cloudlet is reachable depending on its position or LAN identification.

To understand if the cloud or cloudled are able to take care of the ofloading, ARC executes the procedure at least once and check the outcome. if the resulting costs are larger than the cost of executing the method locally then di cloud is discarded for future selection on that specific method call.

if both cloud and cloudlet cannot be used ARC starts looking for opportunistic solutions via the SCAMPI framework.

The SCAMPI framework offers a remote method invocation (RMI) service that enables devices to request remote routine execution from each other. In ARC, the Remote Execution Manager uses SCAMPI's RMI jointly with an *Execution Service Manager* on the remote device. The Execution Service Manager is tasked with running the desired method on the remote device and provide the related results, also by way of SCAMPI's RMI.
I now delve into the details of the Inference Engine, which contains most of the intelligence in the system.

make it particularly suitable for the implementation of computation offloading schemes. A viable alternative would be the use of Microsoft's .NET Framework, employed by Cuervo et al. [2010]. Further details about the Java implementation are provided in Section 3.2.3



```
public method caller(){
        Object o = new …
        C c = new C();
        c.method(o);
}

class C{
        public void method(Object o){ ...}
}
```

```
public method caller(){
        Object o = new …
        ARCSTub s  = new ARCStub((new C());
        s.call(“method”, o);
}

class C{
        public void method(Object o){ ...}
}
```

Figure 3.6. ARC Java Refactoring where the method stub is included in the application control flow.

The Inference Engine receives a rich set of information about the available remote devices from SCAMPI (for more details, please refer to Section 3.2.3)and employs the information to help the stub decide whether offloading is advantageous compared to local execution, *i.e.*, whether it can reduce the local footprint compared to local execution in the dimensions of interest (CPU and RAM usage, execution time, and energy consumption). The output of the Inference Engine is an empirical estimate of the probability that offloading is advantageous in the aforementioned sense.

3.2.3 ARC Seeker - Implementation Details

My ARC implementation is written in Java and works in both the Dalvik virtual machine (Ehringer [2010]) on Android-based[4] devices and the Oracle's HotSpot virtual machine [5] used in desktop environments. Portability to other Java-compatible virtual machine may be possible depending on the implementation of the Java memory model. I use the standard Java Object serialization to exchange class instances at runtime and Java reflection to dynamically choose the appropriate method at runtime. Object mobility is allowed inside the previously cited virtual machines because they share the same object representation in the memory space.

In ARC, similarly to MAUI, if a developer wishes to offload a portion of her application, she simply implement a ARC class stub that will call the offloadable

[4]Due to changes in the virtual machine Android version should be 4.0 (Ice Cream Sandwich) or newer. The most recent version I have tested is 5.1 (Lollipop).

[5]http://www.oracle.com/technetwork/java/javase/ tech/index-jsp-136373.html

code dynamically through the framework.

In ARC I implemented a Java *refactoring* technique (that could be easily automatized in the majority of Java IDE, e.g. Eclipse[6]) to allow developers to easily include the system in their application. Figure 3.6 shows the standard refactoring procedure in ARC:

- At first it replaces the initialization of the classes I want to offload with the ARCSTub class (let *C* be the class I want to use in offloading).
- It initializes the *ARCStub* class with *C* as its constructor argument.
- It replaces each method call in *C* with the call to the *ARCStub.call* method and it passes the method name and the parameters as arguments. If the user doesn't want a given method to be offloaded, the "*callLocal*" function can be called on *ARCStub* to force local execution.

I am currently implementing and automatizing this refactoring technique in the new Android IDE (Android Studio[7]).

A standard profiler is run by ARC upon the execution of each offloadable method (either if run locally or remotely).

The ARC profiler keeps track of several variables ranging from the execution time to the battery drain. In the implementation, ARC estimates the local footprint as a function of the dimensions of interest (execution time and energy). *E* is populated with remote node attributes and is advertised by each remote device in its broadcast control messages (ARC presupposes the existence of a beacon-based neighbor discovery protocol).

ARC Seeker - Inference Engine

This component implements the solution proposed in Section 3.1.4. As said before, ARC works at method level for this reason the variable a is now replaced with m for methods. In the implementation, I employ the following attributes for any method m:

- the CPU and RAM usage and the battery usage on the remote device;
- the mean duration of the past contacts between the local device and the remote device (computed over a sliding window);

[6]http://www.eclipse.org

[7]https://developer.android.com/sdk/installing/studio.html

- the taxonomy of the remote device m is offloaded to (smartphone, tablet, laptop, or others);

Information about past contacts is used as an empirical estimate of the expected duration of the contact time between the local device and the remote device. The information about the remote device is passed to the *inference engine* jointly with the information about the local execution of the method and the local status of the system (in terms of RAM usage and CPU usage) as input parameters. The inference process then returns an estimate of the probability that offloading to the remote device will be advantageous in the dimensions of interest. I use this inferred probability to make the offloading decision as explained in Section 3.1.4. In the remainder of this Section, I set T_{max} for each method m to coincide with the mean local run time; however, in general, T_{max} can be freely set by the application.

In the scenario where it is possible to offload to multiple devices, ARC sends offloading request in parallel to all available devices that are inferred as possible good candidates. The requests are sent to all the good candidates in descending order of probability of success, starting from the one that offers the highest probability of success. I plan to refine this greedy approach in future work.

I also set a timeout S_{max} that captures the maximum latency value I can settle for.
The execution of method m is offloaded if the probability that offloading is advantageous exceeds or matches λ and is run locally otherwise. If the overall execution latency exceeds S_{max}, ARC defaults to local execution.

ARC continuously collects information about newly encountered devices that advertise their resources to assess whether they could be good offloading partners.

3.2.4 ARC Seeker - Code Mobility

Due to the dynamic nature of the environment, the assumption that Provider already owns the right set of instruction to process the data it receives during the offloading is simply impractical due to memory limitations. In fact, there is not limits of the number of applications and consequently to the line of code that can be remotely-execute in such paradigm. To address this problem I provide a utility in the ARC framework that allows a seeker to sent-out to the provider the instruction he needs to execute in conjunction with the data they must process.

The mobility of the code is something that has been studied for years, the first notable work on application partitioning for Java-based system is J-Orchestra by

Tilevich and Smaragdakis [2002], who propose an effective tool to partition Java applications to dynamically distribute the load onto different virtual machines. J-Orchestra can correctly partition almost any pure Java program, allowing objects to be placed on any machine, without the need to know how application objects access each other and Java system objects. It works at the Bytecode level, generating several different components that could be dynamically deployed on different machines if needed. J-Orchestra offers the basic offloading of computation through Remote Method Invocation (RMI) but it does not tackle the problem of mobile and heterogeneous devices.

Chen et al. [2012] explore the design of a system able to automatically partition Android applications without modifying the underlying system. Basically, the application extracts the android-services and redeploy them inside a "virtual-smartphone" in the cloud. They also provide a set of decision metrics do decide whether offload or not, the metrics has to be computed before and they consider the local execution costs vs. the transmission costs. This approach is clearly flawless, first because the network changes during time thus the metrics is not available, secondly because services are composed by a whole set of activity and many of them require accessing the local context (i.e. access to location) thus they cannot be remotely deployed.

Integration with ARC:

ARC has been developed for Google's Android [8] and Java-base platforms. The granularity of ARC offloading is at the method level, where any developer willing to use the framework has to call its methods through the ARC class-stub objects. Such objects decide whether the method should be run locally or remotely depending on the current and past situation in the network. The key points addressed by the solution are:

- the extraction of portable offloadable code;
- the insertion of the code in an Android Application Package (APK).

Offloadable code extraction

I start with a compiled Android application and use Dex2Jar[9] to extract the Dalvik/Android Runtime (ART) Bytecode (.dex files). I then convert it to Java Bytecode so I can use standard Bytecode manipulation tools that, nowadays, are

[8]http://www.android.com
[9]https://github.com/pxb1988/dex2jar

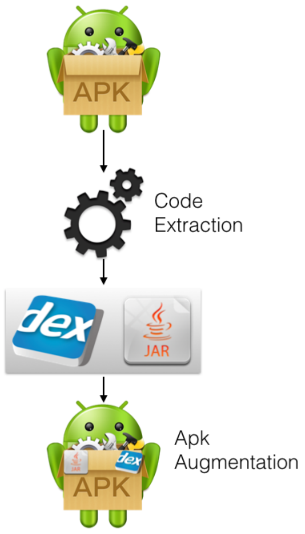


Figure 3.7. Steps of the code extraction process; first the APK is passed through the system and two distinct class containers are generated (JAR and DEX), then the two containers are injected into the APK.

more advanced and stable compare to Dalvik/ART Bytecode manipulators. In this work I use the Java Bytecode manipulation and analysis tool ASM [10] .

The rationale of the extraction is illustrated in Figure 3.7. The JAR contains a set of classes and each class has his own method.First of all we must find the class that implements the ARC Stub interface. This is the entry point of the offloading process. The output is a JAR file for each implementation of the stub that contains all the classes needed to run the code inside a Java virtual machine. In order to produce such jAR the system analyzes the Bytecode to find each internal reference. I check and extract each reachable class by recursively looking at:

- Superclasses
- Implemented interfaces
- Class Instantiations (at the method and class level)
- Calls to static classes

At the end of the process we obtain a JAR file for each task to be offloaded. At this point I use the Jar2Dex utility (part of the Dex2Jar library) to convert

[10]http://asm.ow2.org/

the Java Bytecode into the Dalvik/ART Bytecode and generate a new file, to be named classes.dex, that is injected into a new JAR. This last step is very important because the DexClassLoader utility offered by the Dalvik/ART platform only loads files named classes.dex inside JAR files.

APK and ARC Integration

Once the two files are generated, a developer that is willing to exploit this new feature in ARC must put both files inside her APK. I use the *assets* folder in the APK structure that is intentionally designed to contain extra files. Once the APK is extended, ARC automatically finds the code and spreads it as needed.

When a potential offloading target is encountered, the following steps are undertaken.

1. Check the device taxonomy (whether it is a computer or a mobile device) to determine whether a DEX file or a JAR file is to be employed (ARC learns this from its beaconing process).

2. Ask the offloading target if it already has the code it needs to run.

3. If the offloading target does not have the code, the code is provided.

4. The computational offloading strategy (ARC in our case) is carried out.

At the offloading target, the offloaded code is associated to a time to live counter to ensure its removal from memory after a pre-determined time.

Note that the overhead of making the code available to the remote device is ignored by ARC's decision-making process (to decide whether or not to offload) because it is a one-off transaction.

3.2.5 ARC - Provider

ARC Provider performs three main activities:

- It shares information concerning the status of its resources to neighbouring devices.
- It performs the remote computation.
- It receives and mange extra code from the remote device.

Information Sharing is performed within the SCAMPI framework. This is implement by sampling at a pre-defined interval the CPU status, virtual memory status and battery (if one is present otherwise it is indicated a fully charged). When a beacon is sent it also includes the most up-to-dated observation and the device taxonomy.

To perform the remote computation the device receive a set of Serialized Class instances, when they are loaded the fist action consist in checking if the Class is already present in the system by using the Java Reflection. If not the class has to be loaded, in case the Seeker has also sent the DexBytecode in the call it has to be loaded with DexClassLoader into the system.

The major risk there is that the Provider send malicious code able to steal sensible information, to solve this problem Sandboxing solutions are required mixed with Mocking techniques are Required.

3.2.6 ARC Provider - Sandboxing

Mocking is a standard software testing technique. Mock-objects are usually used in objected-oriented unit testing. The key idea is to use mock-object instead of real ones and for several reasons (e.g. to supply fictitious data Mackinnon et al. [2001]). The relationship between mocking and privacy is very simple: due to the fact that I can change the app-parameters I can also conceal the real data and thus ensure user privacy; this approach is presented in MockDroid from Beresford et al. [2011] who propose a modified version of Android that grants privacy by mocking the application behaviour. MockDroid s based on Android 2.2.1 where authors change the APIs at the system level.

I propose *MockingBird*, a novel solution that allows users to choose which context information can be shown to the application. MockingBird offers the possibility to decide which applications are to be mocked. MockingBird uses bytecode-manipulation techniques to detect the privacy-challenging instructions within an app and insert external instructions to preserve user privacy. It also uses a dedicated application to enable users to manage their context information to assemble their custom mocking solution.

Mockingbird is included in ARC Provider to guarantee isolation at the code shared during the offloading process

MockingBird, uses Dex2Jar [11] work with the Java Bytecode from an Android application and ASM (Bruneton et al. [2002]) (a bytecode manipulation tool) to modify and extend the application without the need for recompilation. ASM (Bruneton et al. [2002])allows us to clearly determine what information is used by an application and where it is used by extracting its bytecode, which is then modified to mock the application. This procedure offers flexibility and portability because it can be applied at the application level without modifying the operating system or the virtual machine. I also provide a tool to record real context that can be fed to the mocked applications, thus eliminating the need for random data in existing mocking tools. This is useful independently of the reason why I employ mocking: in the case of app testing, previously recorded data is close to the real-world data the app will encounter, while in the case of privacy preservation, previously recorded data will look like real-world data to the application, which will not be able to easily detect it is being mocked.

The context-related information managed in Mockingbird are:

- *Location* in the form of coarse location and fine grained location; I replace latitude and longitude.
- *Gyroscope*: I replace the rotation rates around the three axes.
- *Accelerometer*: I replace the acceleration values.
- *Barometric pressure:* I replace the pressure value.
- *WIFI:* I modify the connection status, the ESSID, and the signal strength values shown to the app; I always return an empty list to the app whenever it requests scanning information.
- *Mobile data:* I modify the connected cell tower, the network operator, and the signal strength values.

Other context-related data sources (e.g. calendar information or SMSs) are also considered; however, in these cases, I currently return randomly generated data (i.e. a random event or random strings).

[11]https://code.google.com/p/dex2jar/

There are several advantages in adopting Mmockingbird as sandboxing technique in anyrun computing. One may argue that solutions that work on Operating System level may be more effective, this is true but only in Cloud/Cloudlet domain where we have full access to the System. Furthermore, anyrun computing also works on Smart and Mobile devices where any device may be elected as Provider, consequently, we cannot pretend to have the access to all the OS. On the other hand, Mockingbird is particularly suited for this domain because it works at application level.

3.2.7 ARC - Authentication Mechanisms

By definition, Authentication is the process to identify another user or device to (in our scenario) share with them resources or services. In anyrun computing authentication is not a trivial part. To this extend the solution has to work in multiple domains with multiple conditions. Therefore, in ARC I propose a hybrid model that adapts itself depending on the provider taxonomy. The solution is currently under investigation in the CHIST-ERA 20015 UPRISE-IOT[12]project where the main goal is to provide security and privacy mechanisms considering the current mutation in device taxonomies and networking paradigm.

In ARC depending on the Provider's Taxonomy, different solutions are implemented:

Cloud/Cloudlet

in this scenario, the Seeker is known a priory thus traditional authentication mechanisms can be applied. in ARC I apply standard SSL techniques. The rationale is that The cloud/cloudlet behave as certification authority (CA), a certificate is therefore used as starting point; each certificate is pre-deployed in each cloud or cloudlet. At the end, the certificate it is used to secure the connection.

