A Task Offloading Framework for Energy Saving on Mobile Devices using Cloud Computing

by

Majid Altamimi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Over the last decade, mobile devices have become popular among people, and their number is ever growing because of the computing functionality they offer beyond primary voice communication. However, mobile devices are unable to accommodate most of the computing demand as long as they suffer the limited energy supply caused by the capacity of their small battery to store only a relatively small amount of energy. The literature describes several specialist techniques proposed in academia and industry that save the mobile device energy and solve this problem to some extent but not satisfactorily. Task offloading from mobile devices to cloud computing is a promising technique for tackling the problem especially with the emergence of high-speed wireless networks and the ubiquitous resources from the cloud computing. Since task offloading is in its nascent age, it lacks evaluation and development in-depth studies.

In this dissertation, we proposed an offloading framework to make task offloading possible to save energy for mobile devices. We achieved a great deal of progress toward developing a realistic offloading framework. First, we examined the feasibility of exploiting the offloading technique to save mobile device energy using the cloud as the place to execute the task instead of executing it on the mobile device. Our evaluation study reveals that the offloading does not always save energy; in cases where the energy for the computation is less than the energy for communication no energy is saved. Therefore, the need for the offloading decision is vital to make the offloading beneficial. Second, we developed mathematical models for the energy consumption of a mobile device and its applications. These models were then used to develop mathematical models that estimate the energy consumption on the networking and the computing activities at the application level. We modelled the energy consumption of the networking activity for the Transmission Control Protocol (TCP) over Wireless Local Area Network (WLAN), the Third Generation (3G), and the Fourth Generation (4G) of mobile telecommunication networks. Furthermore, we modelled the energy consumption of the computing activity for the mobile multi-core Central Processing Unit (CPU) and storage unit. Third, we identified and classified the system parameters affecting the offloading decision and built our offloading framework based on them. In addition, we implemented and validated the proposed framework experimentally using a real mobile device, cloud, and application.

The experimental results reveal that task offloading is beneficial for mobile devices given that in some cases it saves more than 70% of the energy required to execute a task. Additionally, our energy models accurately estimate the energy consumption for the networking and computing activities. This accuracy allows the offloading framework to make the correct decision as to whether or not offloading a given task saves energy.

Our framework is built to be applicable to modern mobile devices and expandable by considering all system parameters that have impact on the offloading decision. In fact, the experimental validation proves that our framework is practical to real life scenarios. This framework gives researchers in the field useful tools to design energy efficient offloading systems for the coming years when the offloading will be common.

Acknowledgements

All thanks and praise is to ALLAH Almighty, the most beneficent, the most merciful, who helped and gave me the ability and knowledge to successfully complete this thesis.

I would like to express my thanks and deep gratitude to my supervisor, Professor Kshirasagar Naik, who was always with me with endless support, encouragement, and friendly discussions. He taught me how to have different thinking strategies and how to have novel solutions. This work would have not been done without his advice and valuable comments.

I would like to thank Professor Krishnaiyan Thulasiraman, Professor Ajit Singh, Professor Otman Basir, and Professor Mahesh Pandey, for the effort to read my thesis and give me their feedback with very valuable comments and suggestions that purify and clarify my thesis.

I sincerely acknowledge the scholarship I received from the King Saud University in Saudi Arabia and its representative here in Canada, the Saudi Culture Bureau, for providing me the opportunity to successfully build my career.

My deepest appreciation, and my grateful thanks go to my father, Lafi, and my mother, Alyah Altamimi, who were the permanent source of love, encouragements, and support. Their prayers are the only reason behind my success. Foremost amongst the individuals to whom I am thankful is my wife, Shatha Altamimi, for her overwhelming love, ceaseless patience, and continuous motivation. Being away from her parents does not prevent her to fill our family with gladness during our long journey to accomplish my goal. My children, Meshari, Moayad, and Refal, are the ones who make my life colourful, enjoyable, and full of fun.

I am indebted to my brothers and sisters, my father-in-law, and my mother-in-law for their support, help, and love.

Dedication

To my parents... I am proud of you
To my wife.. I am with you
To my children... I am for you

Table of Contents

List of Tables

List of Figures

List of Abbreviations

1G	The first Generation mobile system
2G	The second Generation mobile system
3G	The Third Generation mobile system
3GPP	3rd Generation Partnership Project
4G	The Fourth Generation mobile system
ACK	Acknowledgement
ARQ	Automatic Repeat reQuest
CC	Cloud Computing
CDMA	Code-Division Multiple Access
CQI	Channel Quality Index
CTS	Clear to Send
CW	MAC Contention Window
DIFS	Distributed InterFrame Spacing
DRX	Discontinuous Reception
DTX	Discontinuous Transmission
DVFS	Dynamic Voltage and Frequency Scaling
EDGE	Enhanced Data Packet for Global Evolution

FTP	File Transfer Protocol
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communication
GUI	Graphic User Interface
HSDPA	High Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
LAN	Local Area Network
LTE	Long Term Evolution
MAC	Media Access Control
MC	Mobile Computing
MCC	Multimedia Cloud Computing
MCS	Modulation and Coding Schemes
MIMO	Multiple-Input and Multiple-Output
MobCC	Mobile Cloud Computing
NRA	Network Related Applications
PaaS	Platform as a Service
PDC	Japan Personal Digital Cellular
PHY	Physical layer
RLC	Radio Link Control
RRC	Radio Resource Controller

RSS	Received Signal Strength
RTS	Request to Send
SaaS	Software as a Service
SIFS	Short Inter-Frame Space
SIN	Signal to Noise ratio
SISO	Single-Input and Single-Output
SMS	Short Message Service
TCP	Transmission Control Protocol
TDMA	Time-Division Multiple Access
TTI	Transmission Time Interval
UE	User Equipment
UMTS	Universal Mobile Telecommunications Systems
WLAN	Wireless Local Area Network
WNI	Wireless Network Interfaces
WPAN	Wireless Personal Area Network
XaaS	Everything as a Service

List of Notation

$Apps$	Set of mobile device applications
A_n	The application n on the mobile device
OS	Operating System of the mobile device
HM	Set hardware modules of the mobile device
M_r	The module r in the mobile device
x	Number of applications on the device
y	Number of hardware modules
M_{Cm}	Set of computing modules
M_{Tr}	Set of communication modules
M_{Sn}	Set of sensing modules
M_{Ui}	Set of user interface modules
HM_k	Set of hardware modules that are required by the application A
T_M^A	Required time by the module M for the application A
$\rho(t)$	Probability density function of the execution time

μ	Expected value of the execution time
σ	Distribution variance of the execution time
T^A	Total execution time for the application A
T_s^A	Total time of series execution for the application A
T_p^A	Total time of parallel execution for the application A
U	Set of mobile device users
U_j	User j in the set U
C_j	Configurations of the user U_j
B_j	Behaviour of the user U_j
k_{Uj}^A	Number of used modules by the application A if the user is U_j
T_{Uj}^A	Total execution time of the application A if the user is U_j
P_M^a	Active power consumed by the module M
T_{Uj}^A	Total execution time of the module M for the application A if the user is U_j
E_M^A	Energy consumed by the module M for the application A
E^A	Total energy consumed by the application A
E^{Apps}	Total energy consumed by all mobile device applications ($Apps$)
T_M^i	Idling time of the module M
P_M^i	Idling power consumed by the module M
T_M^s	Sleeping time for the module M

P_M^s	Sleeping power consumed by the module M
E^{is}	Energy consumed by idle and sleep modules
E^B	Energy capacity of the mobile device battery
QoS	Quality of Service
QoE	Quality of Experience
m_b	Number of MAC backoff stages
T_s	Total packet transmission time for successful transmission
T_c	Total packet transmission time for collide transmission
T_{RTS}	Transmission times for the RTS packet
T_{CTS}	Transmission times for the CTS packet
T_{ACK}	Transmission times for the ACK packet
T_D	Transmission times for the data packet
L_{max}	Packet payload
H_{MAC}	MAC Header
T_{PHY}	Transmission times for the physical layer header
σ	Physical channel time slot
R_{data}	Data rate
τ	The probability that a node sends a packet at a random time slot
F	Transmission file size in bytes

F_s	MAC frame size in bytes
P_{RX}	Power consumption at receiving
P_{TX}	Power consumption at transmitting
T_{TACK}	TCP acknowledgement transmission time
$N_{d_{ACK}}$	The number of TCP acknowledgements received
T_H	The transmission time for the MAC header
$N_{d_{seg}}$	The number of TCP segments sent without acknowledgements in file downloading
R_s	Server limited data rate
$N_{u_{ACK}}$	The number of TCP acknowledgements received
$N_{u_{seg}}$	The number of TCP segments sent without acknowledgements in file uploading
$E_{3G/4G}$	Energy consumption by the 3G or 4G interfaces
E_{ps}	Promotion signalling energy consumption
E_{trx}	Transmission energy consumption
E_{tail}	Tail energy consumption
P_{ps}	Promotion signalling power consumption
P_{trx}	Transmission power consumption
P_{tail}	Tail power consumption
T_{ps}	Promotion signalling time

T_{trx}	Transmission time
T_{tail}	Tail time
R_{TCP}	Data rate limitation by the TCP
$R_{3G/4G}$	Data rate limitation by the 3G or 4G networks
CWD	TCP congestion window
RTT	Round-trip time
IWD	TCP initial window size
T_{ss}	Time for the TCP slow-start to reach CWD
F_{ss}	The data transferred during the TCP slow-start
γ	The growth of the TCP window size
E_{WNI}	The energy consumed by the wireless network interface
E_{OS}	The energy consumed by the operating system
P_s	CPU static power
P_d	CPU dynamic power
cp	CPU capacitance
v	CPU supply voltage
f	CPU clock frequency
P_c	CPU core power
P_b	multi-core CPU base power

P_{mc}	multi-core CPU total power
P_{cpu}	CPU total power
P_{min}	CPU power at zero utilization
P_{max}	CPU power at 100% utilization
U	CPU utilization
P_{su}	Storage unite power consumption
R_{su}	Storage unite data rate of the writing/reading
T	Expected execution time
I	Total number of application instructions
cc	Number of CPU cycles per instruction
NS	Networking Status
T_u	User threshold time for the offloading
E^A	Application total energy
QoS^A	Application Quality of Service
U_A	Application utilization to the CPU
R^A	Application throughput
E_c	Total energy consumed by the component c
T_c	Total time of using the component c
P_c	Average power consumed by the component c

E_{NT}	Total energy consumed in networking
T_{NT}	Total networking time
P_{NT}	Average power consumption in networking
R	A Network data rate
e	The energy efficiency of a communication module
E^{off}	Total offloading energy
E^{B}	Remaining energy on the device battery
NT_i	The availability binary variable for the network i
CC_j	The availability binary variable for the cloud j
T^{cc}	Total time of the task in the cloud
T^{off}	Offloading threshold time
QoS^{cc}	Cloud application supported Quality of Service
G	The offloading gain in Joule
t_G	Offloading gain in battery life time

Chapter 1

Introduction

These days, the number of mobile devices (*i.e.,* smartphones and tablets) has been growing dramatically, more than other computing devices. The new mobile devices are rich in data resources, such as sensors and camera, and rich in user interfaces, such as speakers and colourful screens. The Internet connectivity gives their users the ability to communicate with each other through social networking and online gaming. Moreover, mobile users can share their daily life with friends and followers by text, picture, or video clip. Accordingly, the mobile devices are noticeably the largest contributors to social networks [1, 2]. As a result, advanced mobile devices are required to handle these functionalities, most of which are known as intensive computing tasks.

However, mobile devices are constrained by their small batteries that store a limited amount of energy. Battery technology is not making a consistent progress with the semiconductor technology in term of increasing the energy density (*i.e.,* Joules per cubic centimetre) that can accommodate the energy consumed by the semiconductor components of a mobile device. Therefore, the major concern for mobile device users is the limited energy capacity of their devices. As limited battery capacity is a significant issue, industry and academic researchers have been extensively addressing the issue from the hardware level up to the application level. Smart batteries, power scheduling, efficient operating systems and applications, efficient graphical user interfaces, energy-aware communication protocols, and task offloading are all examples of these methodologies and techniques [3].

Task Offloading is a promising solution to overcome many of the mobile device limitations, especially the energy limitation [4]. As current mobile devices feature Internet connectivity with fast wireless networks, reaching remote computing resources is practical. In the era of cloud computing, remote computing resources are accessible at anywhere and

anytime. Cloud computing provides its users with virtually unlimited computing resources from its data center [5]. Based on these observations, a mobile device is able to offload any specific computing task to the cloud for remote task execution on the cloud and receive the result with less energy consumption than executing that task on the device itself [6, 7].