Opportunistic Domain:

removing the central authority requires different solutions. The one that is currently implemented in ARC takes inspiration by Bluetooth [13] authentication. Bluetooth[15] is a short-range radio technique that operates in a 2.4 GHz ISM (industrial scientific medic) band. The Bluetooth specification defines link level

[12]http://uprise-iot.supsi.ch/

[13] http://www.bluetooth.com

security mechanisms fo provide confidentiality, integrity, and authentication between devices. Services may use one of the following three security levels for authentication:

- *Level 3:* they require PIN exchange between devices.
- *Level 2:* (authentication only) thy use a fixed pin but users has still to confirm that they accept the connection.
- Level *1:* totally non-secure. Everybody can access without any request.

Devices that are authenticated are placed in the so-called "trusted list" and they can access all the services in that security level in the remote device.

Other approaches like reputation-based or the one proposed by Hossmann et al. [2011] in the SCAMPI Project are still possible and will be part of future studies within the CHIST-ERA 20015 UPRISE-IOT project [14].

3.3 Conclusions

This chapter has illustrated the analytical principles of anyrun computing, a possibile solution to the Host-Selection problem that is the key challenge in anyrun, and the architectural design of the ARC framework. In essence, anyrun computing can be used when a given application cannot rely on cloud-based solutions (several use-cases have been illustrated in the chapter), to overcome this limitation it exploits resources located in different position (e.g. local). Within this context, several problems may arise, first of all, the high mobility of devices and the difference in computation capabilities may compromise the offloading or sensible affect the gain. Challenges and problems in anyrun are firstly been enumerated and described in the chapter and then they are addressed in the ARC framework.

The dissertation continues by explaining the solutions I have implemented in order to been able to provide a detailed study on the anyrun computing solution. After, a deep investigation of ARC compared to the state of the art solution is presented.

[14]http://uprise-iot.supsi.ch/

Chapter 4

AnyRun Testing Environment

4.1 Introduction

The advent of smart devices has drastically changed the landscape where such IT technologies operate. Traditional devices were stationary (or with a very low degree of mobility) and usually were connected to a power outlet. Nowadays, devices have become smaller, thus they are more portable, to do this they are battery powered. This mutation in the physiognomy of the devices requires different approaches in order to measure the performances of new applications such as the ARC framework presented before.

The goal of this Chapter is to illustrate a new environment specifically created to measure AnyRun-like solutions. The environment is composed of two main elements:

- *Energy Profiling:* Energy profiling has been performed with success for decades, however, new smart devices have a set of features that require new solutions. The first problem is in low-granularity of reading frequencies of the battery on modern devices, and the second one is dynamic environment where smart devices operates (manly dictated by device mobility). A solution that has been used to test ARC is presented and evaluated with a Simple example in the next section.

- *Experiment Replication:* Offering a fair comparison between different solutions, the AnyRun domain requires the same environment replicated in different experiments. To approach these issues DroidLab is proposed; a testbed that offers realistic emulation of a dynamic environment, but it also offers full control over it.

The two components will be carefully described in the remaining in the chapter.

4.2 Producing Accurate Energy Profile

AnyRun solutions are mainly meant to increase the user's experiences. This can be expressed as the gain, in terms of time and energy consumed, a given application obtain when offloading. This leads to a clear need for a detailed understanding of the power consumption of a given app.

Due to the poor quality of smartphone internal battery monitoring unit, a set of freely available solutions that help developers in their work have been proposed in the scientific community. Many of them rely on smartphone hardware modelling to retrieve accurate power measurements according to Pathak et al. [2011], whereas others rely on external power monitors to do the same job (Balasubramanian et al. [2009]). On the software side there are several solutions to study the application behaviour, ranging from static code analysis to the analysis of the system calls (Yoon et al. [2012]; Li et al. [2013]).

In this section, the design and implementation of a Portable Open Source Energy Monitor (POEM) is presented. This is a novel solution to overcome the limitations of current approaches. Based on a portable powermeter, whose design is in turn based on Battor (Schulman et al. [2011]) and NEAT (Brouwers et al. [2014]), I extend the state of the art of application analysis with the energy annotation of the control flow down to the basic blocks, the call graph, and the Android API calls, been able at the end to find where the energy is spent inside a given app with the maximum possible accuracy and with the ability to isolate the Offloading section in our case. Given the power consumption is tied to the system status (for instance, whether the screen is lit or whether the radios are on), I also synchronized the energy annotation with a system activities logger. Finally, I offered a powerful and dynamic visualization able to easily isolate the interesting portioning of the application and analysed where energy is spent.

The rest of the Section is organized as follows: at first I provided more insights in the related work that is needed as background to understand the next subsection. After I llustrated the details of the design and implementation of POEM. I concluded the section by illustrating a set of simple examples to clarify the usage of POEM.

4.2.1 Related work

I first illustrated the state of the art on the retrieval of accurate power measurements in modern smartphones, after I presented an overview of the state of the art of the power consumption analysis of smartphone apps.

Energy measurements on smartphones

Power consumption in modern smartphones is generally measured with hardware-based or model-based techniques. Hardware-based methods are based on the internal battery unit on the smartphone or external measurement platforms such as the Monsoon power meter [1]. It is notorious that the internal battery unit on mobile devices is inaccurate and in the majority of cases works in a very high sampling rate. On the other hand, external power monitors offer high accuracy and low level sampling rates, but they require tethering the smartphone to a desk, thus keeping the user outside the testing loop and making it impossible to record usage patterns and environmental conditions. Model-based (Dong and Zhong [2011]; Jung et al. [2012]) techniques partly rely on the internal battery unit (BMU) in the smartphone, but refine the measurements by accurately modelling the internal hardware consumption. Though these techniques offer an acceptable level of accuracy, the complexity of modern smartphones makes it virtually impossible to model every single state according to McCullough et al. [2011]. Recently, solutions based on portable hardware have been proposed. The systems described by Schulman et al. [2011] and by Brouwers et al. [2014] provide external portable power monitoring tools based on a small custom board with memory and processing capabilities that are connected between the battery and the phone itself. All current flows between the battery and the smartphone is routed over a shunt resistor that allows the accurate measurement of the current drawn by using the Ohm's Law. All measurements need to be synchronized with the smartphone in order to detect which activities are responsible for a recorded energy pattern. Battor uses predetermined power consumption patterns and NEAT is physically connected to the phone buzzer, which is dedicated to synchronization (leading to minimum loss of the user experience). POEM provides a fully open source version of the portable energy monitor based on the Arduino[2] hardware platform and non-intrusive LED2LED-based (Giustiniano et al. [2012]) synchronization.

[1]http://www.msoon.com/LabEquipment/PowerMonitor,

[2]http://www.arduino.cc

Extracting application behaviour

Several solutions have been recently proposed to determine where the energy is spent inside an application. AppScope from Yoon et al. [2012] physically modifies the Android kernel by adding a logger that monitors software/hardware interactions and provides an estimation model to catch the energy consumption of the application. Despite its advantages, AppScope does not provide any information at the application level (e.g., which method is taking most of the energy). eCalc by Hao et al. [2012] and vLens by Hao et al. [2012] focus on the application side and use static code analysis to extract meaningful information about the software as well as code-injection to extend and annotate programs in order to collect information at runtime. The approach in Lit et. al. Li et al. [2013] uses code injection to annotate and extract information from the application and employs linear regression to find the energy consumption at the bytecode instruction level. These methods all provide high accuracy on the application side, but fail to consider the control flow of the application (e.g. loops and jumps) and in many cases they have a significant level of inaccuracy.

My approach focuses on the static analysis of the bytecode and code injection techniques to obtain measurements with several levels of granularity: class level, method level, internal control flow level, and Android API level. On the control flow level I looked at the basic blocks (Lilja [2005])and employed the technique known as dominator analysis to extract loops and jumps (Allen [1970]). To account for the system behaviour, I periodically logged most system activities (i.e. CPU frequency, network usage, ...) that can be further used to refine the analysis, in our case, on AnyRun solutions if needed.

4.2.2 System architecture

There are several levels that are simultaneously approached in my solution in order to provide a useful tool chain to accurately measure the power consumption in modern smart devices. POEMs works at:

- *hardware level*, where I provide a portable external power meter.
- *system level*, where I provide an Android system logger.
- *software level*, where I provide an offline code analysis tool to extract the internal properties of the application as well as a code-injection mechanism to annotate the internal state of the application to track what happens at runtime.

Figure 4.1 shows the path that must be followed in order to use the POEM. The first step consists in the annotation and extraction of the meaningful features from the application's bytecode. The second step is to run the application and simultaneously collect the traces with an external power meter and with the phone logger (basic blocks, method usage, and system status). The final step is the analysis and the visualization of the results.

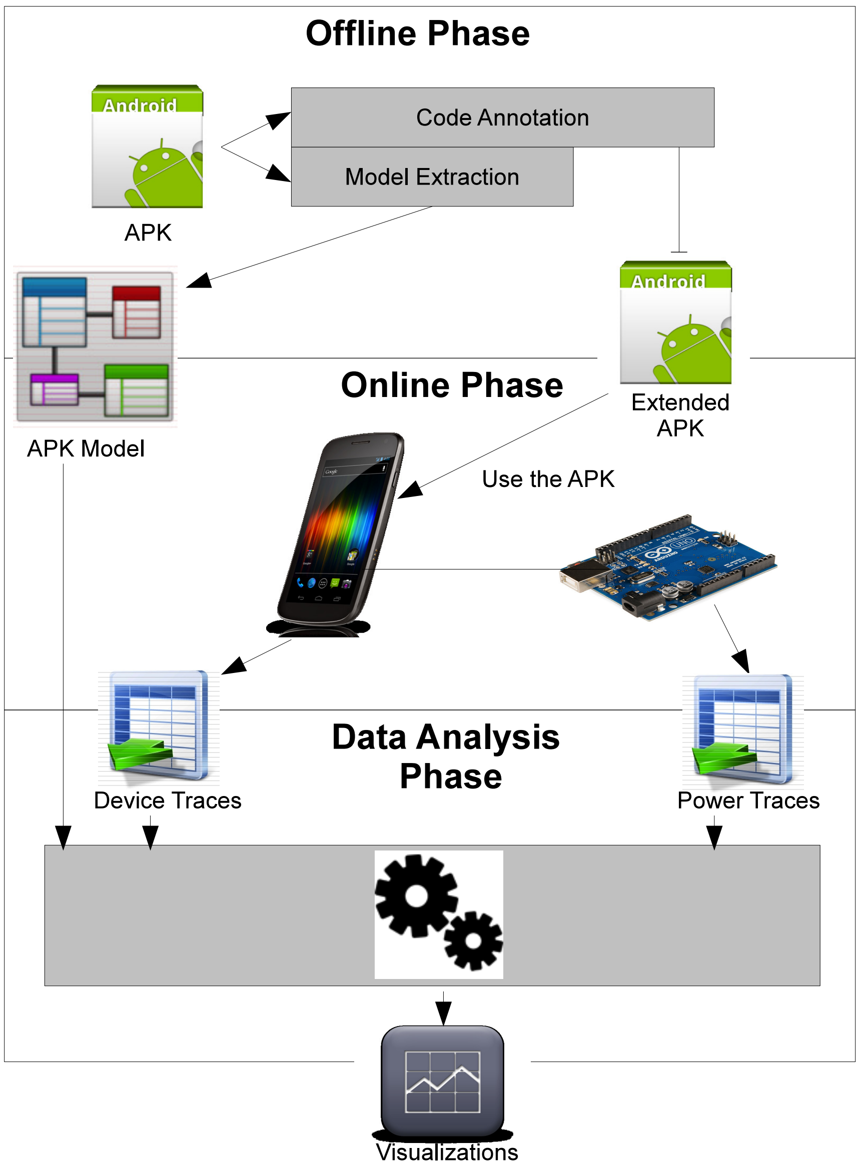


Figure 4.1. The various phases needed to analyze the power consumption of a specific application.

Hardware Layer

The core of the portable power meter prototype is the Arduino Leonardo board. It uses the ATmega32u4 micro controller with 32KB flash memory and 2.5KB of SRAM Memory. Arduino offers 20 digital I/O pins and 5 Analog pins. The analog pins can read signals from 0 to 5 V and their ADCs have a 10 bit resolution (roughly 5 mV). The minimum reading time for an analog pin is 100 microseconds, which corresponds to a maximum reading frequency of 10KHz.

On the smartphone I used a rooted Galaxy Nexus smartphone [3] equipped with a 1.2 GHz ARM Cortex CPU, 1 GB of RAM, running Android 4.3 Jelly Bean. I placed a shunt resistor in series with the phone (a shunt resistor is designed to have a very low ohmic value causing a negligible voltage drop that does not affect the smartphone), and I used the Arduino ADC to read the voltage across the shunt resistor, whose value is equal to 0.1Ω in the prototype. The current through the shunt is the same as the current drawn by the smartphone and can be computed with Ohm's law based on the voltage readings. The corresponding power draw of the smartphone can be obtained as the product of the computed current and the input voltage (3.7 V for our phone).

To improve the granularity of the readings, I used an op amp to amplify the voltage across the shunt (by a factor of $\approx$ 10 in our implementation), and I used a reference voltage of 3.3 V in the Arduino ADC to achieve a resolution of 12 mW. I extended the Arduino board to save the results to an external SD-Card and I created an application to track and record the real-time power draw of a smartphone. The recorded measurements were synchronized with the device so I knew the energy footprint of the various components of an app. I used LED2LED communication Giustiniano et al. [2012] between an external LED connected to the Arduino and the phone camera LED to initiate and terminate the recording of energy measurements, which were saved to a dedicated file in the Arduino's SD-Card. The Arduino board always begins by measuring prior to the phone due to the duration of the LED pulse and the lead time of the Arduino board has been found to be fairly deterministic.

System Layer

Many power consumption factors are not directly related to applications (e.g. screen light). So I provided a system logger that monitored the phone hardware usage (CPU usage, memory usage, network usage and connectivity, screen light) and stored the system *logcat*, which was used to extract other events (e.g. the

[3]https://www.google.com/intl/us/nexus/

calls to the Garbage Collector) and stored the logs of the extended version of the APK (the original APK is extended with code annotations as shown in Section 4.2.2).

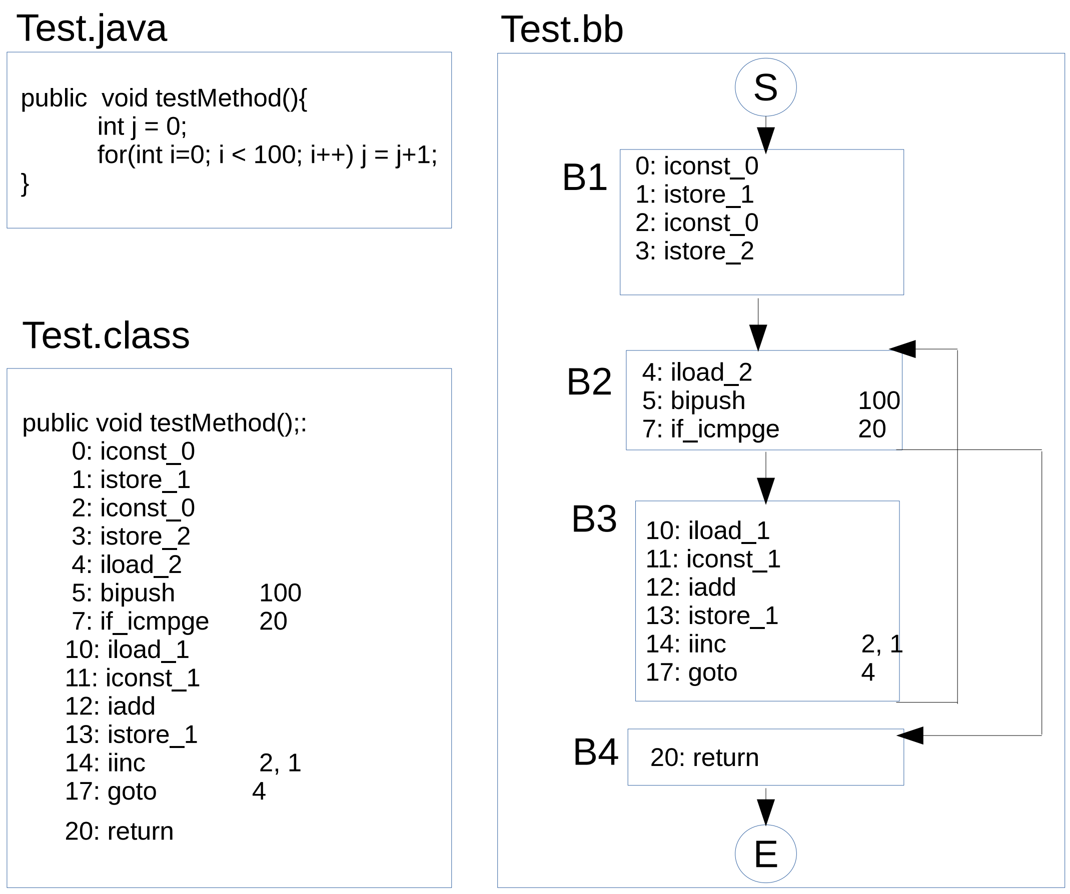
Application Layer

Following Hao et al. [2012], I performed a static analysis of the bytecode to obtain a model that describes the internal properties of the applications (from classes to method control flow). I then used code injection techniques to mark all points of interest inside the application: classes, methods, control flow (basic blocks), and Android API Calls. I pursued the static analysis of the bytecode instead of the standard code annotation because different compilers added different optimizations that might of change the control flow (e.g. method inlining). Adding code lines prior to compile time may affect those optimizations, thus leading to significantly different energy consumption patterns. I automatically extracted the call graphs regarding the internal methods of the application.

For each method I also extracted the control flow basic blocks Allen [1970], (that may be described as a *a portion of code inside a program that has only a single entry and a single exit point*), A control flow graph may be extracted as shown in Figure 4.2 and marked each single entry point, which allowed us to follow the runtime path in the control flow of the application, thus providing detailed information regarding the power consumption of the runtime path.

As most of the execution time of an application is accounted for by loops, it is essential to determine whether a certain jump instruction corresponds to a loop. To this end, I applied dominator analysis Cooper et al. [2001], a well known technique used to retrieve several meaningful pieces of information from a control flow graph.

In a control flow graph, $G = (V, E)$, where V are the basic blocks (nodes) and E are their interconnections (edges), a node $d \in V$ dominates a node $n \in V$ if every path from the entry node to n must go through d. Any loop in G has a loop header node that dominates all other nodes in the loop as well as a back edge from a node back to the loop header. Therefore, loops can be identified by identifying back edges in G: first I built a structure known as a dominator tree where there exists a directed edge between nodes $a \in V$ and $b \in V$ if and only if a is the immediate dominator of b (i.e., the closest dominator in G), and then I added all the missing edges that were originally contained in E. A loop exists if any of such edges goes from a node to one of its dominators. An example is shown in Figure 4.3, where the back edge from B3 to B2 indicates the presence of a loop. Once I knew the set of blocks that belong to the loop, I counted the number of


Test.java

```
public  void testMethod(){
        int j = 0;
        for(int i=0; i < 100; i++) j = j+1;
}
```

Test.class

```
public void testMethod();:
    0: iconst_0
    1: istore_1
    2: iconst_0
    3: istore_2
    4: iload_2
    5: bipush        100
    7: if_icmpge     20
   10: iload_1
   11: iconst_1
   12: iadd
   13: istore_1
   14: iinc          2, 1
   17: goto         4

   20: return
```

Test.bb

```
B1
0: iconst_0
1: istore_1
2: iconst_0
3: istore_2

B2
4: iload_2
5: bipush        100
7: if_icmpge     20

B3
10: iload_1
11: iconst_1
12: iadd
13: istore_1
14: iinc          2, 1
17: goto          4

B4
20: return
```

Figure 4.2. Example of Basic Blocks Analysis: Test.java shows the standard Java code, Test.class hows the java byte code and Test.bb shows the basic blocks extracted from Test.class

iterations for each method call and determined the path followed within the control flow at each iteration. The corresponding energy footprint can then be determined based on the power traces. This approach arguably offers a much more detailed view compared to previous approaches because it accounts for the control flow variations that may occur across different runs due to different usage patterns that translate into different energy consumption profiles.

Visualization The Visualization toolkit, provided with the tool, provides an easy and simple way to navigate through classes and explore their details (i.e. control flow on basic blocks or calls per method). Another interface allows users to load and parse the traces collected during real time experiments, furthermore those traces can be visualized in the GUI (in the form of statistics) or are managed by

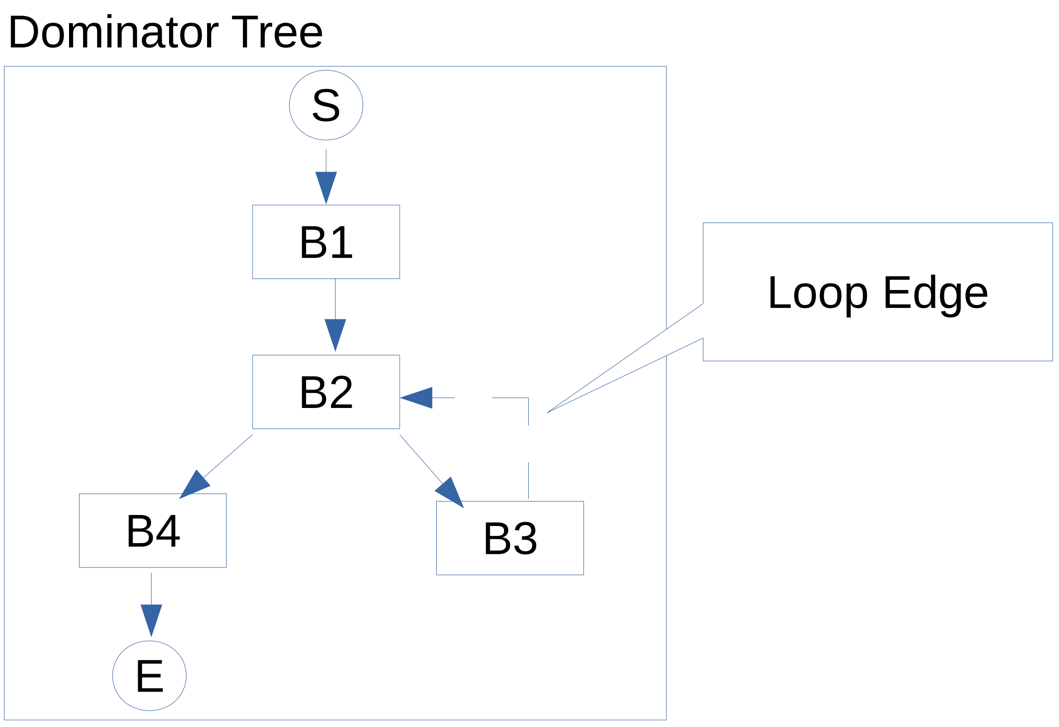


Figure 4.3. Example of Dominator Analysis and Loop catching that uses the structures showed in Figure 4.2 as starting point

exporting them managed with external tools (i.e. Matlab).

With the help of that tool the area of the experiment concerning the offloading strategie can be isolated. Thus if the device status are the same for both experiment they can be successfully compared.

4.2.3 Limitations

It is almost impossible to measure the behaviour of a piece of software without affecting its performance. The main goal of this section is to show how much POEM affects the behaviour of an Android application using a set of simple examples.

I want to point out that those are not connected to the benchmark problems used to evaluate ARC. This section only provides an insight about the limitation of the proposed method with the usage of a set of simple (thus easy to analyse) problems.

I proceeded by using a well known sorting algorithm, namely *Shell Sort*, whose computational complexity is a function of the input size *O(n log n)*. I implemented the algorithm in a simple Android application that ran on random generated array with constantly increasing dimension. Increasing the array dimension increases the computational time and thus also the number of records to be extracted by POEM.

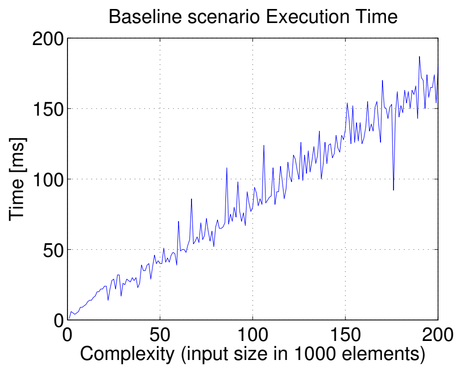
Figures 4.4a-4.4d shows the experimental results. I considered a baseline scenario where the application is run without any modifications and a modified scenario where POEM is employed and uses two dimensions of cost: runtime and energy consumption. Figure 4.4a and 4.4c show, respectively, the runtime and the energy consumption in the baseline scenario while Figure 4.4b and 4.4d shows, respectively, the run time and the energy consumption when POEM is used. Runtime and energy consumption are plotted as a function of the input size for an individual method call, which serves as a proxy for complexity. I observed that using POEM results increased both runtime and energy consumption of about four orders of magnitude; this happens because each time a basic block or a method is run POEM needs to access the file system for logging purposes, resulting in a deterministic logging cost (in terms of either runtime or energy consumption) C_L. The extra runtime and extra energy consumption represents the cost of gaining major insight into the operation of an application and does not affect comparisons among methods. If I let C_{B_i} be the cost of running basic block B_i (again, in terms of either runtime or energy consumption), then the cost of the baseline scenario is equal to $\sum_{i=1}^{N} C_{B_i}$ and the cost of running POEM on top of the baseline scenario is given by $\sum_{i=1}^{N} C_{B_i} + NC_i$, where N denotes the total number of basic blocks; therefore, the extra cost of running POEM can be subtracted away and does not affect comparisons among methods (in fact, Figures 4.4a-Figures 4.4c and Figures 4.4b-Figures 4.4d exhibit comparable trends).

4.2.4 Examples of Usage

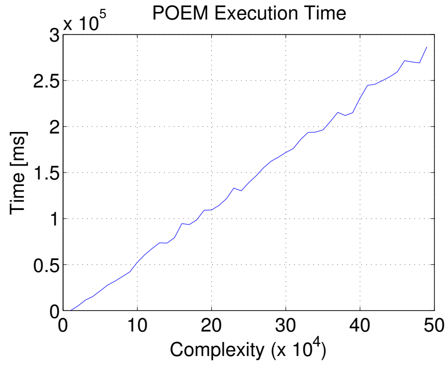
In order to show the possibile usage of POEM I follow the approach of Hao et al. [2012] and build a custom-made application that implements three simple problems, namely:

- *B1:* 10x10 matrix multiplication
- *B2:* 10MB file writing on the internal memory
- *B3:* MD5 hashing over a 10 byte string repeated 20 times

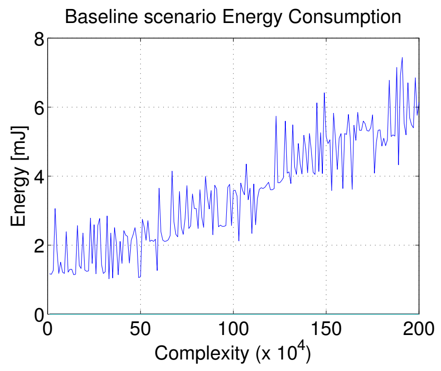
To ensure I have a complete understanding of the application and its methods, I used a custom-made application as opposed to a commercial application. The



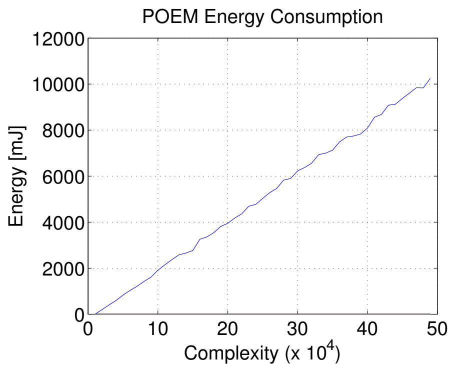
(a) Execution time on the clean application



(b) Execution time on the modified scenario (POEM)



(c) Energy consumption on the clean application



(d) Energy consumption on the modified scenario (POEM

Figure 4.4. Experimental results on Shell Sort example.

three simple problems, were coded in my custom-made application containing 17 classes (3 of which are the test cases, 1 implements the Main class that calls the test, and the others are Android support classes), and 23 methods. I ran each problem 100 times.

After, I extracted the internal method characteristics using basic block analysis. The results are shown in Table 4.1. I immediately saw that the most computationally intensive problems, is B1 with 7 loops, 12 blocks, and 10 branches. I would like to point out that the 7 loops are not directly part of the algorithm, but some of them are needed to randomly generate two matrices.

The costs in terms of runtime and energy consumption are shown in Figure 4.5. I observe that B2 consumes more time and energy compared to others, which can be explained by looking at its I/O operation. Though B2 has only 2 loops, it accesses the I/O as many as 6 times. I also observe that B1 requires more

	Global	B1	B2	B3
block	292	12	4	4
branch	21	10	1	1
loop	10	7	2	3
calls	5	0	0	0

Table 4.1. Internal method characteristics retrieved with the basic block analysis

energy than B3 due to its extra CPU usage. I also observe that that all confidence intervals are relatively narrow for B1 and B3 and wider for B2, which suggests extra system activity triggered by B2's buffered writes.

4.2.5 Impact of system states

It is very difficult to compare different software entities due to the many variables at play. For instance, different hardware states result in different energy footprints (i.e. reducing the screen light dramatically decreases the power consumption). As noted earlier, the solution consists in logging the system status (i.e. screen lighting) in order to differentiate among different methods or block calls. A problem that is not addressed with standard logging techniques is the presence of energy tails due to unclean state transitions, which depend on the internal characteristics of smartphones.