However, offloading is a critical technique that depends on the system parameters of both the mobile device and the cloud. For example, offloading is not beneficial if the mobile device consumes energy on offloading the task more than on execute executing it on the device. Therefore, an offloading decision is vital to make the offloading beneficial for mobile devices. This offloading decision engine estimates the required energy in both cases and then decides to whether or not the mobile devices will save energy by the offloading a given task.

The aim of this thesis is to implement an offloading framework that is suitable for implementation in mobile devices and applicable to the cloud computing environment, which provides the offloading capability to mobile devices. We consider all sources of the system parameters and classify them as profiles, where we build the framework based on them. These profiles let the proposed framework to accurately make the correct offloading decision that save energy on mobile devices. In addition, we develop energy estimation models for these profiles to estimate the energy cost of the networking and computing activities that is needed for the offloading decision.

1.1 Research Motivation and Objectives

In the recent years, mobile devices become essential on people life and their usages having relatively a great breadth that includes but not limited to gaming, stock marketing, online media, and communicating. The advanced capabilities of these devices encourage the multimedia (*i.e.,* photo, audio, and video) and their applications to be hosted by mobile devices. However, the main concern for mobile device users is the battery cycle life. While the multimedia is resource intensive application, they drain most of the battery energy. Thus, the users demand solutions that give them the ability to have advanced multimedia applications on their devices.

The communication and semiconductor technologies present a noticeable positive trend in the recent years. The Internet becomes ubiquitous and low-priced, where in contrasts its bandwidth grows dramatically. Wireless communications are one of the most developed technologies in the 21st century. The wireless speed is increasing everyday and wireless access points available almost everywhere. Semiconductor technology supports small and

light devices by high dense electronics, which associates the mobility with much enhanced functionalities. On the other hand, Cloud Computing (CC) is a new computing paradigm that provides unrestricted computing resources to the end-users. The computing resources are provided as services that are available on-demand, anywhere, and anytime. The key feature of the CC is that the end-users have the computing recourses in the form pay-per-use model and have no responsibility for operating or managing the cloud infrastructure.

The offloading technique is a promising technique to overcome the limited energy problem for the mobile devices. Offloading a task from mobile device to CC is a feasible solution with the assistance of the advances on the Internet, wireless communications, and CC.

In this research, we have the following objectives:

1. Introduce the concept of Mobile Cloud Computing (MobCC), which a cloud computing provides its services especially to mobile devices (*Published in* [7]);
2. Develop and analysis accurate mathematical models for the energy consumption of the mobile devices (*Section* 1.2);
3. Examine the feasibility of the offloading from mobile devices to the cloud in case of saving energy, and introduce the concept of Multimedia Cloud Computing (MCC) (*Chapter* 3 and *Published in* [8]);
4. Build mathematical models that estimate the energy cost for the networking activity (*Chapter* 4 and *Published in* [9]);
5. Build mathematical models that estimate the energy cost for the computing activity (*Chapter* 5 and *Published in* [10]);
6. Implement and develop offloading algorithms that perform the offloading decision (*Section* 6.4 and *Published in* [11]);
7. Profile the system parameters and formulate them as mathematical models that allow the offloading algorithms to estimate the energy cost (*Section* 6.2 and *Published in* [12]);
8. Build a comprehensive offloading framework that describe the decision approach for offloading implementation on mobile devices (*Section* 6.3 and *Published in* [12]); and
9. Validate our developed models and the proposed offloading framework by conducting sets of experiments on real mobile devices and clouds using real applications and scenarios (*Chapters* 4-6).

1.2 System Models

At the beginning of studying the energy aspect for mobile devices, it is important to have detailed energy models for the system. These energy models give us the ability to identify how and where the energy is consumed. Furthermore, with the help of these models, we can analyze and develop our proposed system, which is the offloading framework. Therefore, we model the mobile device with respect to the energy consumption. Consequently, we model the energy consumption of an application on a mobile device. Finally, we present the network model for the offloading technique.

1.2.1 Energy Model of Mobile Devices

A user of a mobile device uses the device by a set of applications (*Apps*) that contains x number of applications, which is presented as the set $Apps = \{A_1, A_2, \cdots, A_x\}$, where A_n is the application number n in the *Apps* set. These applications interact with the device operating system (OS). Besides, the OS interacts and manages the device hardware according to the application requests and requirements. The device hardware consists of a set of hardware components that we name each component as a "Module". A mobile device has a set of modules (HM) that contains y number of modules, which can be presented as a set $HM = \{M_1, M_2, \cdots, M_y\}$, where M_r is the module number r in the HM set.

In particular, there are different types of modules to perform different tasks. For example, computing modules (M_{Cm}) are to do the computation tasks; and similarly, the communication modules (M_{Tr}) are to do the communication tasks. These modules are directly connected to the mobile device battery that powers all of these modules. Importantly, the battery is rechargeable to support the device mobility with no need for replacement. Figure 1.1 shows our perspective model of the mobile device hardware and the corresponding energy flow.

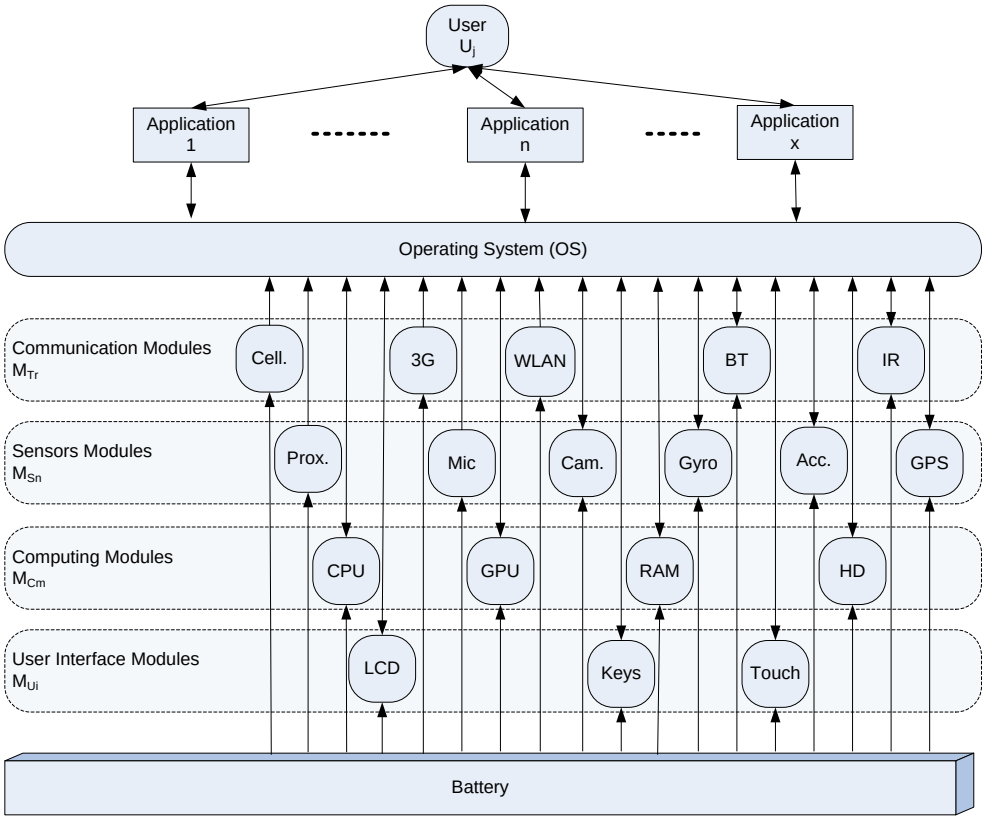


Figure 1.1: The general energy model of a mobile device [1]

1.2.2 Energy Model of Mobile Applications

In this section, we implement and develop an energy model for a general mobile application. Assume an application A requires some of the hardware modules (HM_k) to run properly, where HM_k is a subset of modules from the set HM (*i.e.,* $HM_k \subseteq HM$). That is, the number of modules required by an application A should be less than or equal to the total number of hardware modules ($k^A \leq y$). As a result, there is a number k^A associated with each application. In fact, the application A uses several modules for some time according

[1] *Cell.*: Cellular voice telecommunication technology, *3G*: Third Generation mobile telecommunications, *WLAN*: Wireless Local Area Network, *BT*: Bluetooth communication technology, *IR*: Infrared communication technology, *GPS*: Global Positioning System, *Prox.*: Proximity sensor, *Mic.*: Microphone, *Cam.*: Camera ,*Gyro.*: Gyroscope sensor, *Acc.*: Accelerometer sensor, *CPU*: Central Processing Unit, *GPU*: Graphic Processing Unit, *RAM*: Random Access Memory, *HD*: Hard Disk, *LCD*: Liquid crystal display, *Keys*: Keyboards, and *Touch*: Touch Screen.

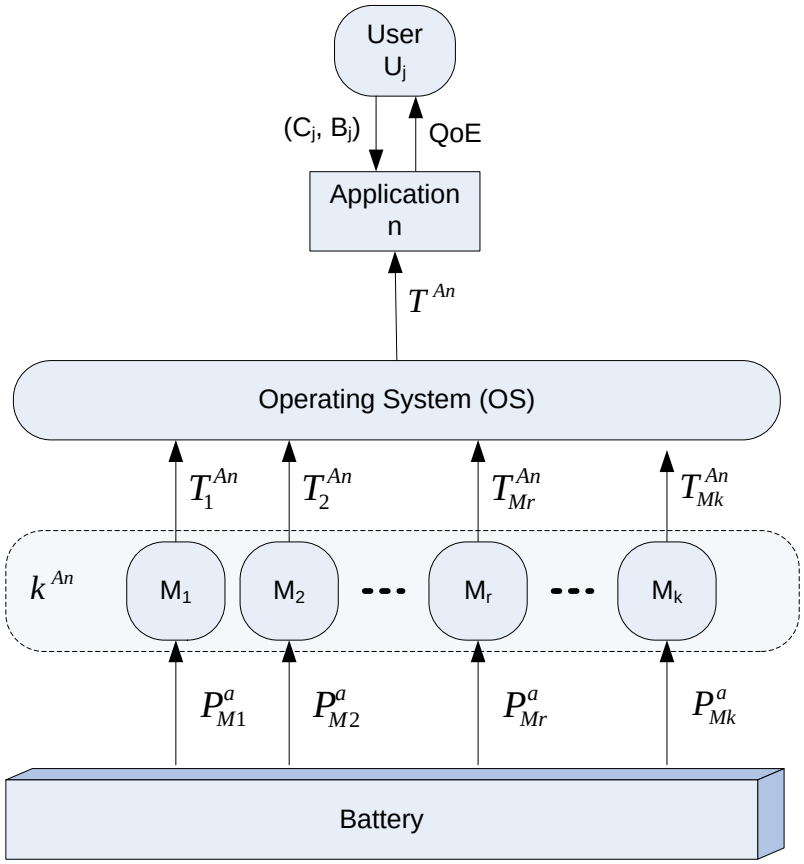


Figure 1.2: The energy model of an application

to its functionalities and requirements. Thus, the application A runs a module M for a time T_M^A, where M is a module in the module set HM_k ($M \in HM_k$). Figure 1.2 depicts this energy model for the application A. In the following subsections, we present the model for the time and the power based on this figure because the energy is function of the time and power.