The presence of such tails may affect the estimation of the energy usage of whatever task happens to overlap with energy tails. Currently, I address this problem by disallowing comparisons in two cases: If one of the blocks to be compared is run immediately after any hardware component has changed status, and if the one of the blocks to be compared is run in a different hardware state than the other block (i.e. with different screen lighting). Residual errors may persist because I cannot expect to capture every possible event that may take place. I therefore suggest multiple test runs to isolate out outliers and ensure consistency.

4.3 Accurate Environment Emulation

The dynamic nature of the AnyRun environment requires novel solutions to perform accurate and exhaustive experimentation. Since performance is the key

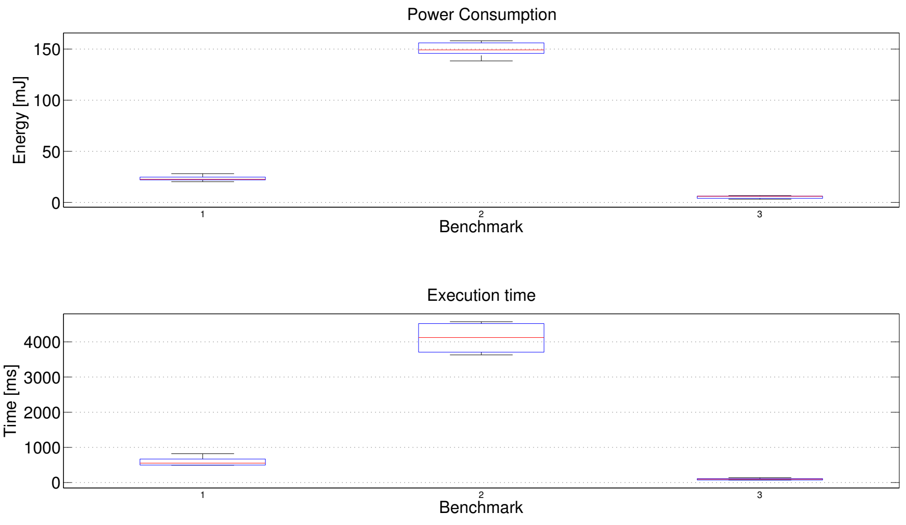


Figure 4.5. Costs in terms of runtime and energy consumption observed during the runs of our three simple problems,

element to be studied, performing the study via simulations is not accurate, due to the difficulty in identifying all the variables involved when executing a piece of application in a modern device. To overcome this limitation I propose using DroidLab, a different approach based on **emulation**. Droidlab is a testbed for smart devices that has been created to offer the flexibility of a simulation, as well as accuracy of the execution in a real environment. This section deeply explains its architecture and its usage in the context of AnyRun testing.

4.3.1 Related Work

Testbed solutions have been widely adopted in Wireless Sensor Networks (WSN) domain. The main goal of a testbed is to provide easy access to a large number of devices deployed in a real scenario where researchers can run their experiments and easily collect their results. A pioneer in that domain is Harvard's Motelab testbed built by Werner-Allen et al. [2005]. At its peak in the second part of the past decade, Motelab provided access to hundreds of sensor nodes (motes) deployed over three floors in an office building. Motes are managed and accessed through a web interface where researchers are allowed to deploy their software and plan the time of their experiments; data is collected into a MySQL server that receives all the motes data from the backchannel. Many other WSN testbeds have also been set up (i.e. Zurich ETH FlockLab by Beutel et al. [2009]), but all of them share the same basic principle: providing remote access to devices and letting users set up their own experiments.

As previously mentioned, opportunistic technologies are based on human mobility, an element that is totally missing on WSN-based testbeds. LiveLab by Shepard et al. [2011] uses real devices carried around by real people to offer access at a real environment. LiveLab is the best possible solution to test OC applications, however, setting up the experiment is harder compared to Motelab, due to the participation of real people that could potentially lead to issues with privacy laws.

Hardware and network emulation in testbeds has been adopted in MovIt from Giordano et al. [2012] from . where a fully working testbed able to emulate mobility patterns into real devices was built. MovIt works into a set of virtualized operating systems connected through a controller interface that defines the connection pattern between them. In the real implementation, the packets have been filtered trough the MovIT controller, which shapes dynamically the network taking the parameters from SUMO (Urban Mobility Simulator) from Krajzewicz et al. [2002]
I plan to propose a completely different approach. I built a smartphone-based testbed following the Motelab architectural style and I followed MovIt in the way that it emulates the contacts among devices. Users are allowed to setup the testbed with their own applications and contact traces, as well as have access to a set of real devices where they can monitor the software and protocol performances into emulated scenarios.

4.3.2 Architecture

Droidlab Ferrari et al. [2013] is a custom, scalable Android Emulation Testbed that enables protocol testing with real hardware under realistic conditions with connectivity patterns dictated by contact traces. In DroidLab, contact patterns are managed in a central server that periodically forwards the new pattern to the connected devices. Connections in the devices are filtered with *iptables* [4], a user-space application that permits to configure the Linux Kernel firewall (usually implement in the kernel as a Netfilter module). *Iptables* is currently offered in the majority of the GNU/Linux distributions including Google's Android.

A global picture of the testbed is Architecture is showed in Figure 4.6 that contains the following key components:

[4]http://www.netfilter.org

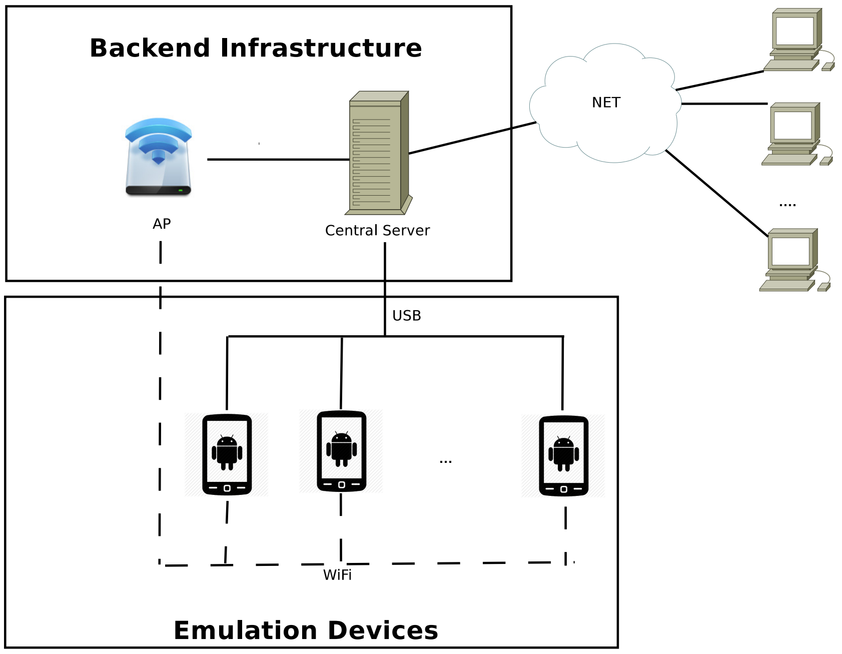


Figure 4.6. DroidLab Architecture

Time Deamon:

The goal of the time-deamon is to process the mobility traces that are part of the experiment's input, determine how many devices are involved and connect them to the predefined trace. Periodically, the deamon sets the devices connections by looking at the trace and informs all devices by sending messages through the USB back-channel. The update-time is a parameter that must be set in the contacts file.

APK Manager

This application manages the Android Applications packages allowing their dynamic deploying in the connected devices. Applications are stored and managed by the APK Manager, which keeps track of their location and dynamically uploads them to all the Android devices when the related experiment starts and removes them after the experiment ends.
The APK Manager is responsible to keep the Smartphones clean after each execution and constantly checks whether the application has crashed, in which case it removes the application from the phone and sends an alert to the user who is responsible for the experiment.

Logger Manager

Logger manager starts a communication with all the Smartphones every time an experiment is started and puts all the logged-data into a MySQL database. In case there is no log-data coming out from a smartphone the user is notified.

Device Manager This component has the primary goal to manage the connection between multiple devices using Netfilter as the main tool following the traces the experimenter has uploaded

4.3.3 Droidlab in AnyRun

DroidLab is the key tool to perform dynamic and exhaustive emulation in AnyRun domain. Hence, to make it work with POEM a change is needed. POEM requires that the device currently under examination is not connected to the USB backchannel, instead it logs everything internally. This is necessary because the USB injects energy into the device, making the POEM's measurements totally useless.

4.4 Conclusions

In this chapter, a complete solution to test AnyRun-based applications has been presented. The solution is composed of two main components; the first, POEM that allows researchers to have fine-grained power measurement on their application and the second, Droidlab, that allows to emulated a realistic network scenario with multiple and heterogeneous devices.

The key advantage of the proposed solution is that it allows testing AnyRun-based application on real devices. This is fundamental because in the AnyRun domain the solution is effective only if it has performance improvements. It is widely understood that this is very difficult to prove without measuring it directly on the device.

The testing environment will be used in the next Chapter to deeply investigate ARC performances in the considered domain using three benchmarks. ARC will also be compared to state of the art solutions in order to gain more insight into its behavior.

Chapter 5

Evaluation

5.1 Introduction

As previously introduced the anyrun computing paradigm allows to perform the offloading of computation into a set of heterogeneous devices that may be located in different positions (i.e. in the closest proximity or into a remote datacenter). Basically, there are two main scenarios that I am interested in studying:

- *Cloud-Based Scenario:* In this case, the device accesses the service on a high-performance device through WANs communication. The key challenges in this scenario are mainly connected to WANs communication that may add a significant delay (even infinite) on the offloading that may lead to a failures.
- *Cloudlet - Opportunistic Scenario:* in this case, the device is located in the proximity and it is accessed through LAN communication mechanisms. The key challenges there are mainly dictated by the Users mobility that makes the connection between the hosts very unstable. A second challenge is the heterogeneity of the devices, their internal characteristics affect the performance of the offloading.

This chapter shows the empirical investigation I have performed for both scenarios. At first, the benchmark problem used to perform the evaluation are presented and compared. Then, thanks to the testing environment described in the previous chapter, I presents the setup and the results of the evaluation.

Before concluding the chapter, I will present the evaluation of the performances for the ARC extension for CodeMobility.

The key goal of this section is to quantify the cost of moving the code in conjunction to the data needed for the computational offloading process. The main outcome of this analysis is to determine if and in which circumstances moving the code can be used. More specifically, it is important do determine if the impact of code mobility on the offloading process can be tolerated.

The chapter concludes by highlighting the key findings that I have observed during the process.

5.2 ARC Evaluation Benchmarks

To investigate ARC's performances I selected three benchmark problems. They are well known problems from the literature and used into a wide variety of applications. They are:

- *The Ant Colony Optimization (ACO)* by Dorigo et al. [1996]. algorithm is a an artificial intelligence technique based on the pheromone-laying, food-searching behavior of ants. ACO algorithms have been applied to a wide range of combinatorial optimization problems and have also been used to find near-optimal solutions to the travelling salesman problem (TSP). I choose a TSP problem with 52 cities that is computationally intensive but it is very light in sending data.
- *Face Recognition (FR)*: we use the eigenfaces approach proposed by Turk et. al by Turk and Pentland [1991]. The widely adopted eigenfaces approach can recognize a face in a user-provided image by comparing features from an existing database of faces. The eigenfaces approach works in an unsupervised manner to learn and recognize new faces. This benchmark is the heaviest in both computation and data exchanged between hosts.
- *Character Recognition (CR):* we use a pre trained Kohonen Neural Network by Kohonen [1990] trained with 32 pre-parsed sets of printed characters and then use the neural network to identify 1000 randomly chosen characters within the aforementioned sets. This case try to stay in another extreme where the comunication problem is light compared to the cost of transmission.

The benchmarks selection has been made to emulate the problems that are usually connected with Computation Offloading. The ACO benchmark requires the least amount of transmission costs compared to the others but it has the highest computational needs. Face Recognition has the highest transmission cost because it has to share a set of images but it can be compared to Character Recognition in term of computation. At the end, Character recognition is the benchmark that requires fewer resources in both the considered dimensions.

5.3 Cloud-Based Scenario

Two main physical actors play a role in this scenario:

- *Amazon Elastic Compute Platform EC2 as a state of the art solution for cloud*: EC2 is the central part of the Amazon.com's cloud computing platform also know as Amazon Web Services. It allows its users to rent a virtual computer where they can run their own applications. A user can launch/ and terminate as many virtual instances as she needs and at the end it pays only for the hours they have been working. Another feature of EC2 is the geographical distribution where the instance can be started and executed in a set of predefined locations (i.e. EU or USA). On EC2 I have used a Ubuntu Virtual Machine in a standard EC2 setting with a single Intel Xeon CPU and 0.5GB of RAM Memory.
- *Google Galaxy Nexus smartphone* equipped with a 1.2 GHz ARM Cortex CPU, 1 GB of RAM, running Android 4.2 Jelly Bean.

The key goal of the following section is to study the behaviour ARC in case the Cloud is used as offloading candidate. To this extent I proceed as follow: at first I execute the program locally on the mobile device, the key idea here is to have a common baseline. After, I run the benchmarks with ARC and access to EC2. The Cloud is accessed by using three type of WAN connections:

- *3G:* indicates the third generation of mobile telephony standard it has been introduced in 1998 and has experienced several evolutions. The current standard in Switzerland is named HSDBa that has as maximum data rate 14.4 Mbps.
- *4G:* it indicates the fourth generation of Mobile Telephony standard, it has been introduced in 2006 and it increases by more than one order of magnitude the data rate of the 3G standard, with theoretical maximum speed up to 100Mbps.
- *LAN* this is the case when the mobile device is connected to the local LAN via WiF and exit to the internet with high-speed connectivity, in this scenario I used the SUPSI's WAN access which is done via fiber optics reaching a data rate up to 1Gbps.

	ACO	FR	CR
Local	5	15	30
3G	10	20	32
4G	20	36	50
Wired.	20	50	52
TOT	75	121	164

Table 5.1. Number of Experiments per Benchmark in the Cloud-based Scenario

Parameter	Definition
Local	Executing the benchmark in the current device (Seeker)
3G	Executing the benchmark it the EC2 Cloud via 3G WAN connectivity
4G	Executing the benchmark it the EC2 Cloud via 4G WAN connectivity
LAN	Executing the benchmark it the EC2 Cloud via HighSpeed wired connectivity

Table 5.2. Parameters Definitions.

In the remainder of this section, I show the results of the experiments. To carefully evaluate each possible scenario I've executed up to 15 hours of continuos runtime (1 hour per experiment), the overall number of experiments is illustrated in Table 5.1.

The figures 5.1, 5.2, and 5.3 illustrate the main outcomes of the experiments performed for each considered WAN access standard. Table 5.2 shows the definition of the parameters used in the figures; each illustrates the distribution of the energy consumption of a single method call and highlights the mean as well as the 95% confidence intervals.

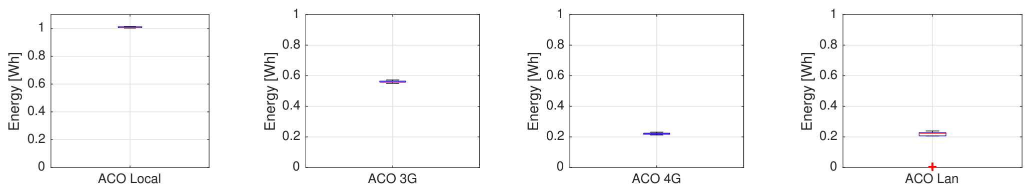


Figure 5.1. Energy consumption of the ACO benchmark in the four WANs Technologies.

As expected the benchmark with the biggest reduction compared to local execution is the ACO benchmark. The key reason is the small amount of data exchange (only a data structure containing a 52x52 matrix is shared) and in its high

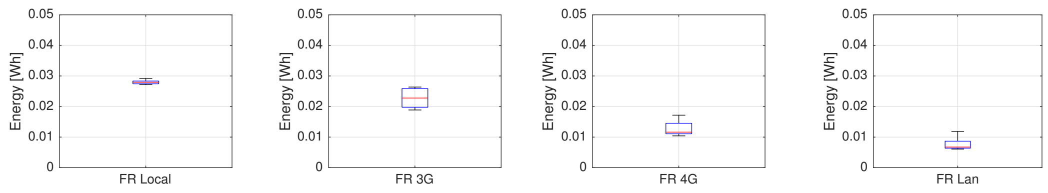


Figure 5.2. Energy consumptions of the FR benchmark in the four WANs Technologies.

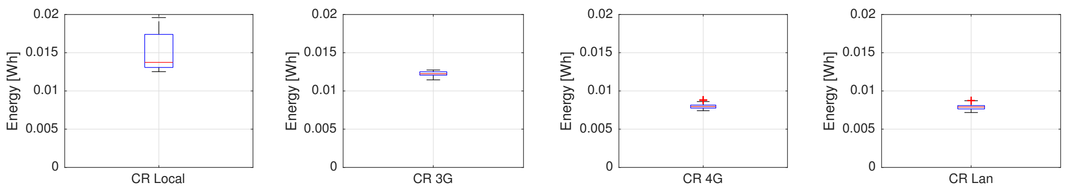


Figure 5.3. Energy consumptions of the CR benchmark in the four WANs Technologies..

demanding computational needs. The aforementioned properties (high computational demand and low data exchange) make the ACO the perfect candidate for WAN-based offloading. But, even in this scenario, the 3G network latencies add a significant cost almost double the cost of 4G and LAN connectivity. Also, in this case, there are no differences between 4G and High-speed LAN connectivity. In the last WANS technologies (4G and Fiber Optic) in both the latency times are fast enough and the cost is mainly dictated by the computation time.

On the other hand, both CR and FR benchmarks provide practically zero gain when the 3G WANs access is used but they provide interesting results with other two WAN access technologies. The explanation is in the benchmark transmission costs that are significant compared to ACO, consequently, the 3G WANs access does not offer enough speed to be effective but when 4G and LAN technologies are used they let the two benchmarks improve their energy footprint, but the improvement is not so sensible compared to ACO because, as said before, ACO is the benchmark that requires more computation compare do transmission cost.

It is important to mention here that the results of this section empirically demonstrate that the main outcome of the analysis presented by Cuervo et al. [2010] is still valid. In fact, Cuervo et al. reconsider the usage of 3G for offloading and move to cloudlet approach due to the prohibitive latency times in that domain. This work extends the mentioned analysis by: at first considering multiple WANs access technologies and second by proving the results empirically with a computation offloading framework.

Parameter	Values
Solutions Compared	ARC, MAUI
Benchmark Problem	Ant Colony Optimization, Facial Recognition, and Character Recognition
Network Typologies	Local, Fully Connected, Dynamically
Number of Devices	Single device, and Multiple Devices

Table 5.3. Emulation variables.

Exp.	**Scenario**	**Mean Time [s]**	**Median Time [s]**	**Mean ED [Wh]**	**Median ED [Wh]**
Local		110.2	111.8	1.04	1.04
ARC	Full	11.4	11.2	0.08	0.08
	Dynamic	20.0	11.0	0.18	0.08
	Dyn. MD	19.7	11.5	0.08	0.08
MAUI	Full	11.3	11.1	0.10	0.10
	Dynamic	103.5	104.8	1.04	1.04

Table 5.4. ACO Results Table (MD stands for Multiple Devices and ED stands for Energy Drain)

In conclusion, the key finding from this experimentation phase is that the cloud solves computational offloading problem very efficiently. However, in some circumstances (that strongly depends on the amount of data exchange and on the WAN access technology), the cost to use the Cloud is too prohibitive and this may cause failures in offloading or a sensibile reduction of the benefits.

The question to answer at this point is: is it possibile to use resources located in the proximity and accessed opportunistically to overcome the limitation posed by Cloud? The next section will answer the question.

5.4 Opportunistic / Cloudlet Scenario

In this scenario, I tested ARC against MAUI[1] in three experiments (plus one that consisted in executing the benchmark the local device) that covers all the possible alternative use-cases in opportunistic domain. Two of them only use a single high-end device (the MacBook Pro) to carry out a fair comparison with

[1]I used my own Android implementation of MAUI
, available at http://code.google.com/p/maui-android.

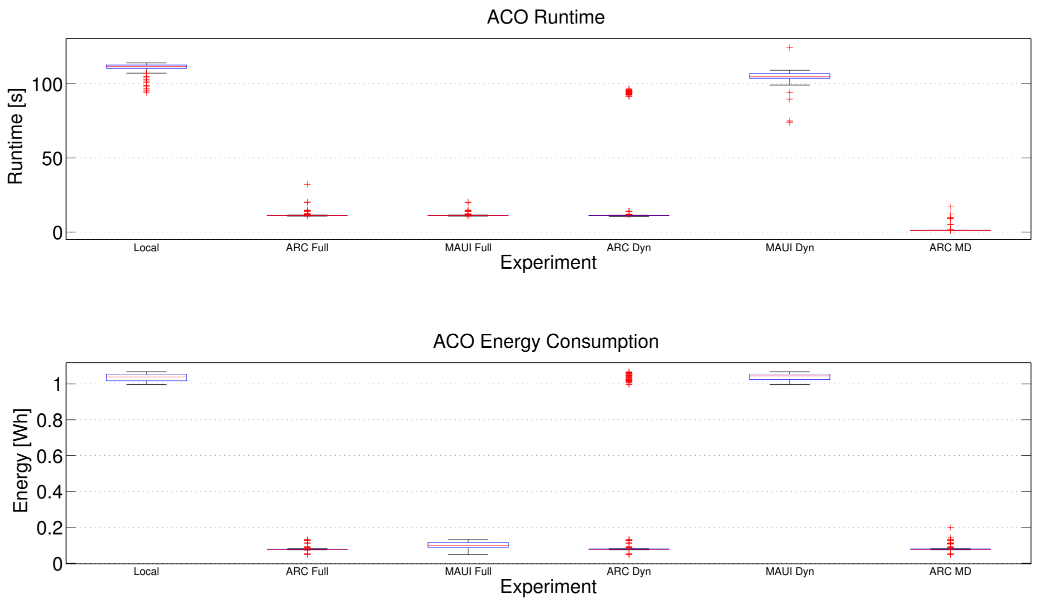


Figure 5.4. ACO energy consumption and runtime. MD stands for Multiple Devices.

MAUI, which is designed to work with a single piece of hardware located nearby (Cloudlet). The last one is designed to illustrate the benefits on using multiple devices, thus it will be performed only with ARC.

The experiments are:

- local execution: all benchmarks are run on the Nexus[2] smartphone to acquire a performance baseline;
- fixed network typology with a single high-end device (labelled as **Full**): all benchmarks are run with full connectivity;
- dynamic network typology with a single high-end-device (labelled as **Dynamic**): all benchmarks are run with an emulated dynamic network typology;
- dynamic network typology with multiple and heterogeneous devices (labelled as **Dynamic with multiple devices**): same as above, but with an augmented testbed.

[2]https://www.google.com/intl/us/nexus/

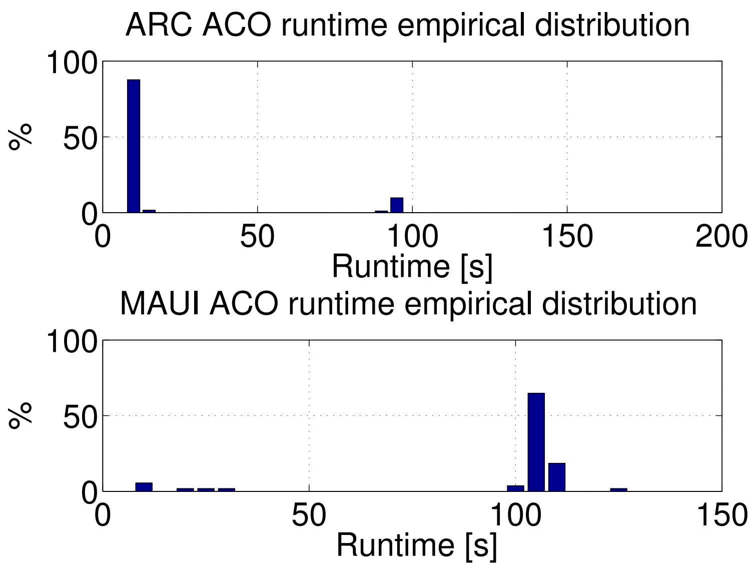


Figure 5.5. ACO Dynamic: runtime distribution with ARC and MAUI.

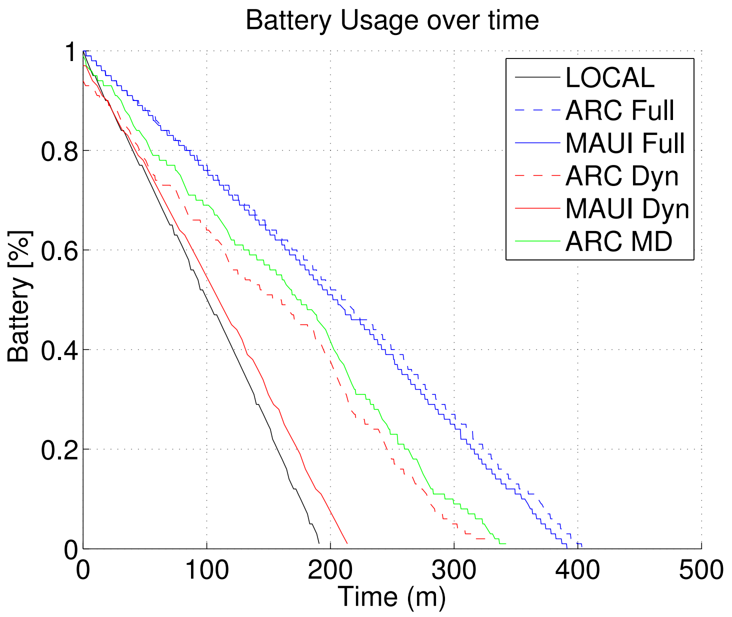


Figure 5.6. Battery usage (as a proxy for energy consumption) over time in a sample ARC run.

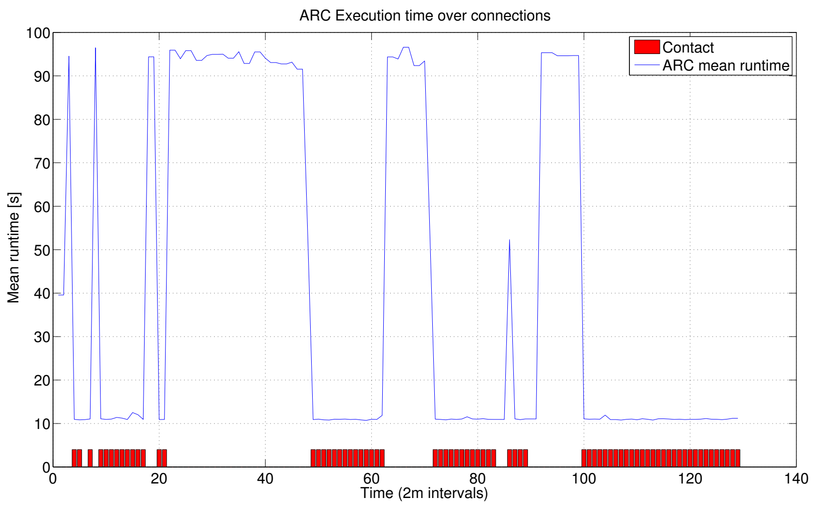


Figure 5.7. Mean runtime as connectivity conditions fluctuate in a sample ARC run.

The centerpiece of the experiment setup is the Google Galaxy Nexus smartphone equipped with a 1.2 GHz ARM Cortex CPU, 1 GB of RAM, running Android 4.2 Jelly Bean. Our testbed also features an Apple MacBook Pro laptop with an Intel iCore 7 CPU, 4 GB of RAM, running Mac OS X Lion (10.7.5). The setup also includes two extra Galaxy Nexus smartphones and a Samsung Galaxy Tablet 2 equipped with 1GHz dual core Processor, 1GB of RAM, running Android 4.0 (Ice Cream Sandwich). I collected data regarding the various runs of the benchmarks and stored it on the smartphone's SD card; runtime measurements are based on the internal clock. The dynamic scenarios are based on the Cambridge iMote data traces by Scott et al. [2006b] available on the Crawdad (cra [2013]) archive offered by Dartmouth college. The iMote traces contains the data collected from 70 users during INFOCOM 2006, where users were asked to carry a set of Bluetooth-based devices that periodically scanned the users? neighbours. I subsample the dataset choosing five devices and reduced the sampling ratio (from 5 minutes to 2 minutes) to generate a more dynamic typology.

In the remainder of this section, I explain the results of the experiments and to carefully evaluate each possible scenario I've executed up to 80 hours of continuous runtime (up to 7 hours per experiment). The the overall number of experiments is illustrated in Table 5.5.