Time Modelling

It is important to note that the time (T_M^A) is not a constant for a specific application and module and it could be continuous or discrete time (*e.g.,* intermittent periods of time). In particular, it could follow an arbitrary probability distribution function as it has been approved in [13, 14, 15], which defined by the application, the used modules, and

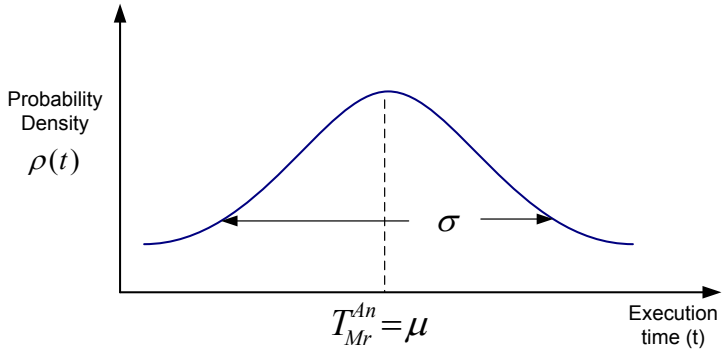


Figure 1.3: Execution time distribution

the user behaviour as we explain next. Figure 1.3 shows an arbitrary probability density function $\rho(t)$ of the execution time, where μ and σ are the expected value and the variance, respectively. For simplicity, we assume the execution time equals the expected execution time of the distribution as expressed in Eq. (1.1).

$$T_M^A = E[T] = \mu \tag{1.1}$$

The total time required by an application A to finish its tasks is given by T^A. In reality, there are series and parallel execution patterns for the modules by the application A. If the application A executes the modules in series as presented in Fig. 1.4, the total time T_s^A equals the sum of time required by all HM_k modules to finish application A tasks.

$$T_s^A = \sum_{r=1}^{k} T_M^A \tag{1.2}$$

On the other hand, if the execution is performed in parallel as presented in Fig. 1.5, the total time T_p^A equals the maximum time required by all HM_k modules to finish application A tasks.

$$T_p^A = \max_{1 \leqslant r \leqslant k} T_M^A \tag{1.3}$$

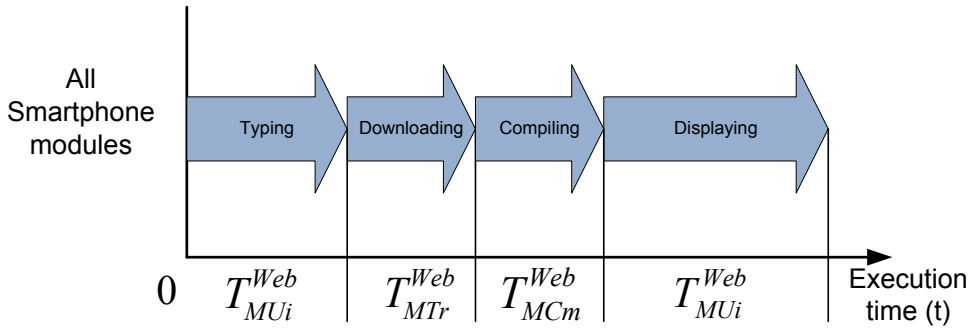


Figure 1.4: Series time execution of the hardware modules for web browsing

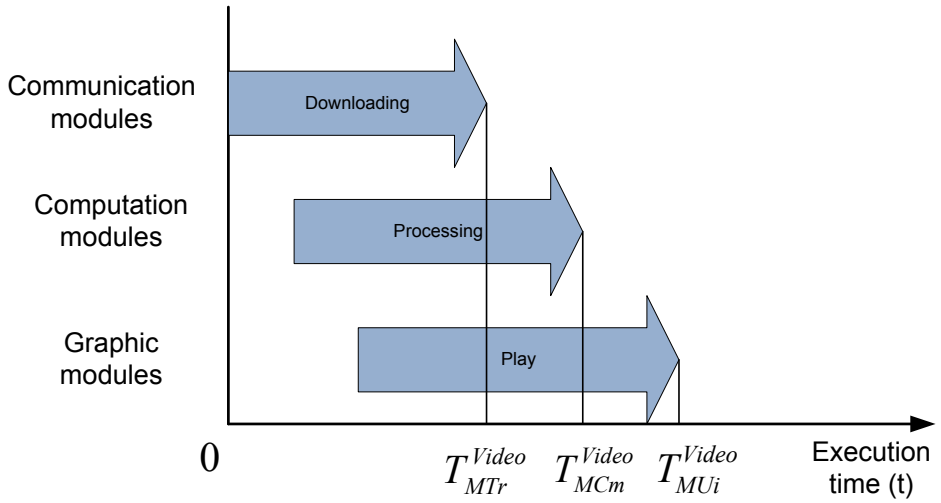


Figure 1.5: Parallel time execution for the hardware modules for video streaming

In general, the application A could execute the modules by these two execution patterns together in a hybrid pattern. Therefore, the total time for an application A equals the maximum of times required by series and parallel execution.

$$T^A = \max\left(T^A_{ss}, T^A_{ps}\right) \tag{1.4}$$

where $T^A_{ss} = \sum_{r=1}^{ks} T^A_M$ and $T^A_{ps} = \max_{1 \leqslant r \leqslant k_p} T^A_M$ are the execution time of modules subsets that run on series and parallel, respectively.

Exploring the energy consumption on a mobile device should consider the user configurations and behaviour since the impact of the user is crucial on battery powered device [16, 17]. Recent mobile devices give their users the ability to configure many of their operating system and applications features for the users' convenience. For instance, a user can set the scale of the screen brightness and can select one favourite wireless data network. In addition, the usage pattern for a user is different from one user to another. For example, some users prefer to listen to music on the mobile device and others prefer to do web browsing and emailing.

Each user can be distinguished by his/her configurations and behaviour. If we represent a user by U_j, then we can present this as a pair of the user configurations and behaviour as $U_j = (C_j, B_j)$ as shown in Fig. 1.2. As a result, we have a set of users $U = \{U_1, U_2, \cdots, U_z\}$, which comprises all possible of combinations (*i.e.,* z) of users configurations and behaviour.

A user U_j configures an application A by enabling and disabling some of this application configurable settings and features [18, 19, 20]. Consequently, the application A enables and disables its modules M^A based on the user U_j configurations. For example, a user could disable the GPS coordination to be embedded in the captured photos where this action prevents the camera application from using the GPS module. Therefore, the actual used modules by an application A are less than HM_k because of the user U_j configurations C_j. We represent the set of actual used modules by M^A_{Uj}, which size equals to k^A_{Uj} where $k^A_{Uj} \leq k^A \leq k$.

On the other hand, the user behaviour affects the execution time of the applications [13, 14]. To illustrate this effect, assume the user U_1 has a preference to watch high definition (HD) videos while the user U_2 has a preference to watch standard definition (SD) videos. As a result, the downloading time of HD video for user U_1 is definitely longer than the time for SD video in U_2 case. Therefore, the behaviour B_j of user U_j is the key factor that shapes the distribution of the execution time of an application. Since the execution time highly depends on the user behaviour we reformat the execution time as a function on the user behaviour and presented as $T^{(A,Uj)}$.

Power Modelling

Assuming a module M consumes instantaneous power equals $p(t)$, and stays for period of time equals T. Then, the average consumed power P by a module for a period of time is given by the following expression.

$$P = \frac{1}{T}\int_0^T p(t)dt \tag{1.5}$$

Importantly, the active power depends on the type of the task. For instance, usually the transmission power is higher than the receiving power for the communication modules. Another example is that the power consumed for LCD depends on the brightness of the screen. Thus, the active power has d discrete levels and takes only one level at time. Therefore, the active power is presented as $P^a = \left\{P^{a1}, P^{a2}, \cdots, P^{ad}\right\}$, where P^{al} is the active power of level l.

Energy Modelling

If the consumed power by a module M is equal to P_M^a at active mode, then the energy consumed (E_M^A) by this module for an application A is given as the required time (T_M^A) by an application A for module M times the active power (P_M^a) of this module as in Eq. (1.6).

$$E_M^A = T_M^{(A,Uj)} \times P_M^a \tag{1.6}$$

The total energy consumed E^A for finishing application A tasks is equal to the sum of all energy consumed by the modules required by this application as expressed in the following equation:

$$\begin{aligned} E^A &= \sum_{r=1}^{k_{Uj}^A} E_M^A \\ &= \sum_{r=1}^{k_{Uj}^A} T_M^{(A,Uj)} \times P_M^a. \end{aligned} \tag{1.7}$$

Table 1.1: Power consumption of a hardware module at different states

Modules state	Power consumption (mW)	
	Single module	Aggregate for all modules
Sleep	3 - 31	68.6
Idle	7 - 80	268.8
Active	50 - 927	320 - 1355

As a result, the energy consumed by all applications can be driven as given by next equation as a sum of energy consumed by each applications.

$$E^{Apps} = \sum_{n=1}^{x} E^{An} \tag{1.8}$$

The previous models are results for the analysis of the modules in the active state. To complete our energy models, we extend the energy models to consider others operating states as follows. We assume that the OS manages the modules in an energy efficient way. At this case, the modules switch to inactive mode, which is either idle or sleep mode. The idle mode occurs when a module is fully powered but not receiving any task. In contrast, the sleeping mode occurs if a module is powered partially but can not receive any task. To illustrate this amount of power consumption, Table 1.1 shows the average power consumption measurements for a mobile device module [21]. For further measurements on power consumption on mobile device modules, see [22, 21].

The module M stays in idle mode for time equals T_M^i and consumes power equals P_M^i, then it switches to sleep mode and stays for time equals T_M^s and consumes power equals P_M^s. Accordingly, the total energy consumed by the modules for inactive mode (*i.e.,* E^{is} : Energy in idle and sleep modes) is given in Eq. (1.9).

$$E^{is} = \sum_{r=1}^{y} \left(T_M^{(i,Uj)} \times P_M^i + T_M^{(s,Uj)} \times P_M^s \right) \tag{1.9}$$

Finally, the total energy consumed by a mobile device equals the energy consumed by the modules in both state of active and inactive. Since the mobile device is powered by a battery, the energy capacity should accommodate the energy required by the hardware modules as described in Eq. (1.10). Figure 1.6 shows an example of the power breakdown of mobile devices where the system power is the sleep power P^s in our analysis, idle power is P^i, and others for modules at active power P_M^a [13].

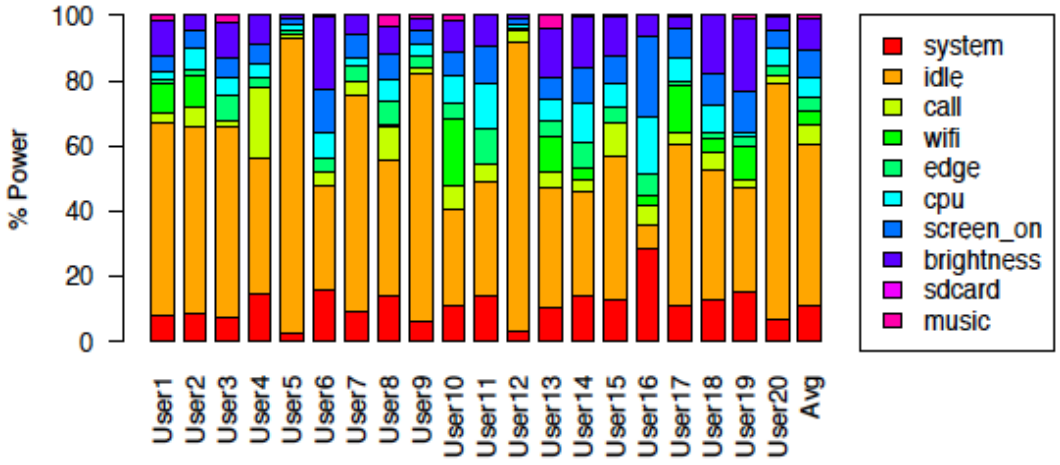


Figure 1.6: Power breakdown of mobile devices

$$
\begin{aligned}
Supply &= Demand \\
E^B &= E^{Apps} + E^{is} \\
&= \sum_{n=1}^{x} E^A + \sum_{r=1}^{y} \left(T_M^{(i,Uj)} \times P_M^i + T_M^{(s,Uj)} \times P_M^s \right) \\
&= \sum_{n=1}^{x} \sum_{r=1}^{k} \left(T_M^{(A,Uj)} \times P_M^a \right) + \sum_{r=1}^{y} \left(T_M^{(i,Uj)} \times P_M^i + T_M^{(s,Uj)} \times P_M^s \right) \qquad (1.10)
\end{aligned}
$$

where E^B is the usable energy capacity of the mobile device battery.