	ACO	**FR**	**CR**
Local	250	19400	1680
Full Maui	1260	126000	12600
Full ARC	1270	126000	12600
Dyn. MAUI	900	84000	6300
Dyn. ARC	280	23000	1940
MD ARC	1400	250000	21000
TOT	5360	621’00	54’440

Table 5.5. Number of Experiments per Benchmark in the Opportunistic Scenario

ACO Benchmark

For each experiment I ran both ARC and MAUI over a complete battery cycle (ranging from 4 up to 7 hours). Figure 5.4 illustrates the distribution of the energy consumption and runtime of a single method and highlights the mean as well as the 95% confidence intervals, while Table 5.4 offers a comparison of the mean and median. With a fully connected typology, there is only a small set of differences that are probably due to system or network vagaries, as observed by Mytkowicz et al. [2009]. Both mean and median values are comparable and the distribution of the runtime values and the energy consumption values are therefore very narrow. Key differences between MAUI and ARC emerge in the dynamic connectivity scenario, where ARC’s ability to better manage challenging typologies becomes clear. In the presence of dynamic connectivity, ARC outperforms MAUI in both dimensions of interest by up to a factor of 10. With dynamic connectivity, MAUI adapts its offloading strategy based on network latency. In our emulated network, latencies are subject to wide scale changes ranging from milliseconds to minutes. MAUI maintains a moving average of the network latency, sampling it each time offloading is performed and at regular intervals (every minute) if there is no offloading. MAUI uses past method execution statistics to make offloading decisions; if it decides to offload and the network connectivity breaks down, MAUI eventually reverts on local execution upon reaching a timeout determined on the basis of past method execution statistics; such statistics are updated to reflect the penalty of having tried to offload in unfavorable circumstances, and even if network connectivity is restored, decisions remain strongly biased toward local execution, thus resulting in extended runtimes and increased energy consumption. This leads to the behavior illustrated in Figure

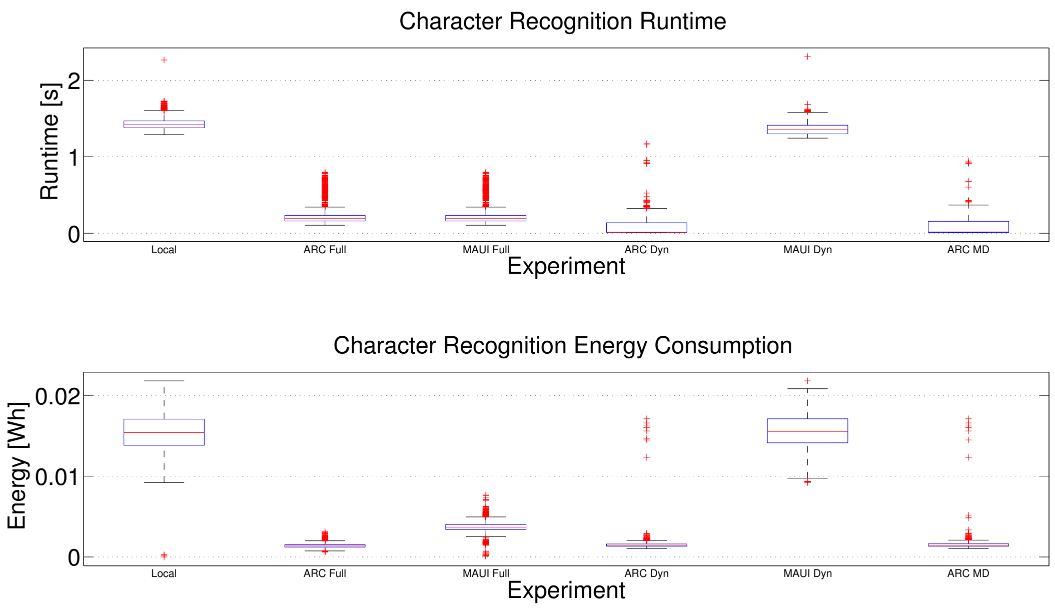


Figure 5.8. Character Recognition: energy consumption and runtime. MD stands for Multiple Devices.

5.5, which shows the distribution of the runtime for both ARC and MAUI with dynamic connectivity. With ARC, I observed a bimodal distribution reflecting its ability to quickly adapt to changing connectivity conditions: local runs (responsible for the higher mode) are only performed when no other options are available due to the lack of connectivity, while offloading to the laptop is performed whenever possible. On the other hand, once MAUI reverts to local execution, it sticks with it independently of the connectivity conditions.

When ARC is run over multiple devices (5.4), the Galaxy Nexus 3 smartphones lead to timeouts (as the CPU load and the battery level of all our Galaxy Nexus 3 smartphones is roughly the same, the actual runtime is roughly equal to the local runtime, but it is augmented by the round-trip time for communication). Therefore, ARC correctly chooses to leverage the behavior of the higher-end devices in the testbed, namely the MacBook Pro laptop and the Samsung Galaxy Tab 2 tablet. (In practice, ARC might choose to offload to smartphones identical to the initiator in case to extend its lifetime.) When both the laptop and the tablet are reachable, the laptop is preferred; the tablet is only picked when the laptop is unavailable. This leads to the behavior illustrated in Figure 5.4: ARC with multiple devices behaves similarly to ARC with dynamic connectivity; the

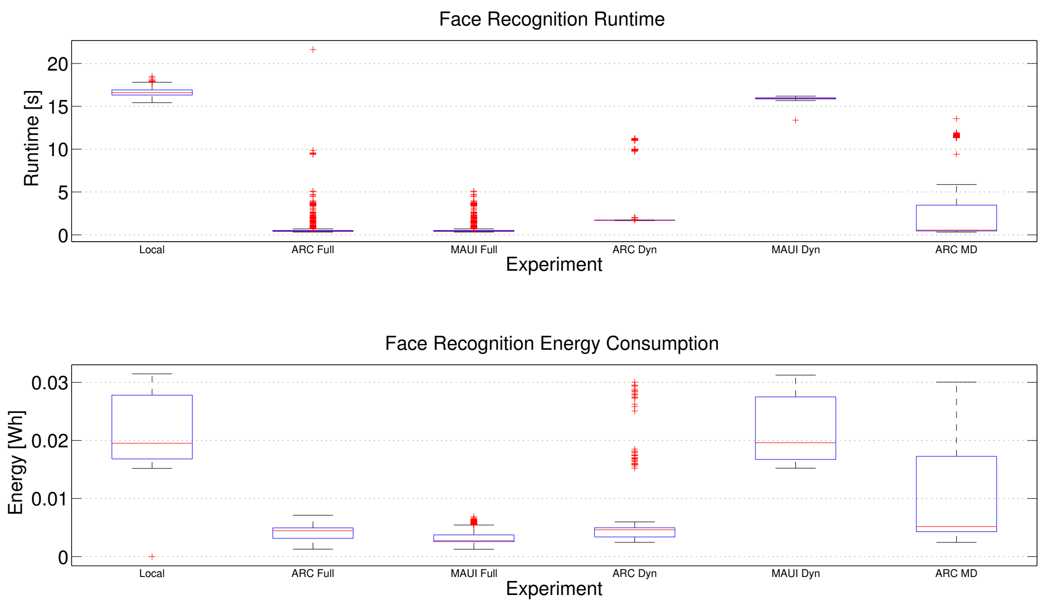


Figure 5.9. Facial Recognition: energy consumption and runtime. MD stands for Multiple Devices.

extra data points that lie roughly in the middle of the distribution range and are only present in the **Multiple Device** results correspond to jobs offloaded to the tablet.

Figure 5.6 illustrates the battery drain over a sample ARC run that covers a whole battery cycle. ARC and MAUI behave similarly with fixed connectivity (offloading results in a 110% extension of the battery lifetime compared to local execution). With dynamic conditions, ARC's battery lifetime gain is much more significant than MAUI's, resulting a battery lifetime extension of about 62% compared to MAUI and as much 71% with multiple devices; in such conditions, MAUI only achieves a modest 13% gain compared to local execution.

Figure 5.7 shows a sample ARC run with dynamic connectivity and illustrates (Boolean) connectivity (in red) and the moving average of ARC'sruntime (computed over a 2 minute time window that matches the contact duration if the trace fed to the testbed for this run). It is clear that ARC takes fulla dvantage of all the available connectivity opportunities.

Character Recognition and Facial Recognition

Figure 5.8 shows the results for the character recognition benchmark in the dimensions of interest. Similarly to the ACO benchmark, both ARC and MAUI results in a significant offloading gain with fixed connectivity and ARC outperforming MAUI with dynamic connectivity. The results with multiple devices are also fairly close to the ones from the ACO benchmark. Figure 5.9 shows the results for the facial recognition benchmark in the dimensions of interest. Due to the nature of this benchmark, the results have a significantly wider statistical distribution. Each image is loaded and parsed, resulting in a large number of I/O operations that affect virtual memory management, leading in numerous calls to the Garbage Collector that create extra work, affecting both the runtime and the battery usage.

The key insight from these two benchmarks is that ARC is able to adapt to a different kind of algorithm that has different spatial and temporal needs in terms of computation. In fact, FR has higher communication costs compared to ACO, but a comparable computational cost. In CR, instead, communication costs as well as computational costs are lower compared to FR.

5.5 CodeMobility Evaluation

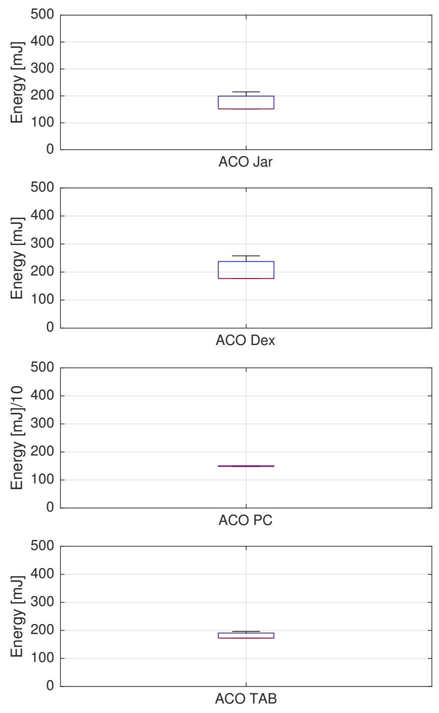
The goal of this section is to evaluate the ARC extension for Code Mobility. Specifically, the goal of this section is to quantify the cost of moving the code in conjunction to the data needed for the computational offloading process determining if and in which circumstances moving the code can be used.

Due to its peculiarities the CodeMobility extension for ARC can be evaluated separately. With respect to the first senario (Cloud) CodeMobility is totale useless because it is reasonably presume that the cloud already owns the code or it can easily access it on the application store or any other file sharing system. From Opportunistic perspective Code Mobility it is important because without any central authority it is unthinkable that a given provider already owns the code but this activity is performed only the first time a given seeker offload to a given provider and only in case the last one do not owns the code. Thus, the evaluation focus on quantifying the overhead for each given code and at end concludes by a reasoning over the overall offloading process.

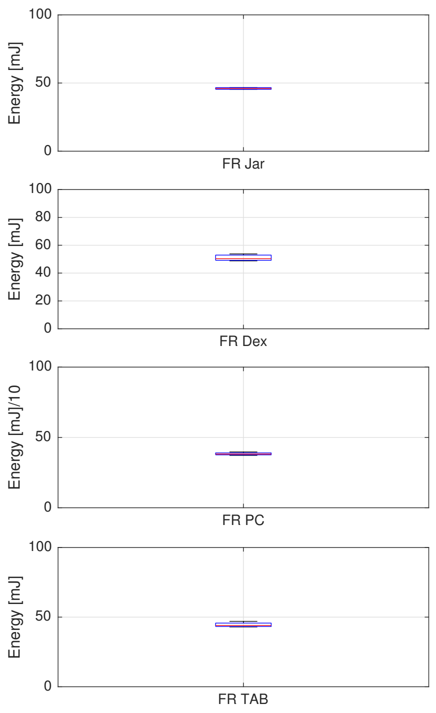
To evaluate the code Mobility inside ARC I used three devices; they are:

- *Google Galaxy Nexus smartphone*[3]*:* equipped with a 1.2 GHz ARM Cortex

[3]https://www.google.com/intl/us/nexus/



(a) Energy consumptions of the ACO benchmark
in the four scenarios.



(b) Energy consumptions of the FR benchmark
in the four scenarios.

CPU, 1 GB of RAM, running Android 4.3 Jelly Bean.

- *Apple MacBook Air laptop*: with an Intel iCore i7 CPU, 7 GB of RAM, running Mac OS X El Capitan (10.11.3).
- *Samsung Galaxy Tablet 3*: equipped with 1.2GHz dual core Processor, 1GB of RAM, running Android 5.1 (Lollipop)

All the measurement are taken with the maximum screen brightness setting so its overhead can be easily filtered out.

I run ARC with and without the code extension to compare the extra costs generated in this process.

The first action consists in extracting the DEX and JAR files from all the three benchmarks; the resulting files are illustrated in Table 5.6.

The fact that the JavaVM and the Dalvik/ART VM use a different set of instructions is clearly visible in the table. In both ACO and CharacterRecognition

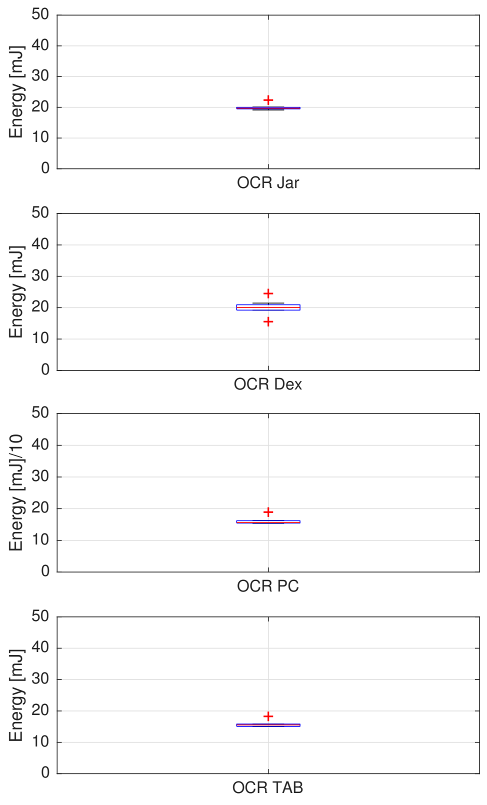


Figure 5.11. Energy consumptions of the OCR benchmark in the four scenarios.

File	Size [KB]
ACO.jar	8.05
ACO.dex	9.09
FaceRecognition.jar	868.77
FaceRecognition.dex	700.80
CharacterRecognition.jar	16.31
CharacterRecognition.dex	18.74

Table 5.6. Files generated in the framework and their size.

benchmark the JAR file is lighter compared to the DEX file but in FaceRecognition I experience exactly the opposite situation. The reason lies in the fact that FaceRecognition uses the Colt math library[4] that employs a large set of primitive instructions that are slightly more optimized in the Dalvik/ART virtual machine.

I evaluate the systems at runtime inside the Galaxy Nexus phone. The key performance metric is the extra cost of sending the code.

For each benchmark I performed four distinct experiments:

- Offloading with the JAR file to the laptop (JAR)
- Offloading the DEX file to the tablet (DEX)
- Offloading to the laptop without the overhead given by the JAR (PC)
- Offloading to the tablet without the overhead given by the DEX (TAB)

The results for the ACO benchmark (are illustrated in Figure 5.10a; in this case I the overhead of the inclusion of extra files into the offloading request is rather negligible due to the computational-intensive nature of the ACO. The results of the FR benchmark are presented in Figure 5.10b in this case the cost of sending the DEX and JAR file impacts the overall cost of the offloading with up to 10 mJ in the JAR to laptop case. This is because the weight of the JAR file is more significant compared to the other scenarios while the computational intensity of FR is relatively lower. The results of the last benchmark (OCR) are presented in Figure 5.11; this benchmark is not very much computational intensive and, even if the code size is quite small, transferring it has a sizable impact on the overall process.

In conclusion, I can argue that the overhead of code mobility is generally negligible in the context of the offloading process. Thus, this solution can be

[4]https://dst.lbl.gov/ACSSoftware/colt/

adopted to extend the device reachability of ARC thanks to a more dynamic and rich framework.

5.6 Conclusions

This section has illustrated the evaluation of the ARC framework for two key scenarios. The first is the cloud-based scenario where ARC is executed in conjunction with cloud-based platform and the second case is in my local testbed with a set of heterogeneous devices. We see in both cases that ARC is able to manage and decide whether is the case to offload or not into the proposed remote device.

The first set of experiments demonstrated that relying on cloud for offloading is beneficial but also poses a set of limitations (as previously demonstrated by Cuervo et al. [2010]; basically problems with high transmission costs but low computational needs are less favorable to have gain when executed on distant cloud to to WANS latency times.

The second set of experiments compare ARC with MAUI (Cuervo et al. [2010]). Overall, in all the experiments I observe that ARC and MAUI offer comparable performances with stable connectivity and that ARC significantly outperforms MAUI with dynamic connectivity by up to a factor of 10 in the dimensions of interest.

Within the experiment I have observed several outlier values most likely due to system activity (Mytkowicz et al. [2009]). If multiple higher-end devices are available, ARC naturally benefits from a performance improvement. I also expect that ARC's performance would further improve as the number of higher-end devices increases.

The last experiment in the chapter illustrates the evaluation of the ARC extension about the code mobility showing that it has a very small impact in the offloading process with the considered benchmark problem. Thus, with the proposed benchmark this solution can be adopted without affecting the overall system performances. However, currently code mobility add a set of security treats; malicious hosts may use this technique to gain sensible information from the device by sending their costum code. This risk is even more exacerbated due to the fact that root access is required on the seeker side. A sandboxing mechanism is provided in ARC specifically to mitigated this risk.

In this section I have also evaluated ARC, by performing computational offloading to any resource-rich device willing to lend assistance, ARC is advantageous compared to local execution with respect to a rich array of performance dimensions. The device may be located in proximity and accessed opportunistically or via local infrastructure (i.e. cloudlet domain) or via WANs, in case it is located in a distant cloud.

The key lesson learned is that ARC is able to leverage complex and dynamic network conditions and choose the best provider for its computational offloading needs. It is important to mention that ARC behaves selfishly., Is a key element that is needed due to the partial and incomplete view of the network.

In fact, a possible solution to avoid the selfish behaviour needs a complete view of the network (moving to a NP-Hard combinatorial problem) that it can only be maintained with a centralized infrastructure. This is not the case in an opportunistic enviroment. Alternative solutions may be based on incentives in several forms; for instance a given industry may offer offloading as a service and place a set of several machines in proximity that offer this service and are accessed opportunistically via ARC. The reward in this case will be based on the common form (e.g. advertisement).

The privacy and security solutions presented in this thesis will be further improved in our forthcoming investigation in the CHIST-ERA UPRiSE-IoT [5] project.

[5]http://uprise-iot.supsi.ch

Chapter 6

Conclusion and Outlook

In the dissertation I have proposed a novel paradigm for computation offloading named *anyrun computing*, whose goal is to use any piece of higher-end hardware (locally or remotely accessible) for offloading a portion of the application.

Several contributions have been provided in this dissertation; the first is the definition of a set of use cases where anyrun computing may be beneficial. Then, I provided the analytical definition of the AnyRun Computing paradigm. Later in the dissertation, I introduced the software solution I built to approach the problem. I illustrated the environment created to investigate its behavior and I concluded with a detailed evaluation of the software solution.

Basically, the dissertation offered a complete overview from both an analytical and technical point of view for the AnyRun Computing paradigm clearly illustrating its advantages and its limitations.

6.1 Summary of the main contribution

The dissertation has provided four fundamental contributions that are:

6.1.1 AnyRun Computing Fundamentals

Cloud computing is very efficient but there are situations where it may not be able to satisfy the requests of its users efficiently. A set of examples are illustrated in Table 6.1.

The limitations due to latency times have been investigated by Cuervo et al. [2010]. In the dissertation I performed a set of similar experiments to measure the cost of offloading into the cloud. The key result is that the latency limitation

Use case	**Performances**
Latency times:	The bigger the payload, the worst the cloud-based solution performs.
Roaming:	Those costs make the offloading process prohibitive for the majority of users.
Mobile Plan Costs	Mobile plans pose a set limitations on the amount of WAN free-data usage.
Network Unreachable	the morphology of the terrain or the building architectural characteristics may block access to mobile WANs.

Table 6.1. Scenarios where Cloud-based solutions are not efficient.

experienced in Cuervo's experiments is still valid nowadays with modern WANs technology.

The previously mentioned limitations only apply to the cloud. *cloudlets* devices are no longer restricted to WANs technologies limitations because the hardware is located in the closest proximity. However, the Cloudlet approach has a huge limitation due to the fact that hardware must be pre-deployed and known a priori. For this reason the cost to give full Cloudlet coverage is definitely prohibitive.

6.1.2 Analytical Definition of the AnyRun Computing paradigm

To approach the challenges illustrated in the use cases presented previously, The solution in the form of a novel- computing paradigm is *AnyRun computing* and described as follows:

Rather than assuming the existence of a dedicated piece of higher-end hardware and being rigidly tied to it, AnyRun may elect to use any piece of hardware (accessible locally or remotely) that can do the job better (in terms of impact on the local energy footprint).

With AnyRun Computing I removed the boundaries that tie the solution to an infrastructure by adding locally available devices to augment the chances to succeed in offloading.

To achieve the goals of the dissertation it is fundamental to have a clear view of all the steps that take part in the offloading process. To this extent, I firstly provided a categorization of such activities combined with their interactions and assessed the impact on the system.

A simplified view is: the process starts in the host that is interested in offloading a portion of its application (Seeker), at the beginning it has to select the best remote host to satisfy its request (Provider). When the provider has been selected the Seeker forwards the data and eventually the code that must be executed and goes to sleep. It will be waked up from the selected provider when it forwards the results of the offloading or eventually, it will be waked up by a timer indicating that the process takes too long to return the results thus the execution is recovered locally.

Afterwards, an analytical formulation of the AnyRun computing problem where the key costs is provided.

it has been shown that the problem is a combinatorial optimization problem that is notoriously know to be NP-Hard and, due to the absence of a centralized view, traditional solution cannot be used.

I proposed a solution based on the NaïveBayes inference model to select that it has been adapted to select the best device on which to forward the offloading of computation; in this sense, offloading is considered advantageous when there is a reduction of the energy footprint of the smart device.

6.1.3 The ARC Framework

To empirically evaluate the solution I propose for the anyrun computing paradigm, I have developed the ARC framework. This is a novel framework whose objective is to decide whether to offload or not to any resource-rich device willing to lend assistance is advantageous compared to local execution with respect to a rich array of performance dimensions.

The system is built on top of the SCAMPI opportunistic computing framework (Conti et al. [2010a]; Pitkänen et al. [2012])developed within the FP7 EC project SCAMPI (Service Platform for Socially Aware Mobile and Pervasive Computing)[1] and it comprises two key entities:

- *ARC Seeker:* is the device that is in needed to offload the computation, and it generates the offloading request.
- *ARC Provider:* is a device that is willing to accept to take care of the Seeker's computation (that can be either Cloud, Cloudlet, or any other device in the proximity).

[1]http://www.ict-scampi.eu/

The core of my solution is the *Inference Model* which receives a rich set of information about the available remote devices from SCAMPI and employs the information to profile a given device, in other words, it decides whether offloading is advantageous compared to local execution, i.e. whether it can reduce the local footprint compared to local execution in the dimensions of interest (CPU and RAM usage, execution time, and energy consumption). The output of the Inference Engine (that implements the proposed solution to select the best remote-host) is an empirical estimate of the probability that offloading is advantageous. The dynamic and distributed nature of ARC also poses other challenges. In the dissertation I propose a solution to physically move a set of instructions between the devices. This is needed because it is assumed that the Provider already owns the right set of instructions to process the data received and during the offloading it is simply impractical due to memory limitations. In fact, there is no limits of the number of applications and consequently to the line of code that can be remotely-execute in AnyRun domain. This new feature increases the device-reachability of ARC but it also increments the security threats. For this reason, in the dissertation I provided a preliminary solution that addresses the security challenges in AnyRun domain composed by:

- A dynamic authentication mechanism that uses traditional solutions in case the framework selects to use a device located behind an infrastructure (i.e. Cloud/Cloudlet) but it uses a pin-based authentication in case the device is located in the proximity.
- A Sanboxing functionality to blocks undesired access on the provider side, in essence the framework isolates and block the calls that access local information.

The security solution right now is only partial and needs further studies that will be provided in the Uprise-IoT project [2].

6.1.4 Detailed Evaluation of ARC

To empirically evaluate ARC I presented a set of experimental results on cloud, cloudlet, and Opportunistic domain. In Cloud I used the state of the art in cloud solutions over a set of significant benchmark problems and with three WANs access technologies (i.e. 3G, 4G, and high speed WAN). The main outcome is that the cloud is an appealing solution for a wide variety of problems, but there is a set of circumstances where the cloud performs poorly (as discussed before).

[2]http://uprise-iot.supsi.ch

Moreover, in the evaluation I showed the main limitation (in terms of latency) in adopting a cloud-based approach that is strictly connected to the difference between computation and transmission costs. This essentially confirms the results that Cuervo et. al. provided more by Cuervo et al. [2010] more than 4 years ago.

The second part of the evaluation is done in opportunistic/cloudlet scenarios where I used my custom-made testbed to compare ARC and MAUI (Cuervo et al. [2010]), the state of the art in computation offloading. To this extent, I have performed two distinct set of experiments: the first with a cloudlet environment and the second with an opportunistic environment. The key outcome is that ARC matches the performances of MAUI in cloudlet environment, but it outperforms MAUI by roughly 50% to 60% in opportunistic domain.

6.2 Future Directions

The work presented in the dissertation extends the state of art in opportunistic computing and computational offloading strategies and unifies them with an efficient inference mechanism to select the best host in such dynamic domain. The work in the dissertation also extends the state of the art in power measurement proposing a portable power monitor and on privacy preserving mechanisms proposing a tool to automatically build a layer between the applications and the sensitive information accessible on the smart device.

Other possible impacts can be identified and foreseen in the paradigms presented in Section 2 , more specifically:

- *Internet of things and fog computing*: computational offloading and privacy-preserving mechanisms are fundamental in the domain of IoT and fog computing. This kinds of devices are usually very small and powered by an internal battery. Thus, computational offloading solutions may be very useful to reduce their energy footprint. On the other hand, privacy preserving mechanisms are fundamental because sensitive and personal information may be interfered even from unexpected devices.

- *Elastic computing:* in this domain, the system adapts its resources depending on its current load. Traditionally this is done by moving and allocating virtual machine instances, due to the dynamic nature of the system and difficulty in estimating future request rate, the algorithm presented in ARC can be a good candidate to perform such kind of inferences.

- *Public resource Computing:* the solution presented in the dissertation may be beneficial in this domain thanks to the work on code mobility.

Currently, the results of the dissertation have been successfully applied in several applications:

- The Project CHIST-ERA UPRISE-IoT [3] has been started specifically to study the security and privacy issues that arise in smart and dynamic domains. The bottom layer of the solution is the sandboxing solution (mockingbird) proposed in the dissertation. The key goal of the project is to increase security at different levels in both IoT and smart devices with a strong focus on applied cryptography. Authentication solutions in distributed domains will be part of the solution and investigated as well.

- The Power Monitor Solution (POEM) presented as part of the testing environment (Section 4) has been used in several circumstances. It has been the tool used in the CHIST-ERA Macaco Project [4] as the solution to measure and compare the cost of reading the data sources in order to have a clear picture of the impact that the software produced in the project has on the user's device. The solution has also been used from Vanini et al. [2016] as a measurement tool to compare different solutions in the activity recognition domain.

From the work presented I also envision a set of application scenarios that may be a good exploitation strategy in the future, they are:

- *Home Scenario:* ARC may be beneficial to reduce the use of GPUs in virtual reality applications. Currently, virtual reality has become a key feature for all the smartphones, but it heavily impacts the local GPUs and consequently the energy footprint of the device. The key characteristic of this computation is the enormous amount of information processed and the strict time constraints (one frame one for eye usually per time frame-rate). With ARC will it be possible to offload part of the GPU into a connected device (i.e. your computer) to have significant improvements in the devices? lifetime and having better and more realistic images due to the access to powerful hardware resource. This problem is hardly approachable by cloud solutions due to the amount of data exchanged, and for this reason, the solutions currently proposed by the academic community use the cloudlet approach.

[3]http:/urpise-iot.supsi.chj
[4]http://macaco.inria.fr/

- *Cafeteria Scenario*: assume that a cafeteria is interested in acquiring more customers, and decide to implement a new service that allows the users to play their favorite games faster on their mobile devices without depleting the battery. This service may be implemented with means of ARC.
- *Public Transportation Scenario:* nowadays most public transportation like trains or even airplanes offer on-board WiFi to their customers. It is known that the WAN connectivity when traveling is very unstable or even not accessible. I envision a solution where each transportation method carries a cloudlet-like device that performs the computation when it receives the request via ARC. The device may also use ARC to forward the requests to the cloud when the WANs connectivity is favorable (and use ARC to perform dynamically the evaluation).

The key aspects that I am interested in investigating further are security and privacy (thanks to the CHIST-ERA Uprise-IoT project mentioned before). I am interested in performing an empirical evaluation in a real application (i.e. virtual reality use-case illustrated before). Another aspect I am interested in is to include the costs of accessing the WAN in the AnyRun equation. This extension will allow the users to better define their preferences in the solution to adopt for their offloading needs.

6.3 List of Publications

6.3.1 Peer-reviewed Articles

1. Ferrari, Alan, Vanni Galli, Daniele Puccinelli, and Silvia Giordano, . "On the Usage of Smart Devices to Augment the User Interaction with Multimedia Application." *Accepted for publication*, World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2017 IEEE 18th International Symposium on A. IEEE, 2017.
2. Ferrari, Alan, Silvia Giordano, and Daniele Puccinelli. "Reducing your local footprint with anyrun computing." Computer Communications 81 (2016): 1-11.
3. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Code mobility for on-demand computational offloading." World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A. IEEE, 2016.

4. Vanini, Salvatore, et al. "Using barometric pressure data to recognize vertical displacement activities on smartphones." Computer Communications 87 (2016): 37-48.

5. Ferrari, Alan, et al. "Detecting energy leaks in android app with poem." Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on. IEEE, 2015.

6. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Gesture-based soft authentication." Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on. IEEE, 2015.

7. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Characterization of the impact of resource availability on opportunistic computing." Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, 2012.

6.3.2 Other Relevant Publications

Posters and Demos

1. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Poster: Can Smart Devices Protect Us from Violent Crime?." Proceedings of the 21st Annual International Conference on Mobile Computing and Networking. ACM, 2015.

2. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Managing your privacy in mobile applications with mockingbird." Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on. IEEE, 2015.

3. Ferrari, Alan, Daniele Puccinelli, and Silvia Giordano. "Code offloading on opportunistic computing." Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on. IEEE, 2014.

Bibliography

Crawdad, a community resource for archiving wireless data at dartmouth. http://crawdad.cs.dartmouth.edu/, June 2013.