1.2.3 Network Model

Our network model consists of two major parts: mobile devices and cloud computing where both are linked to the Internet, as depicted in Fig. 1.7. The mobile devices are connected to the Internet through a Wireless Local Area Network (WLAN) (*i.e.,* WiFi) or a cellular data access point (3G/4G) (*i.e.,* HSDPA/LTE). These mobile devices provide computing and communication functionalities to the end-users by the mobile applications. For instance, the users can play/record a video or audio, and show/capture photos. On the other hand, the cloud provides the end users with all computing functionalities such as

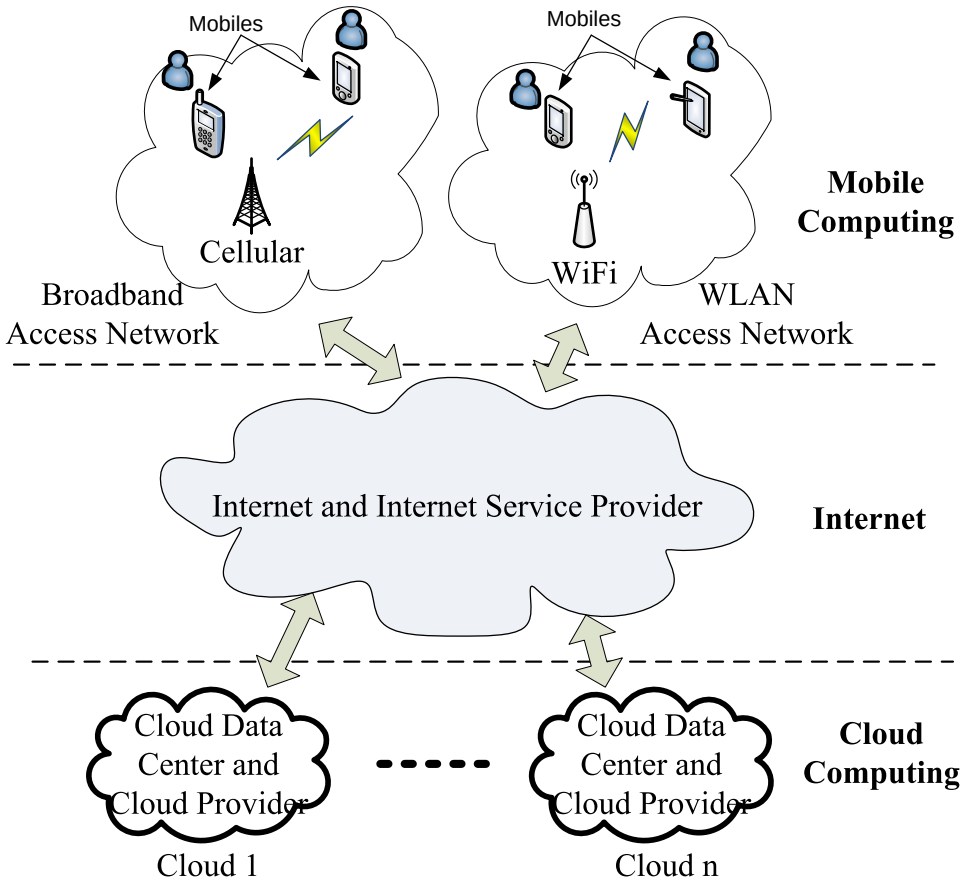


Figure 1.7: The network model

storage and processing from its data center. Moreover, the cloud has a capability to deal with task offloading and assumed it has applications that perform the same tasks as the mobile applications do.

1.3 Problem Definition

Mobile devices are becoming increasingly popular because of their capabilities and functionalities. With powerful operating systems (*e.g., Windows Mobile, Android, Apple iOS, BlackBerry, and Symbian*), mobile devices are able to run heavy applications that are almost similar to desktop computer applications. There is a growth of multimedia appli-

cations on mobile devices because of the capabilities of mobile devices such as compact camera, microphone, and relatively wide and colourful screen [15]. Multimedia applications occupy from 7% to 21% of mobile usage [14]. Nevertheless, these applications drain most of mobile device battery [23].

Multimedia applications are well known as intensive applications in terms of resources such as processing, memory usage, and communication. Due to the limited energy capacity of the mobile battery, mobile developers set rules on the device multimedia capability to limit the energy consumed by these type of applications. Because of these rules, the mobile device battery would last longer but with limited multimedia capabilities. For instance, a mobile device can render only a narrow range of multimedia file formats. An example of this is the *HTC Nexus One*, which can recognize and play only *H.263*, *H.264*, and *m4v* video formats [24]. If the user of this device wants to watch a video in *Flash Video (flv)* format for example, which is very common to website hosting, it is obligatory to convert the *flv* to another format supported by this device. The same difficulty applies to other video formats as well. Moreover, applications that process the multimedia contents are very rare due to their massive energy consumption. Video and audio encoding, which converts media files from one format to another, speech recognition, which converts spoken words to text, and face detection, which allocates and identifies the human face, are all examples of multimedia processing applications.

In recent years, the advance in battery technology has not kept pace with other technologies. On the other hand, the communications, software, and semiconductor technologies that are involved in the mobile device show great progress. According to Moore's law, which describes the long-term trend in the history of computing hardware, the number of transistors on the integrated circuit can double every two years. In contract, the battery capacity increases by 5% every year [25]. This difference in the pace of technologies produces a gap between energy demand and supply as depicted in Fig. 1.8 [26]. In fact, this gap grows by 4% annually [27].

Reducing the energy consumption of the mobile devices has been studied extensively and many methodologies and techniques have been proposed. Smart batteries, power sleep mode, power scheduling, efficient operating system and applications, efficient graphic user interface (GUI), and redesign and implement energy-aware communication protocols, all are examples of these methodologies and techniques. For an in depth survey, see this comprehensive report [3].

In the era of the cloud computing, most of the mobile device constraints can be eased off by offloading heavy applications from the mobile to the cloud [4]. Cloud computing has virtually unlimited computing resources such as processing and storage. With current

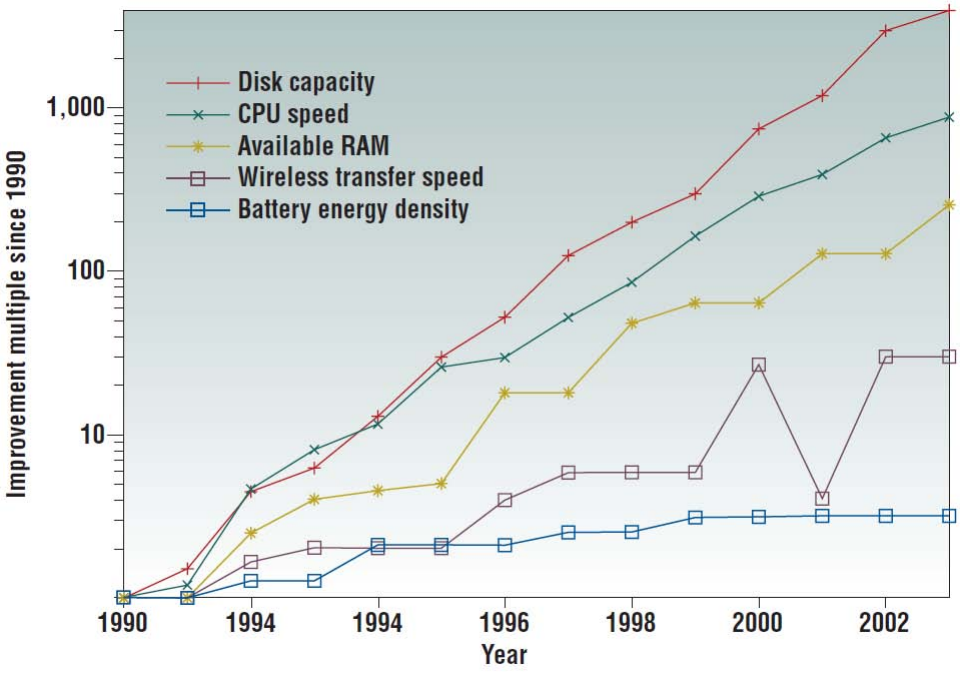


Figure 1.8: The trends of the mobile device technologies

increasing and development on the wireless and the Internet bandwidth, the offloading is promising to overcome the problem of energy capacity in mobile devices. Mobile devices can take the cloud capabilities to request a multimedia processing in an efficient way. For example, a mobile can upload a video file to the cloud, and then request from the cloud to convert that file into a desired format fitting the device capability. Thus, task offloading to the cloud is promising as ICT technology to fill the gap between demand and supply energy of the mobile devices.

Filling the gap by offloading mobile applications to the cloud introduces a new area that has to be investigated, studied, and examined for good insights into the feasibility of this technique. Task offloading from mobile devices to the cloud is essential to enhance their computing capabilities and at the same time save their battery energy. However, task offloading introduces another challenge due to the impact of system parameters on the offloading performance. This challenge is to perform the offloading decision correctly before executing the offloading processes. An accurate offloading framework allows mobile devices to obtain the correct decision as to whether or not to perform the task offloading based on the system parameters at the time of making the decision. In this work, we tackle this challenge by developing an offloading framework that gives mobile devices the ability to make the correct offloading decision accurately. The framework accomplishes the correct decision after it collects all of the system parameters that have direct influence on the offloading decision.

1.4 Thesis Contributions

The focus of this thesis is to solve the mobile device energy problem in a feasible solution with the help of other systems in Information and Communication Technology. Integrated Mobile Computing (MC) with the Cloud Computing (CC) is a feasible option to overcome the energy problem of the mobile devices. We introduce this integration by taking the increasing of the Internet bandwidth as an advantage for the offloading technique. With the existence of the CC, the energy constraint of the mobile devices can be relaxed by offloading heavy tasks from the MC to the CC. An example of heavy task is the video encoding where there is no existing encoding efficient application for mobile devices. Since the offloading to CC is in its nascent state and undeveloped, at least to the best of our knowledge, it is important to understand whether the CC extends the battery life or not. Then, we develop an offloading framework that allows to save energy by the offload heavy tasks. The following subsections highlight our contributions in the thesis.

Before that, we would like to mention to the fact that the mobile devices include smart-

phones, tablets, and some mobile gaming and music devices. Nevertheless, the mobile devices came to be known as the smartphone, though the two seem to be used interchangeably in this work. In fact, the smartphone represent the cutting-edge device among these devices, which is the most who suffers from limited the energy problem. Therefore, we use the word smartphone hereafter in many places; mostly, when we talk about a specific mobile device that we conduct experiments on it.

1.4.1 Evaluate Offloading Energy Costs

At the beginning of studying the offloading technique, we address the problem of running multimedia video applications on mobile devices, and we investigate the benefit of using offloading framework in this regard. We present an extensive evaluation of the energy costs of mobile devices and setup a large number (more than a hundred) of experiments on mobile devices to measure their energy for running multimedia applications. Furthermore, we experimentally evaluate the energy cost on mobile devices when the offloading technique is used. This evaluation has been conducted on a real mobile device and cloud.

Our results give researchers much insight into the energy cost of such applications, which is important to implement offloading algorithms. Indeed, the measurement of energy costs in this study helps the developers of mobile devices and cloud to design efficient algorithms that save mobile device energy. The results show that the cloud provides the mobile devices with more functionalities and save mobile device energy from 30% to 70%. To our knowledge, this is the first study to evaluate energy costs of applications on mobile devices connected to the cloud. Specifically, by the means of experiments, we show the following:

- We measured the energy costs for playing an online video and show the different phases of energy consumption, such as ‘download only’, ‘download-and-play’, and ‘play only’.
- As energy consumptions vary with time, we have measured the costs of sending and receiving file over HTTP and FTP protocols via 3G and WLAN interfaces, and present the statistics of the results.
- We investigated whether or not mobile devices save energy by using cloud encoding service. This investigation is done by evaluating the energy costs for uploading and downloading a video file to and from cloud using HTTP and FTP protocols though 3G and WLAN connections. Then, we compare the results with the energy costs of doing video encoding on the mobile device for the same video.

- In the aforementioned investigation, we consider two broad experimental scenarios related to the location of the original file to be encoded.

Chapter 3 corresponds to this contribution that we published in [8].

1.4.2 Develop Energy Models for the Offloading Framework

Modelling the energy consumption of the mobile device is essential for any development for energy saving techniques. For this reason and because of the lack of detailed energy modelling in the literature, we build a comprehensive mathematical energy model for mobile devices. Such modelling is relatively undeveloped for the mobile devices, where this fact motivates us to develop such modelling.

The accurate model provides the best control and management on the energy consumption. This is because awareness of the energy consumption parameters gives understanding where the energy consumed. The offloading technique is critical because it depends on many factors such as application and network characteristics. Each application and network has several parameters that affect the energy consumption and consequently the offloading decision. Develop and implement accurate offloading framework, which makes the accurate decision, needs precise modelling to the energy consumption. Thus, we model the energy consumption on a mobile device for both of networking and computing activities. These models estimate the energy consumption of a given task in the case of offloading it to the cloud and the case executing it on the device to allow the framework make the correct decision.

Chapters 4 and 5 correspond to this contribution that we published in [9] and [10], respectively.

1.4.3 Propose and Develop Offload Framework

After we develop the models for the energy consumption and for the offloading technique, we propose a framework for the offloading technique. This framework describes the offloading processes and algorithms to make the correct offloading decision. We consider an accurate offloading engine that make the offloading decision supported by the developed models. Figure 1.9 shows our proposed framework for task offloading that we develop, implement, and validate.

Chapter 6 corresponds to this contribution that we published in [11] and [12].