Mohammad Al-Shurman, Seong-Moo Yoo, and Seungjin Park. Black hole attack in mobile ad hoc networks. In *Proceedings of the 42nd annual Southeast regional conference*, pages 96–97. ACM, 2004.

Frances E. Allen. Control flow analysis. In *Proceedings of a Symposium on Compiler Optimization*, pages 1–19, New York, NY, USA, 1970. ACM. doi: 10.1145/800028.808479. URL `http://doi.acm.org/10.1145/800028.808479`.

Luzi Anderegg and Stephan Eidenbenz. Ad hoc-vcg: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 245–259. ACM, 2003.

Hassan Artail, Khaleel W Mershad, and Hicham Hamze. Dsdm: a distributed service discovery model for manets. *IEEE Transactions on Parallel and Distributed Systems*, 19(9):1224–1236, 2008.

Kevin Ashton. That ?internet of things? thing. *RFiD Journal*, 22(7):97–114, 2009.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

Baruch Awerbuch, David Holmer, Cristina Nita-Rotaru, and Herbert Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *Proceedings of the 1st ACM workshop on Wireless security*, pages 21–30. ACM, 2002.

A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting Mobile 3G Using WiFi. In *5th Annual International Conference on Mobile Systems, Applications and Services (MobiSys'10)*, San Francisco, CA, USA, June 2010.

Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

Alastair R Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. Mockdroid: trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 49–54. ACM, 2011.

D. Berry. *Statistics: A Bayesian Perspective*. Duxbury, 1996.

Jan Beutel, Roman Lim, Andreas Meier, Lothar Thiele, Christoph Walser, Matthias Woehrle, and Mustafa Yuecel. The flocklab testbed architecture. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 415–416. ACM, 2009.

Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pages 23–30. IEEE, 2014.

Niels Brouwers, Marco Zuniga, and Koen Langendoen. Neat: A novel energy analysis toolkit for free-roaming smartphones. 2014.

Eric Bruneton, Romain Lenglet, and Thierry Coupaye. Asm: a code manipulation tool to implement adaptable systems. *Adaptable and extensible component systems*, 30, 2002.

Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19th international conference on Software engineering*, pages 22–32. ACM, 1997.

Dimitris Chatzopoulos, Mahdieh Ahmadi, Sokol Kosta, and Pan Hui. Have you asked your neighbors? a hidden market approach for device-to-device offloading. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A*, pages 1–9. IEEE, 2016.

Eric Chen, Satoshi Ogata, and Keitaro Horikawa. Offloading android applications to the cloud without customizing android. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 788–793. IEEE, 2012.

Min Chen, Yixue Hao, Yong Li, Chin-Feng Lai, and Di Wu. On the computation offloading at ad hoc cloudlet: architecture and service modes. *IEEE Communications Magazine*, 53(6):18–24, 2015.

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.

Thomas Clausen, Philippe Jacquet, Cédric Adjih, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, Laurent Viennot, et al. Optimized link state routing protocol (olsr). 2003.

M. Conti, S. Giordano, M. May, and A. Passarella. From Opportunistic Networks to Opportunistic Computing. In *IEEE Communications Magazine*, 2010a.

Marco Conti, Jon Crowcroft, Silvia Giordano, Pan Hui, Hoang Anh Nguyen, and Andrea Passarella. Routing issues in opportunistic networks. In *Middleware for Network Eccentric and Mobile Applications*, pages 121–147. Springer, 2009.

Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine*, 48(9):126–139, 2010b.

Keith D Cooper, Timothy J Harvey, and Ken Kennedy. A simple, fast dominance algorithm. *Software Practice & Experience*, 4:1–10, 2001.

E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *MobiSys'10*, San Francisco, CA, USA, June 2010.

Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 121–135. ACM, 2015.

Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348. ACM, 2011.

Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

David Ehringer. The Dalvik Virtual Machine Architecture. *Techn. report (March 2010)*, 2010.

Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM, 2003.

Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 281–286. IEEE, 2011.

A. Ferrari, D. Puccinelli, and S. Giordano. Characterization of the Impact of Resource Availability on Opportunistic Computing. In *ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC)*, Helsinki, Finland, Aug 2012.

Alan Ferrari, Daniele Puccinelli, and Silvia Giordano. DroidLab: A novel Android based testbed with network emulation. *SUPSI Techn. report (March 2013)*, 2013.

Alan Ferrari, Daniele Puccinelli, and Silvia Giordano. Gesture-based soft authentication. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*, pages 771–777. IEEE, 2015a.

Alan Ferrari, Daniele Puccinelli, and Silvia Giordano. POEM: Portable Open Source Energy Monitor. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 260–265. IEEE, 2015b.

Eugenio Giordano, Lara Codecà, Brian Geffon, Giulio Grassi, Giovanni Pau, and Mario Gerla. Movit: the mobile network virtualized testbed. In *Proceedings of the ninth ACM international workshop on Vehicular inter-networking, systems, and applications*, pages 3–12. ACM, 2012.

Domenico Giustiniano, Nils Ole Tippenhauer, and Stefan Mangold. Low-complexity visible light networking with led-to-led communication. In *Wireless Days (WD), 2012 IFIP*, pages 1–8. IEEE, 2012.

Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. Comet: code offload by migrating execution transparently. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, pages 93–106. USENIX Association, 2012.

Shuai Hao, Ding Li, William GJ Halfond, and Ramesh Govindan. Estimating android applications' cpu energy usage via bytecode profiling. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 1–7. IEEE, 2012.

Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. Konark-a service discovery and delivery protocol for ad-hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 2107–2113. IEEE, 2003.

Nikolas Roman Herbst, Samuel Kounev, and Ralf H Reussner. Elasticity in cloud computing: What it is, and what it is not. In *ICAC*, pages 23–27, 2013.

Theus Hossmann, Franck Legendre, Paolo Carta, Per Gunningberg, and Christian Rohner. Twitter in disaster mode: Opportunistic communication and distribution of sensor data in emergencies. In *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition*, page 1. ACM, 2011.

Chung-Ming Huang, Kun-chan Lan, and Chang-Zhou Tsai. A survey of opportunistic networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1672–1677. IEEE, 2008.

Esa Hyytiä, Suzan Bayhan, Jörg Ott, and Jussi Kangasharju. On search and content availability in opportunistic networks. *Computer Communications*, 2015.

Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Donwon Kim, and Hojung Cha. Devscope: a nonintrusive and online power analysis tool for smartphone hardware components. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 353–362. ACM, 2012.

Teemu Kärkkäinen, Mikko Pitkänen, Paul Houghton, and Jörg Ott. Scampi application platform. In *Proceedings of the seventh ACM international workshop on Challenged networks*, pages 83–86. ACM, 2012.

Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

Rajeev Koodli and Charles E Perkins. Service discovery in on-demand ad hoc networks, 2002.

S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953, 2012. doi: 10.1109/INFCOM.2012.6195845.

Dejan Kovachev, Tian Yu, and Ralf Klamma. Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791. IEEE, 2012.

Ulas C Kozat and Leandros Tassiulas. Network layer support for service discovery in mobile ad hoc networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1965–1975. IEEE, 2003.

Daniel Krajzewicz, Georg Hertkorn, C Rössel, and P Wagner. Sumo (simulation of urban mobility). In *Proc. of the 4th Middle East Symposium on Simulation and Modelling*, pages 183–187, 2002.

Marc Langheinrich. Privacy by design?principles of privacy-aware ubiquitous systems. In *International conference on Ubiquitous Computing*, pages 273–291. Springer, 2001.

Jason LeBrun, Chen-Nee Chuah, Dipak Ghosal, and Michael Zhang. Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks. In *2005 IEEE 61st Vehicular Technology Conference*, volume 4, pages 2289–2293. IEEE, 2005.

Vincent Lenders, Martin May, and Bernhard Plattner. Service discovery in mobile ad hoc networks: A field theoretic approach. *Pervasive and Mobile Computing*, 1(3):343–370, 2005.

Ding Li, Shuai Hao, William GJ Halfond, and Ramesh Govindan. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, pages 78–89. ACM, 2013.

Fei Li, Yogachandran Rahulamathavan, Mauro Conti, and Muttukrishnan Rajarajan. Robust access control framework for mobile cloud computing network. *Computer Communications*, 68:61–72, 2015.

David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2005. ISBN 0521646707. URL `http://www.arctic.umn.edu/perf-book/`.

Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE mobile computing and communications review*, 7(3):19–20, 2003.

Changling Liu and Jörg Kaiser. Survey of mobile ad hoc network routing protocols. 2005.

Wei Liu, Takayuki Nishio, Ryoichi Shinkuma, and Tatsuro Takahashi. Adaptive resource discovery in mobile cloud computing. *Computer Communications*, 50: 119–129, 2014.

Songbai Lu, Longxuan Li, Kwok-Yan Lam, and Lingyan Jia. Saodv: a manet routing protocol that can withstand black hole attack. In *Computational Intelligence and Security, 2009. CIS'09. International Conference on*, volume 2, pages 421–425. IEEE, 2009.

Tim Mackinnon, Steve Freeman, and Philip Craig. Endo-testing: unit testing with mock objects. *Extreme programming examined*, pages 287–301, 2001.

Verdi March, Yan Gu, Erwin Leonardi, George Goh, Markus Kirchberg, and Bu Sung Lee. μcloud: towards a new paradigm of rich mobile applications. *Procedia Computer Science*, 5:618–624, 2011.

Constandinos X Mavromoustakis, Georgios Kormentzas, George Mastorakis, Athina Bourdena, Evangelos Pallis, and Joel Rodrigues. Context-oriented opportunistic cloud offload processing for energy conservation in wireless devices. In *Globecom Workshops (GC Wkshps), 2014*, pages 24–30. IEEE, 2014.

John C McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C Snoeren, and Rajesh K Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conf*, 2011.

Pietro Michiardi and Refik Molva. Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Advanced communications and multimedia security*, pages 107–121. Springer, 2002.

Abderrahmen Mtibaa, Khaled A Harras, and Afnan Fahim. Towards computational offloading in mobile device clouds. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 331–338. IEEE, 2013.

Abderrahmen Mtibaa, Khaled A Harras, Karim Habak, Mostafa Ammar, and Ellen W Zegura. Towards mobile opportunistic computing. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 1111–1114. IEEE, 2015.

Björn Muntwyler, Vincent Lenders, Franck Legendre, and Bernhard Plattner. Obfuscating ieee 802.15. 4 communication using secret spreading codes. In *Wireless On-demand Network Systems and Services (WONS), 2012 9th Annual Conference on*, pages 1–8. IEEE, 2012.

Mirco Musolesi and Cecilia Mascolo. Car: context-aware adaptive routing for delay-tolerant mobile networks. *IEEE Transactions on Mobile Computing*, 8(2): 246–260, 2009.

Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! *SIGPLAN Not.*, 44(3):265–276, March 2009. ISSN 0362-1340. doi: 10.1145/1508284.1508275. URL http://doi.acm.org/10.1145/1508284.1508275.

Hoang Anh Nguyen and Silvia Giordano. Context information prediction for social-based routing in opportunistic networks. *Ad Hoc Networks*, 10(8):1557–1569, 2012.

Hoang Anh Nguyen, Silvia Giordano, and Alessandro Puiatti. Probabilistic routing protocol for intermittently connected mobile ad hoc network (propicman). In *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6. IEEE, 2007.

David Alfred Ostrowski. A scalable, lightweight webos application framework. In *Internet Operating Systems (ICIOS), 2012 IEEE First International Conference on*, pages 5–8. IEEE, 2012.

Andrea Passarella, Mohan Kumar, Marco Conti, and Elenora Borgia. Minimum-delay service provisioning in opportunistic networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(8):1267–1275, 2011.

Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168. ACM, 2011.

Luciana Pelusi, Andrea Passarella, and Marco Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Communications Magazine*, 44(11):134–141, 2006.

Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (aodv) routing. Technical report, 2003.

Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity?a proposal for terminology. In *Designing privacy enhancing technologies*, pages 1–9. Springer, 2001.

M. Pitkänen, T. Kärkkäinen, J. Ott, M. Conti, A. Passarella, S. Giordano, D. Puccinelli, F. Legendre, S. Trifunovic, K. Hummel, M. May, N. Hedge, and T. Spyropoulos. SCAMPI: Service platform for soCial Aware Mobile and Pervasive computing. In *ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC)*, Helsinki, Finland, Aug 2012.

Kiran K Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J Rentfrow. Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proceedings of the 17th annual international conference on Mobile computing and networking*, pages 73–84. ACM, 2011.

Francoise Sailhan and Valerie Issarny. Scalable service discovery for manet. In *Third IEEE International Conference on Pervasive Computing and Communications*, pages 235–244. IEEE, 2005.

M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2004.

Aaron Schulman, Thomas Schmid, Prabal Dutta, and Neil Spring. Demo: Phone power monitoring with battor. In *MobiCom (Conference on Mobile Computing and Networking)*, 2011.

James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. Haggle: A networking architecture designed around mobile users. In *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*, pages 78–86, 2006a.

James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. CRAWDAD trace cambridge/haggle/imote/cambridge (v. 2006-01-31). http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/cambridge, January 2006b.

Pravin Shankar, Yun-Wu Huang, Paul Castro, Badri Nath, and Liviu Iftode. Crowds replace experts: Building better location-based services using mobile social network interactions. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 20–29. IEEE, 2012.

V Shanmuganathan and T Anand. A survey on gray hole attack in manet. *IJCNWC*, 2(6):647–650, 2012.

Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Single-copy routing in intermittently connected mobile networks. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 235–244. IEEE, 2004.

Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

Eli Tilevich and Yannis Smaragdakis. *J-orchestra: Automatic java application partitioning*. Springer, 2002.

Sacha Trifunovic, Franck Legendre, and Carlos Anastasiades. Social trust in opportunistic networks. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–6. IEEE, 2010.

Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

Mueen Uddin, Azizah Abdul Rahman, Abdulrahman Alarifi, Muhammad Talha, Asadullah Shah, Mohsin Iftikhar, and Albert Zomaya. Improving performance of mobile ad hoc networks using efficient tactical on demand distance vector (taodv) routing algorithm. *International Journal of Innovative Computing, Information and Control (IJICIC)*, 8(6):4375–4389, 2012.

Amin Vahdat, David Becker, et al. Epidemic routing for partially connected ad hoc networks, 2000.

Salvatore Vanini, Francesca Faraci, Alan Ferrari, and Silvia Giordano. Using barometric pressure data to recognize vertical displacement activities on smartphones. *Computer Communications*, 87:37–48, 2016.

Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3): 94–104, 1991.

G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a Wireless Sensor Network Testbed. In *4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*, Los Angeles, CA, April 2005.

Isaac Woungang, Sanjay Kumar Dhurandher, Alagan Anpalagan, and Athanasios V Vasilakos. *Routing in opportunistic networks*. Springer, 2013.

Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *USENIX ATC*, 2012.

H. Zhang. The optimality of naive bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, 2004.

Bowen Zhou, Amir Vahid Dastjerdi, Rodrigo N Calheiros, Satish Narayana Srirama, and Rajkumar Buyya. A context sensitive offloading scheme for mobile cloud computing service. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 869–876. IEEE, 2015.