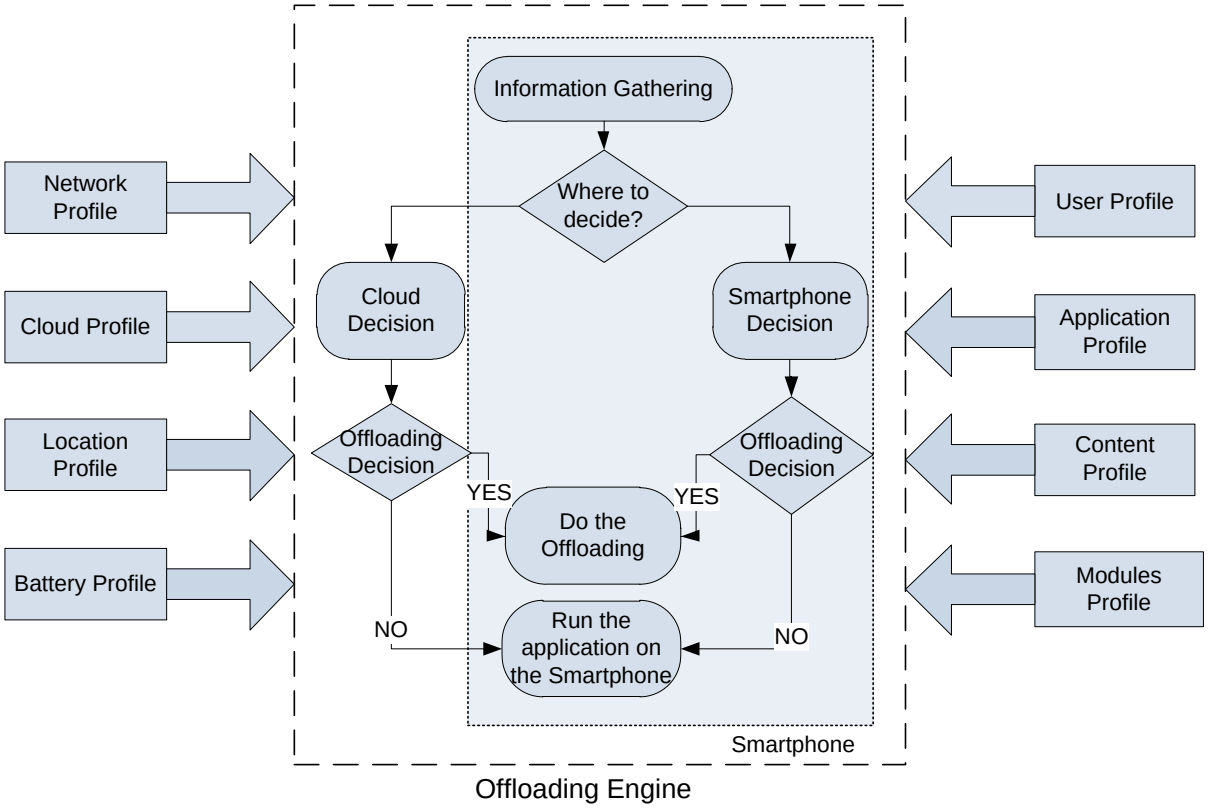


Figure 1.9: Our offloading framework

1.5 Thesis Organization

The rest of the dissertation is organized as follows. Chapter 2 presents the background and the literature review of the smartphone as a cutting-edge mobile device, cloud computing, and offloading techniques. The evaluation results for the offloading to CC are shown in Chapter 3. Chapters 4 and 5 address our development detail and procedure to model the energy consumption of a mobile device on the networking and computing activities, respectively. Our proposed offloading framework is described and validated in Chapter 6. This dissertation concludes in Chapter 7 in addition to the future research.

Chapter 2

Background and Literature Review

In this chapter, we provide a background and a literature review on the smartphones, cloud computing, offloading technique, and the offloading for smartphones. This chapter is divided into four sections as follows. In the first section (2.1), we deliver a brief history about smartphones and how they become important devices in the people life nowadays. The second section (2.2) gives a wide view about the cloud computing, and its services and implementations. We describe the offloading techniques in detail in the third section (2.3). Finally, we review the literature of using the offloading technique especially to overcome smartphone constraints in section (2.4).

2.1 Smartphones

In this section, we present an overview on the importance of smartphones as they promise to be the future of the mobile computing.

2.1.1 History of Smartphones

Mobile devices have been started as voice devices at early '80 using analog radio telephone system that is knowing as the first generation (1G) mobile system. Then in early '90, the technology moves up to the Second Generation mobile system (2G), which reshapes the entire mobile system. The 2G succeeded in implementing the digital communication in the mobile system. Indeed, it was implemented in four main systems: Global System for Mobile Communication (GSM), Time-Division Multiple Access (TDMA), Japan Personal Digital

Cellular (PDC), and Code-Division Multiple Access (CDMA one). In this generation, data packet services were started such as Short Message Service (SMS) using circle-switch data. For instance, the SMS provides a very low data rate for short messages where this service represents the birth of data communication in addition to the voice communication in the mobile systems. After that in early 2000, the data services have been upgraded by engaging General Packet Radio Service (GPRS), which introduces the 2.5 Generation (2.5G). Soon after this upgrade, the Enhanced Data Packet for Global Evolution (EDGE) has been engaged. This last engagement introduces what is called 2.75 Generation (2.75G).

With the success of the 2G evolutions, more demands were increasing on high data rate, network capacity, and frequency bandwidth. These demands drive to the 3rd Generation (3G), which is developed into two systems. The first system is known as the Universal Mobile Telecommunications Systems (UMTS) and CDMA2000, where the second system is known as High Speed Downlink Packet Access (HSDPA). Today, the success of the 3G is remarkable since the traffic of 3G data is increasing between %300 to %700 every year [28].

In parallel to mobile system evolution, other wireless technologies such as Wireless Local Area Network (WLAN) and Wireless Personal Area Network (WPAN) find their way into mobile devices. For example, WLAN becomes popular service in many areas like homes, campuses, coffee shops, airports, and hotels. The main driver for these technologies to become common on the mobile devices is that they provide higher data rate with much less cost than the cost of mobile system data. Hence, today many places provide free access to their WLAN. Note that, the WLAN was invented as an extension to LAN and it does not involve in the mobile system evolution. The primary difference between mobile systems and other wireless system is that the mobile communication systems have coverage range in few kilometres while the WLAN in few tens of meters and WPAN in few meters [29].

The advance in the mobile device technologies have played a role in the growing of the mobile phones. In addition to the significant advance in the communication technology as discussed above, advances in the hardware and software of mobile devices have the major effects. The advances in the hardware of the mobile phone add auxiliary feature to mobile devices such as Global Positioning System (GPS). Likewise, the advances on the semiconductors technology lead to smaller and lighter mobile device and add more capabilities and features. On the other hand, advances in the software support the mobile phones with Internet services (*e.g.,* web browsing, e-mailing, gaming, and office productive applications).

The convergence of the mobile communication systems, which are the Internet, the mobile computing, and the multimedia broadcasting, forms a new ubiquitous Information

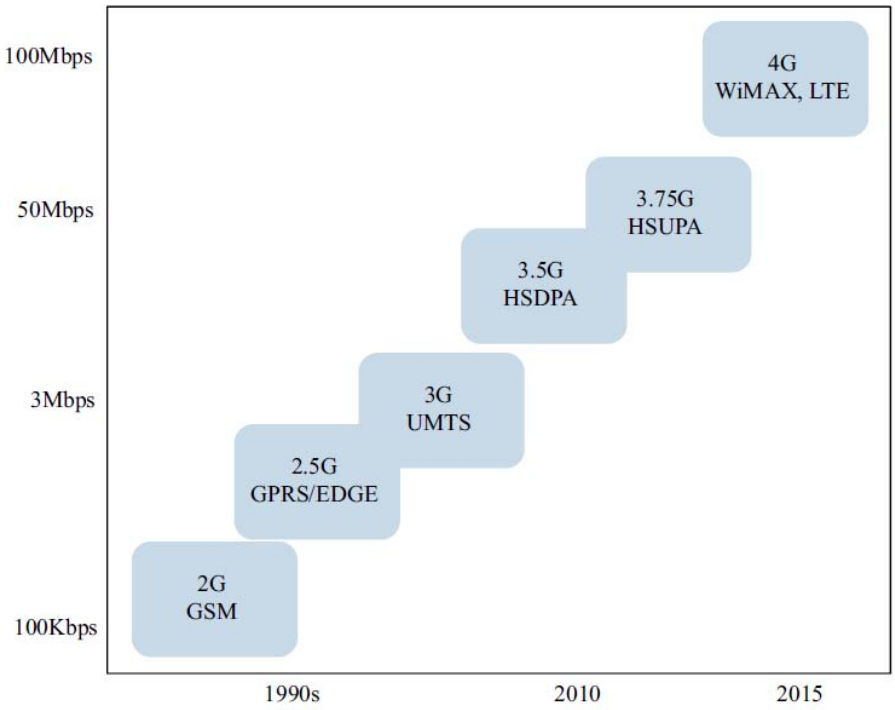


Figure 2.1: Mobile technologies evolution

and Communication Technology (ICT) system that is noticeable these todays and it will be common in the near future. The Fourth Generation (4G) mobile communication system is the most important step toward this convergence. The 4G road map is to reach all-IP architecture where this road reduce the infrastructure cost, expansible, and accommodating for future services on the computing, broadcasting, and the Internet. Moreover, the all-IP architecture supports a seamless connectivity among heterogeneous networks called true Internet mobility [30]. The 4G integrates all global networks and all terminal types in one mobile environment [31]. In this evolution, the mobile phones move from voice centric devices to Internet Protocol (IP) centric devices [29, 31, 32, 33].

The overall view of the mobile systems evolutions is shown in Figure 2.1 [33]. In addition, Table 2.1 summarizes the evolution in the mobile systems and lists some key characteristics of each system [31, 32].

Table 2.1: Summary of mobile system generations

Technology	1G	2G	2.5G	3G	4G
Design Began	1970	1980	1985	1990	2000
Implementation	1984	1991	1999	2002	2010
Services	Analog voice	Digital voice, SMS	Packetized data	Broadband data	Completely IP-based
Standards	NMT, AMPS, Hicap, CDPD, TACS, ETACS	GSM, iDEN, D-MPS	GPRS, EDGE	WCDMA, CDMA2000	Single standard
Data Bandwidth	1.9 kbps	14.4 kbps	384 kbps	2 Mbps	200 Mbps
Multiplexsing	FDMA	TDMA, CDMA	TDMA, CDMA	CDMA	CDMA
Core Network	PSTN	PSTN	PSTN, Packet network	Packet network	Internet
Coverage Area	Large	Medium	Medium	Small	Smaller

2.1.2 Smartphone Definition

The word smartphone (n.) has been added to Oxford dictionary in 2007, and it is defined as

> Any various telephones enhanced with computer technology. Now specially, is a type of mobile phone, which incorporates the functions of palmtop computer, personal digital assistant, or similar device.

As well, the mobile phone contains many intelligent functions; it starts to be called a smartphone.

2.1.3 Smartphone as a Source of Data

Smartphone become an important source of data in the current ICT. People use smartphones to create and edit files, capture pictures, and recode videos and audio using the embedded camera and microphone. Moreover, smartphones are able to provide their location information using GPS technology. For example, current smartphone embeds the location information on the captured image to create location-based albums. Today, smartphones contain many types of sensors: three-axis gyroscope, accelerometer, digital compass, proximity sensor, and ambient light sensor [34]. Each sensor is a source of data, which enhance smartphones functionalities and capabilities.

The hardware and software of a smartphone support the capability to generate a huge amount of data in many formats and types. Unfortunately, smartphones are able to generate data more than their capabilities. For instance, smartphones can capture pictures but hardly do a processing on them such as face recognition or reformatting. In very rare cases, smartphones are able to do some of these processing but they need an intensive processing which drain their battery.

2.1.4 Smartphone Usages

The usages of smartphones are widely varying world wide from kids gaming to stock marketing and president campaign. Social networks applications systems strengthen the need for such smartphone capability [29]. These features become necessary and not auxiliary any more. Furthermore, smartphone finds its way on business activities such as banking and trading in the stock market. That is because the smartphone fits the needs for

many people who need mobility, and computing on the same handheld device. All what is needed on the packet that is much smaller then the regular paper notebook [35]. Examples of smartphones capabilities are having soft-media (*e.g.,* web news and TV), travel e-ticket, e-shipping, e-print, online gaming, build in credit card, etc [36, 37, 38, 39, 40]. All of these capabilities enforce the needing for the smartphone to become a popular mobile device.

2.1.5 Multimedia on Smartphones

With enhanced of smartphones capabilities, smartphones can access vast amount of Internet contents. The social networks and news sites occupy around 80% of web sites [41]. Moreover, most of the web access comes from the mobile devices. In 2008, the number of access from mobile phones beat the number of access from desktop computers [41]. This is predictable as long as wireless speed grows the more people use the mobile and become essential on their lifestyle [29, 42]. A similar study by Nielson company research finds that Australians spend longer time on the Internet if they have faster Internet [43]. That could be people move from other media like TV to online media like YouTube [35]. The mobile usage for the Internet contents, such as news and social sites, grows rabidly in the recent years according to Gartner Group's report [1]. Most importantly, this report argues that the percentage is escalating for the multimedia contents (*i.e.,* image, audio, and video). This means the multimedia on the web has high growing relatively to textual contents.

2.2 Cloud Computing

In this section, we present an overview of the cloud computing (CC) and its services, and we provide some useful definitions of the cloud computing.

2.2.1 An Overview on Cloud Computing

Cloud computing is a new emerging computing paradigm. Cloud computing is Internet-based computing services that are provided on-demand to the end-users. Cloud computing provides its virtually unlimited resources of its infrastructure as services with minimum effort needed by end-users. That is, the end-users of the cloud computing use cloud services with no need to maintain, manage, or operate the cloud infrastructure. On the contrary, cloud computing providers maintain, manage, and operate the cloud to keep the cloud services run efficiently, seamlessly, and friendly for the end-users.

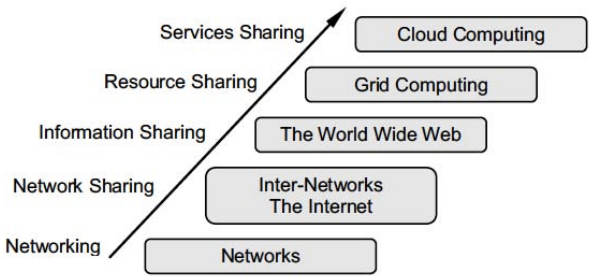


Figure 2.2: Evolutions to cloud computing

Cloud computing is built on a physical data centre that work as the infrastructure for the cloud. The cloud data centre is a large-scale of server clusters, hard-disk storage arrays, and local high-speed networks [44, 45]. It uses the Internet as a media to deliver its services. As a result, it adopts the Internet protocols and offers all types of its services over these protocols [46].

Cloud computing becomes into the reality as a result of advances in the networking, semiconductor, and computing technologies. In the networking and semiconductor technologies, the growing in the Internet speed at the core of the Internet network and at the end-users terminal gives the end-users the opportunity to access the cloud from the Internet. In the computing technologies, the virtual computing, distributed computing, and utility computing all enable the data centre to power the cloud functionalities and provide the cloud services [47]. These technologies are not new but their convergence together forms the cloud computing. However, the cloud computing is new in terms of accessing the computing resources on-demand and the accountability of the utility usage for paying-as-used basis. Cloud computing is a convergence of many ICT technologies such as grid computing, utility computing, service oriented application, and computing virtualization. It is the outcome of the evolution in the ICT technologies [5]. Figure 2.2 shows the evolutions that lead to the CC [48].

2.2.2 Cloud Computing Definition

The term cloud come from the early using and drawing of cloud to represent the telephone networks and then used for the Internet core network. The first introduction to the cloud computing was at 60's [49].

Since the cloud computing is in its early stages, the definition is debatable [46, 50]. A comprehensive view of cloud computing definitions is presented and discussed in [50]. The

National Institute of Standards and Technology defines the cloud computing as

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.,* networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [51].

On the other hand, Cisco defines cloud computing as

> IT resources and services that are abstracted from the underlying infrastructure and provided "on-demand" and "at scale" in a multitenant environment [52].

2.2.3 Cloud Computing Services

The keys of CC are the virtualization of the computing and the scalability of the provisioned services in an efficient way, where many users can share the same resources pool in a scalable manner and in high availability. For that reason, the CC has to maintain the quality of service (QoS) and quality of experience (QoE) of its services to the end-users. The scalability means the cloud can scale up or down its resources as required to accommodate their users requests. Besides, the availability means the cloud is available whenever the end-users request the cloud services. The characteristics of cloud computing is surveyed in [53].

The CC provisions several services of computation over the Internet. It can mainly provision [47, 54]:

- Software as a Service (SaaS),
- Platform as a Service (PaaS),
- Infrastructure as a Service (IaaS), and
- Everything as a Service (XaaS).

SaaS is the cloud service that provides the end-users with software applications, which are usually web-based applications. The end-users in this case use either the conventional

Table 2.2: Examples for cloud computing and its services

Cloud Services	Type of offerings	Examples
SaaS	• Virtual desktop • Applications as web sites • Office productivity • Clients Apps	• IBM Blue Cloud • Microsoft Exchange • Google Doc • Cisco WebEx & Weboffice
PaaS	• Apps servers • File sharing • Database	• Amazon SimpleDB/S3 • Google AppEngine • Saleseforce.com • GigaSpaces • Microsoft Azure • SunCloud
IaaS	• VLAN networks • Logical disks • Virtual servers	• Amazon Elastic Compute Cloud (EC2) • GoGrid • Flexiscale • Mosso • Microsoft Live Mesh

web browser or a specific client to gain this service. **PaaS** is the cloud service where the end-users can define and develop their own operating platform that executed by the cloud servers in a virtual computing environment. **IaaS** is the cloud service where the users have full access to the cloud infrastructure in case to implement their own platforms and applications. **XaaS** is the representation for all or some of the previous services. Examples of cloud computing include: *Amazon Elastic Cloud, IBM Blue Cloud, Microsoft Windows Azure , Google Doc and EnginApp, Salesforce.com,* and *Encoding.com.* Table 2.2 lists the cloud services and their outcomes with some existing examples of present developed cloud computing.

2.2.4 Cloud Computing Implementations

The cloud computing is deployed and Implemented in one of the following models: private, community, public, or hybrid [51, 47]. In private deployment model, the cloud infrastruc-

ture is provisioned for individual user that could be an organization or a company. It may be owned, operated, managed, and controlled by the user or by a third party. The deployment of community cloud is similar to the private model except that the provisioning is for a group of users who have similar interests. In the public deployment model, the cloud infrastructure is provisioned for general public users and only owned and operated by cloud service providers. The hybrid cloud deployment is formed by combining two or more models but keeps each model distinguishable.

We have to mention that the term cloud computing used hereafter for the public cloud computing where the service provide over the public Internet [49, 46].

2.3 Offloading

In this section, we provide a background view about the offloading technique. Furthermore, we discuss the literature of the offloading techniques, the offloading to the cloud computing, and the offloading to save energy for the mobile devices.

2.3.1 Offloading Definition

The first introduction to the offloading concept was in early at 1970s for load balancing between servers of a cluster. Offloading in general is defined as

> The process or technique that is used to improve the performance, quality, or efficiency of a computation task by delegating this task completely or partially to a remote computing machine that is usually has a powerful computation capability more than the local machine.

The computation capability could be in one or more of computation forms such as processing, memory, storage, execution time, and energy consumption.

2.3.2 Offloading Techniques

Based on the definition, we should note that offloading a task requires communication between local machine and remote machine. This reveals that the communication is the key player for the offloading technique. However, this communication consumes energy

as well as the local processing. As a result, there is a trade-off between processing a task locally and offloading it with respect to the energy saving. Technologies include semiconductors, software, and communication could shift the trade-off point. Note that the processor and the network interface are the most power consumer in a mobile device. Therefore, the trade-off or the comparison is difficult and the offloading must be aware of the computation cost and communication cost.

There are several purposes to do the offloading in general [55]. Firstly, as it is invented, the offloading is used for load balancing between servers of a cluster. Secondly, increasing the response time of an application or reducing the execution time. For instance, Wolski et al. presented a framework for the computation grid to make offloading decision based on the network bandwidth [56]. A scheduler performs the decision by predicting the cost of running a task locally and remotely plus the cost of moving and gathering the task to and from remote machines. Thirdly, the offloading can increase the quality of an application because that the results of a powerful machine definitely are better than if they produced from less computation power [55, 6]. Fourthly, energy saving could be gained from the offloading technique for energy limited devices such as smartphones [57, 58]. In general, the offloading technique substitutes many requirements of computing resources. Offloading technique not only satisfies one benefit but also could provide more than one benefit at the same time. For instance, it could provide quality improvement side by side with the energy saving benefit.

2.3.3 Offloading to Cloud Computing

Kumar et al. [4] introduce the concept of computation task offloading to CC to attempt to save mobile energy. They mathematically model the computation cost to run on a mobile device and on the cloud. Based on their model, the factors that effect the offload decision have been investigated. For a given computation task, the energy consumption on a mobile device is given as

$$E_m = P_c \times C/M, \tag{2.1}$$

where P_c is the power consumed by the device in Watts, C is the required computation instructions to perform this task, and M is the speed of the mobile processor in instruction per second. If this task is decided to be offloaded, then there is D amount of data have to be sent and received to and from the cloud. For simplicity, the amount of data and the power consumption to send and receive offloaded task was assumed be similar. The energy cost of offloading transmission (*i.e.,* send and receive) is given in the following equation

$$E_{tr} = P_{tr} \times D/B, \tag{2.2}$$

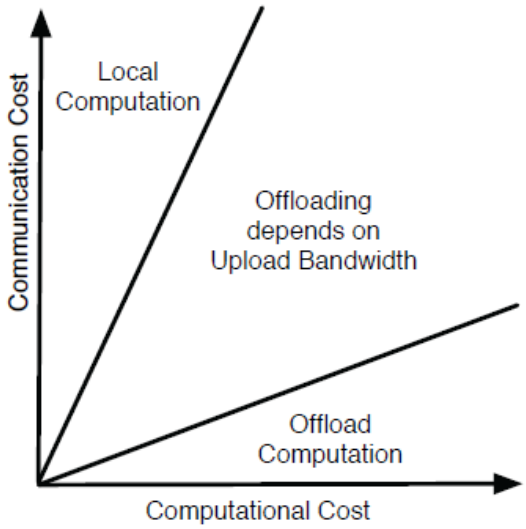


Figure 2.3: Offloading decision areas

where P_{tr} is the power consumed by the device to transmit the offloading data D, and B is the bandwidth of the wireless network.

Assuming the mobile stay idling during the cloud processes the offloaded task, then the energy of idling is giving as

$$E_i = P_i \times C/S, \tag{2.3}$$

where P_i is the mobile idle power, and S is the speed of cloud processing.

The offloading saves mobile energy if the sum of energy that consumed during transmission and idling is less than the energy consumed for running the task on the mobile. That is mathematically is expressed as the following inequality

$$E_m > E_{tr} + E_i. \tag{2.4}$$

Based on this criterion, Fig. 2.3 shows the general decision areas for whether or not to offload [59]. For instance, the offloading is beneficial if the computation C is large while the amount of transmission data D is relatively small. This work concludes that the decision of offloading highly depends on the network bandwidth B, the amount of offloading transmission data D, and the computation of the task C.

Similarly, Lagerspetz et al. [60] analyze and measure the energy trade-off for offloading a task to cloud computing. The analysis follows the same approach of Kumar et al. in [4].

The experiments conducted on *Nokia N900* for indexing a text file. A desktop computer is used on this experiment as a server on a cloud. The experiments show that the offloading is beneficial and save mobile energy up to 97.6% if the WLAN is used for sending the data and there is no receiving of the data again. Authors conclude that the trade-off decision as always offloads if there is small transmission data and heavy computation task.

J. Kim presents six different software architecture patterns to overcome the constraints on the mobile device [61]. These patterns are standalone, full offloading, partial offloading, SaaS-based, CaaS-based, and offloaded CaaS-based. Those patterns are proposed to provide different quality, performance, and energy consumption for the mobile device. An analysis and mathematical model are presented for these patterns. According to the author experimental results, the standalone architecture is energy efficient if the application size is small. On the other hand, if the application size is large, the full offloading is the efficient architecture. The partial offloading architecture is efficient if the size of the mobile part application is small and the amount of data exchange with the cloud is small.

Miettinen et al. consider the opportunity to offload mobile device task to CC in case of saving mobile energy [25]. An analysis is presented similar to the work in [4] for the trade-off between energy cost of mobile application and energy cost of offloading communication. The experimental results reveal the effect of different bearers and different location from the provider transceiver. Moreover, the power consumption is highly affected by the network (or wireless environment) quality. This work proofs that the video encoding is the most intensive computing task.

2.3.4 Offloading Frameworks

Researchers have been studying the offloading problem and proposed several solutions covering application models, offloading objectives, architectures, and approaches [62, 55, 63, 64, 65]. We classify the proposed frameworks in the literature and discussed an example framework from each class that are shown in Fig. 2.4.

Yang et al. propose an offloading service to offload heavy mobile application to nearby powerful PC called surrogate [73]. The offloading service is modelled and prototyped. The purpose of this study is to reduce the response time of the application, where the cost of the offloading is considered for the CPU, memory, and communication. However, the energy cost not evaluated but the authors mentioned to these cost could positively save mobile energy. Moreover, authors propose and study several offloading schemes such as standalone, full offloading, and partial offloading.

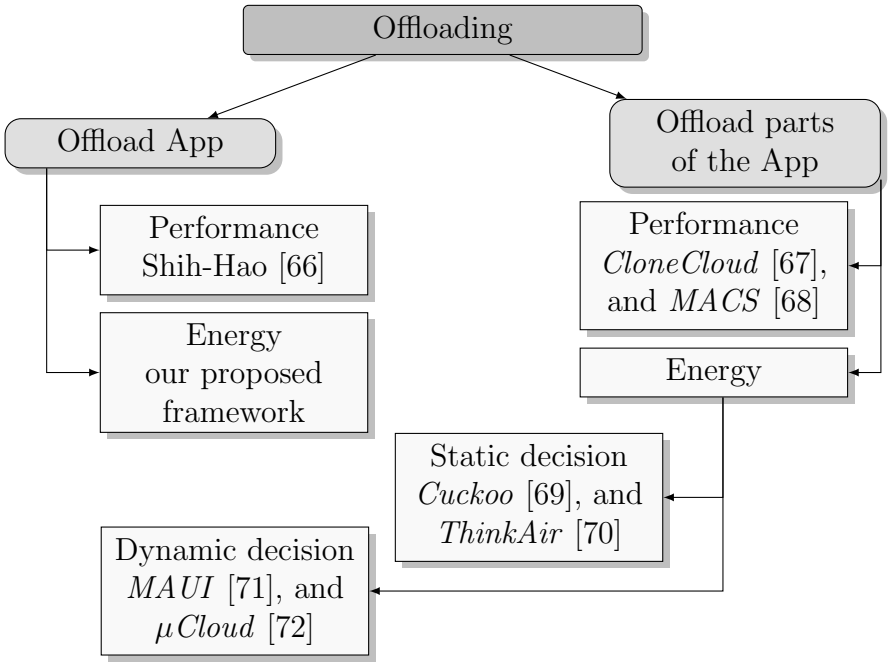


Figure 2.4: Research area of the offloading framework

In like manner, the offloading is proposed to save energy by it does not always save energy on the mobile device [8, 74, 4]. Application partitioning is suggested to offload parts of the application to make offloading beneficial by balancing the energy costs of communication and computation [75, 73]. Application partitioning is not considered in our work, but the proposed framework is a generic one to include application partitioning. However, application partitioning needs some kind of an optimization solver, which adds computation overhead on the offloading decision and reduces the offloading efficiency [76, 63].

Next, we discuss the proposed frameworks shown in each class we classified in Fig. 2.4. *MAUI* [71] and *CloneCloud* [67] are dynamic offloading frameworks that improve application performance and save energy on mobile devices. These frameworks divide an application into parts called methods and decide what optimum solution saves the maximum amount of energy under the current networking condition; the optimum solution identifies the methods that should be offloaded to the cloud. *MAUI* provides a fine-grained energy saving offloading by periodically re-solving the formulated problem to adapt to the changes in the networking environment. However, the overhead of periodically solving the problem can weaken much of the effort to save energy. Moreover, this approach lacks to

models that predict the energy cost of the application methods in the case of offloading and on device execution. In addition, *MAUI* and *CloneCloud* do not consider scalability and software composition, whereas these features are addressed in μCloud [72].

ThinkAir addresses *MAUI's* aforementioned lack of scalability by creating virtual machines (VMs) of a complete smartphone system on the cloud, and removes the requirements on applications inputing environmental conditions that *CloneCloud* induces by adopting an online method-level offloading [70]. Moreover, *ThinkAir* firstly provides an efficient way of performing on-demand resource allocation, and secondly exploits parallelism by dynamically creating, resuming, and destroying VMs in the cloud when needed.

Roelof et al. [69] proposed a framework to offload smartphone computation to a remote cloud. The proposed framework is called *Cuckoo*, which is based on *Android* platform. *Cuckoo* gives the ability to *Android* applications to decide dynamically whether to offload a part of an application to remote cloud or execute it locally. *Cuckoo* can be integrated with a development tool to handle both local and remote code. The authors also provide a programming model for Cuckoo to application developers. The offloading decision is performed based on context information that excluded to the remote resources are reachable or not.

Dejan et al. [68] proposed an offloading framework called Mobile Augmentation Cloud Services (*MACS*) that enables *Android* applications to adaptively execute on the cloud. The framework is based on *Android* platform and considers application portioning in which the framework decides which partitions of the application to execute locally and remotely on the cloud. The offloading decision is formulated as an optimization problem according to device and cloud parameters, such as remaining energy on the battery and the connection bandwidth between the device and the cloud. In contrast to *Cuckoo*, *MACS* allows dynamic application portioning whereas *Cuckoo* only allows a static partitioning at the time of generating the application package.

Shih-Hao et al. [66] proposed a framework for mobile application execution on the cloud which is built on the *Android* platform. The authors consider user protection against eavesdropping from cloud providers by allowing the framework to take advantage of isolated environment on cloud virtual machine. Moreover, the framework is developed to be transparent and controllable by the mobile users. However, this framework is limited to *Android* platform.

However, those frameworks are limited to *Android* platform and need to be generalized to other mobile device platforms. Moreover, those frameworks lack a theoretical view and they just focus on the practicality of the framework. For this reason, the authors choose *Android* because its model is well understood and easy to manipulate. In contrast to those

works, we build a generic offloading framework that considers all system parameters. Most importantly, our proposed framework is applicable to perform the offloading for any mobile device and cloud platform.

2.4 Energy Saving Techniques for Mobile Devices

Reducing mobile device energy consumption has been studied extensively and many methodologies and techniques have been proposed. Smart batteries, power sleep mode or power scheduling, efficient operating system and applications, efficient graphic user interface (GUI), and redesign and implement energy-aware communication protocols, all are examples of these methodologies and techniques [3]. In this section, we narrow down our survey to external assistance for the smartphone like proxy server and CC. We believe that the new computation paradigm like CC promises to be a powerful extension for the mobile computing.

2.4.1 Cloud Computing for Mobile Devices

Recently, mobile devices contain a wide variety of applications. However, there are some limitations of using these applications due to mobile device limited resources and capabilities such as CPU, memory, and connection bandwidth. An essential solution moves heavy task that beyond the mobile device capability to somewhere else that compensate the required capabilities. Cloud computing has most of these requirements. CC that provides these requirements for the smartphones and in general for mobile devices is the key to introduce the concept of Mobile Cloud Computing (MobCC) [77, 7]. MobCC tackles the energy problem for the smartphones since it has context aware capability to serve mobile computing.

The MobCC has special characteristics to provide each cloud service as we see in Figure 2.5. For example, to provision a SaaS for the mobile device, less connection speed is needed than provision IaaS; since in IaaS provisioning, the connection carries all traffic from consumer platform to cloud infrastructure.

Wireless connection is the basic aspect in the mobile devices. This kind of connection has special requirements that are summarized as following [77, 78]:

- Low data rate seamless connectivity,

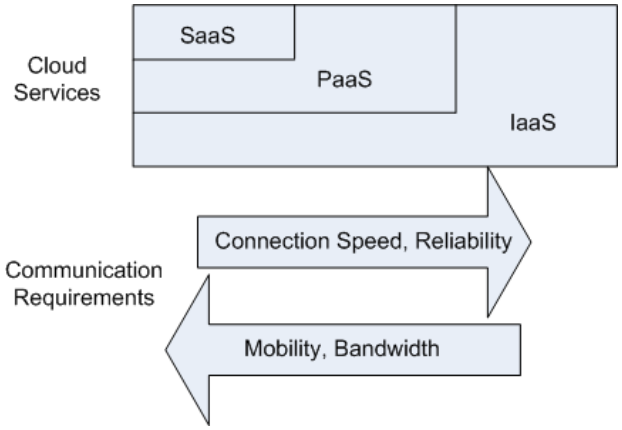


Figure 2.5: Communication requirements for cloud services

- On-demand availability and bandwidth scalability,
- Network and interface selection, and
- Context-aware connectivity.

2.4.2 Saving Mobile Device Energy by the Offloading

Othman et al. presented the early study for task offloading to save energy [79]. In the literature, much work has been reported in the field of energy saving in smartphones using the offloading techniques. We categorize the offloading approaches into three major classes: *(i)* using the cloud computing [80, 4, 45]; *(ii)* using power-aware web proxy [80, 3]; and *(iii)* using local powerful servers [73, 56, 81, 57]. In our study, we adopt the CC approach for task offloading and we focus our study on the approaches that involve task offloading to servers on the Internet in general.

Zhao et al. propose offloading schemes to offload the video encoding from the mobile to a nearby powerful server [81]. This study focuses on the video applications since they are computationally heavy. There are some applications do video decoding and encoding before playing the video on the mobile device. Authors choose the H.264 video encoder because it has the most advantages for the mobile devices such has it provides high compression. They use the same mathematical model as in the previous works [4, 60]. Experimental results show the offloading of video encoding is beneficial. Because the H.264 encoding contains many modules, authors measure the exaction time for each module. Based on

these measurements, they introduce three offloading schemes: search-remote scheme, inter-remote scheme, and remote scheme. The search-remote scheme is to offload the inter-mode-decision module to the server. The inter-remote scheme is to offload inter-frames module to the server. Finally, the remote scheme is to offload the entire encoding application to the server. The experiments reveal that all of the proposed offloading schemes save the mobile energy. In detail, remote scheme saves more energy than the inter-remote scheme and the inter-remote scheme saves more energy than the search-remote scheme.

Kelenyi et al. proposed an offloading technique where the cloud servers are used as a *BitTorrent* client to download torrent pieces [80]. A mobile device requests the torrent process from the cloud to make the cloud do this task behalf of the mobile device. This process offloads the *BitTorrent* download from the mobile device to the cloud. During the time of cloud downloading the *BitTorrent*, the mobile device switches to sleep mode until the cloud finishes the torrent processes and upload the torrent file in one shot to the device. This strategy saves the energy of smartphones because downloading torrent pieces from torrent peers consumes more energy than downloading a one burst of pieces from the cloud.

Kumar et al. [4] introduce the concept of computation task offloading to CC to attempt to save mobile energy. They mathematically model the computation cost to run on a mobile device and on the cloud. Based on their model, the factors that effect the offload decision have been investigated. This work concludes that the decision of offloading highly depends on the network bandwidth, the amount of offloading transmission data D, and the computation of the task.

Baliga et al. studied the energy cost for three cloud computing services: SaaS, PaaS, and IaaS [45]. This study considers the total energy consumption on the user end, networking and provide simple models for the expected energy consumed in each part of the system.

Besides these work, several issues have been addressed fro the offloading to save mobile devices energy. As an illustration, choosing the optimum amount of data to be offloaded has been addressed in the literature [61, 73, 82, 83, 71, 57], where the application is divided into parts and offload some parts that keep the offloading inequality valid. On the other hand, selecting the most energy efficient communication module and data center has been studied in several works [56, 71]. A comparison is provided in their work between the energy consumption for the system parts to show the impact of each part on the overall system consumption.

Chapter 3

Evaluating Offloading Energy Costs

In this chapter, we present an evaluation study to examine the feasibility of the offloading technique to save smartphone energy. We explain the methodology of our experiments and discuss the statistics of our experimental results. We show the results of our experiments, which we conduct for two major parts: network related application experiments and cloud application experiments. The following sections give the details. The contributions of this chapter are published in [8].

3.1 Preamble

The aim of this chapter is to examine the feasibility of the offloading in order to save smartphone energy. The experiments are performed to investigate whether or not a smartphone saves energy by offloading to cloud computing (CC). Our results give researchers much insight into the energy cost of such applications, which is important to implement offloading techniques. Indeed, the measurement of energy costs in this chapter helps the developers of CC to design efficient offloading that save energy of smartphones. The results show that CC provides the smartphones with multimedia functionalities and saves smartphone energy from 30% to 70%. To our knowledge, this is the first study to evaluate energy costs of applications on smartphones connected to real CC. Specifically, by the means of experiments, we show the following:

- We measured the energy costs for playing an online video and show the different phases of energy consumption, such as 'download only,' 'download-and-play,' and 'play only' (Section 3.3).

- As energy consumptions vary with time, we have measured the costs of sending and receiving file over Hypertext Transfer Protocol (HTTP) and File Transfer Protocol FTP via 3G and WLAN interfaces, and present the statistics of the results (Figs. 3.2-3.5).
- We investigated whether or not smartphones save energy by using CC. This investigation is done by evaluating the energy costs for uploading and downloading a video file to and from CC using HTTP and FTP protocols though 3G and WLAN connections. Then, we compare the results with the energy costs of doing video encoding on the smartphone for the same video (Figs. 3.10-3.13).
- In the aforementioned investigation, we consider two broad experimental scenarios (Figs. 3.8 and 3.9) related to the location of the original file to be encoded (Figs. 3.10-3.13).

3.2 Methodology

A comparison study of energy cost for smartphone applications over different hardware smartphones is presented in [20, 19]. This comparison demonstrates that not all smartphones are comparable with respect to configurations, but they exhibit the same kind of energy cost behaviour for each application. To illustrate this, Fig. 3.1, which is from our work in [20], shows the pattern of power consumption of some smartphones in downloading and playing multimedia file. We choose *Android OS* as smartphone platform since it has the biggest market share in the smartphone industry. We use *Android* based *HTC Nexus One* as it is popular, easy to access its battery contact pins, and full of multimedia and communication functionality. Hence, we use this smartphone in all our experiments for the consistency of our experiments.

For general smartphone battery usage, we study the power consumption instead of energy because the power gives a good insight the device consumption regardless of the file size or the time required to finish a task. However, the total energy is used to show a comparison between specific tasks as we see next. This is because the total energy is more meaningful metric to compare one particular task executed on two different processing rates. In addition, we measure the speed of the network interface to demonstrate the obtained data rate at the user level.

The smartphones access the cloud via the Internet and the smartphone applications that are connected to the cloud are considered as Network Related Applications (NRA).

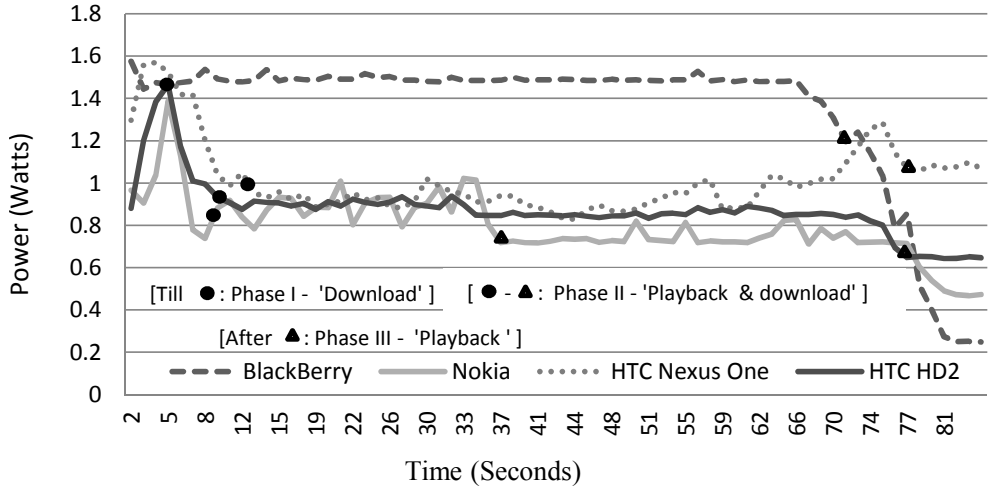


Figure 3.1: Smartphones power consumption for progressive download via WLAN

At the beginning of studying NRA, network interfaces (*i.e.,* 3G and WLAN) are considered because each of these interfaces has its own characteristics, such as coding overhead. As a result, each network interface consumes unequal level of power and provides different data rates. Nevertheless, the Internet protocols (*i.e.,* HTTP and FTP) should be taken into account for NRA. In other words, the network interfaces and the used protocols are the major factors that impact the energy costs of these applications. Thus, we examine the energy costs of common network interfaces and Internet protocols for the smartphones.

For NRA multimedia applications there are two scenarios in downloading the content from the cloud through the Internet: *(i)* progressive download, and *(ii)* download-and-play. In progressive download, the process can be divided into three phases: *(a)* download from the server; *(b)* when enough data is in cache, application starts playback; and *(c)* playback continues after the download is complete. On the other hand, download and play consists of two parts: *(a)* download the entire file from the server to the smartphone storage; and *(b)* play the content form its local storage.

As it is understood, a smartphone is a set of hardware and software. The hardware subsystems of a smartphone include CPU, RAM, HDD, sensors, and Wireless Network Interfaces (WNI); besides, the software includes the operating system and applications. The energy is consumed in a smartphone by these hardware parts, which are managed by the operating system based on the needs of the applications. As a result, each application requires a specific amount of energy depending on the services that are required by each hardware part. In reality, these services are highly affected by the application settings

and configurations. Moreover, the users interact with smartphone applications in different ways. Therefore, the same application leads to different amounts of energy consumption, see Section 1.2.2. This is because of the large space of application settings, configurations, and number of users interaction change the application response and use of the hardware components [84, 20]. For this reason, we provide our results at the user level and avoid breaking down the energy consumption for each hardware subsystem. Our results present the overall energy or power consumptions by the applications since our focus in this study is to compare the total energy consumed by the offloading process to the cloud and the total energy consumed in smartphone to process a multimedia file.

3.3 Experiments on Network Related Application

We conduct sets of experiments to measure the power consumption and the obtained data rate for uploading and downloading over HTTP and FTP protocols using the 3G and WLAN interfaces. The results are shown in Fig. 3.2-3.5. Figures 3.2 and 3.3 show that statistics of the power consumption for the examined Internet protocols over the 3G and WLAN interfaces in the upload and download cases, respectively. Similarly, Figs, 3.2 and 3.5 show the statistics of the experimentally obtained data rate at the application level for different Internet protocols over 3G and WLAN interfaces in the case of file uploading and downloading, respectively.

We use box plot to represent the statistical details of the result. The box plot represents from the bottom to the top the following statistics: the smallest observation, lower quartile ($Q1$), median ($Q2$), upper quartile ($Q3$), and the largest observation. From these figures, we observe that the FTP protocol supports higher data rate with less power consumption than the HTTP protocol. By this aspect, the FTP protocol is more efficient for task offloading.

From the energy saving point of view, the performance metric for network interfaces and protocols is the energy consumption per byte. Table 3.1 displays the summary from above figures the average energy consumption per byte obtained from our experiments.

On the other hand, we perform experiments for each of downloading scenarios, which are download-and-play and progressive download, that are directly have effect on the offloading technique. In download-and-play scenario, we configure the client to download a file from the Internet and we measure the energy costs at the download time. The statistics of the power consumption up to this point is shown in Figs. 3.2 and 3.3. After the download is completed, we allow the media application to play the downloaded file while we read the

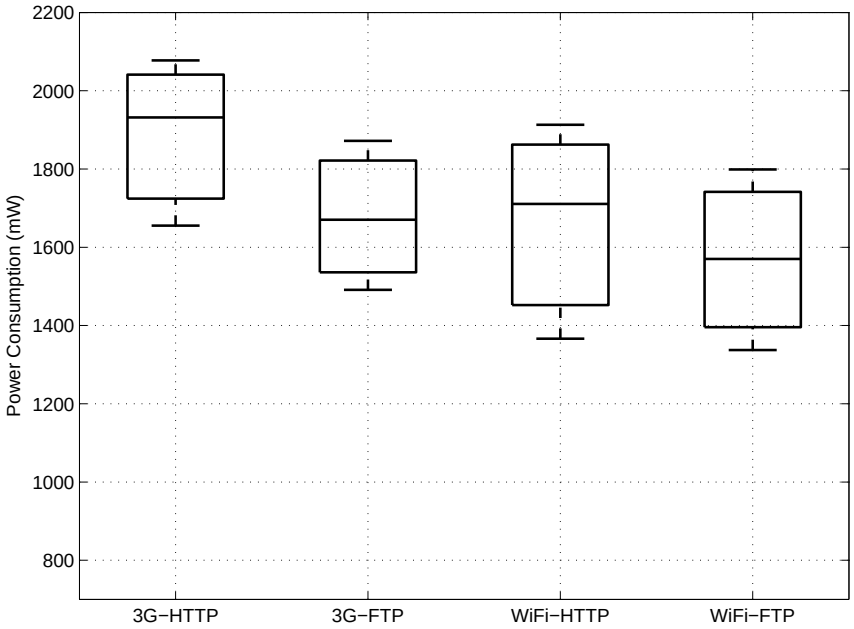


Figure 3.2: Upload power consumption

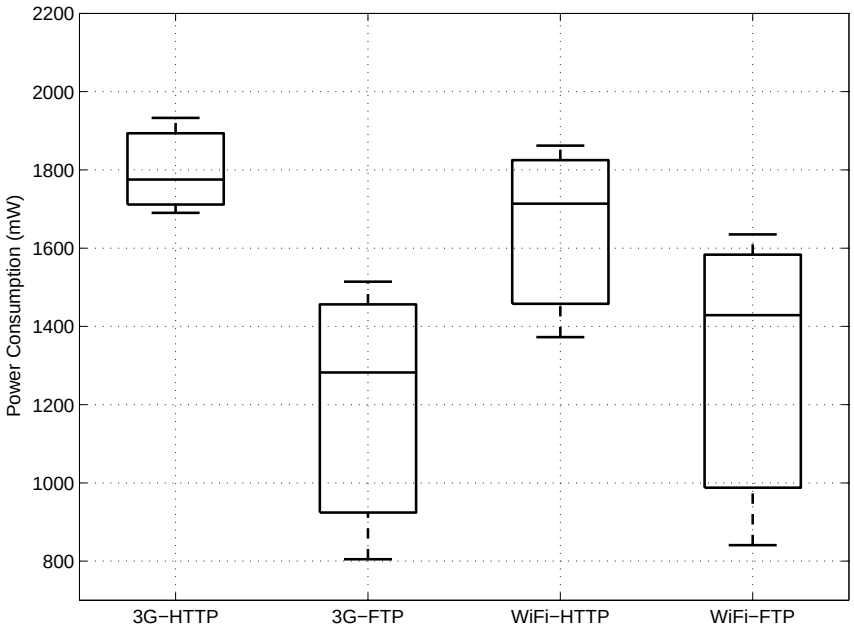


Figure 3.3: Download power consumption

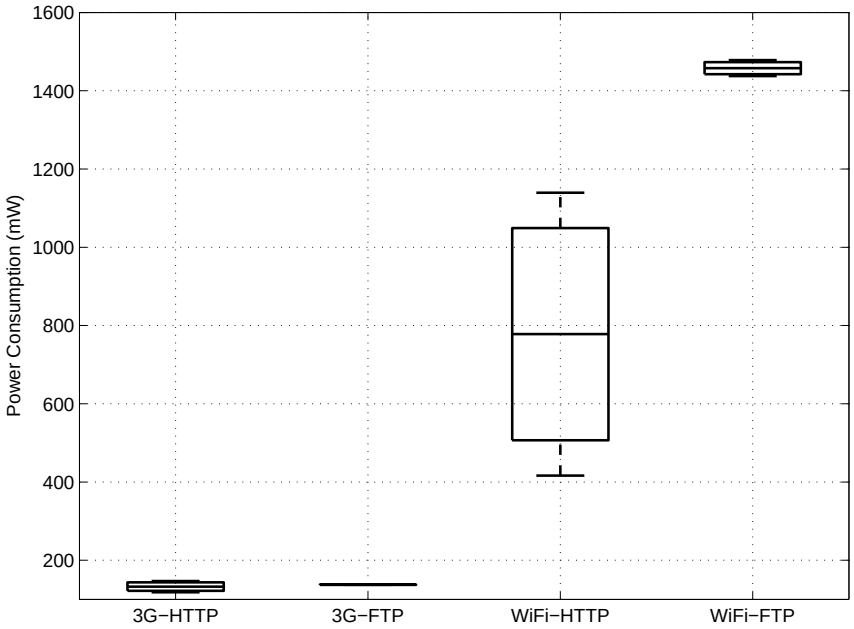


Figure 3.4: Upload data rate

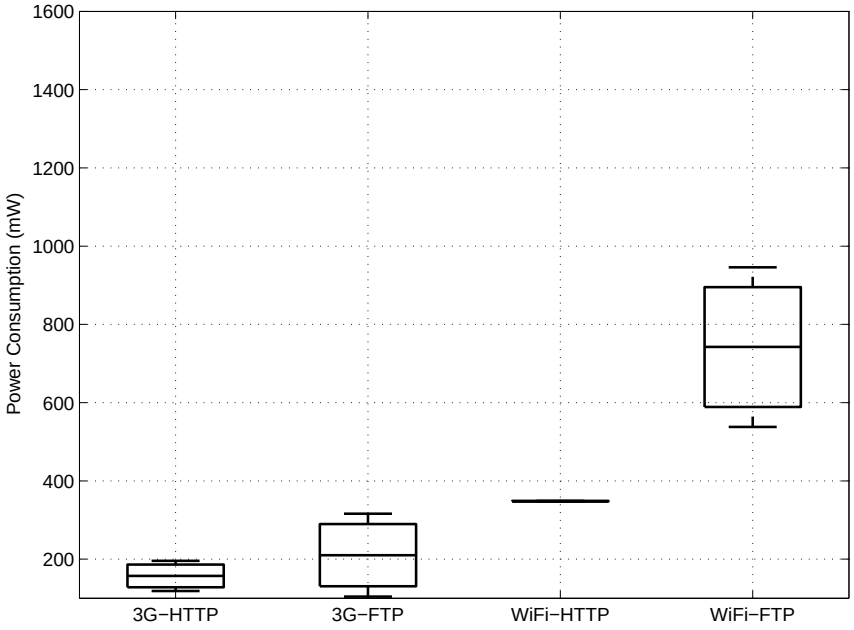


Figure 3.5: Download data rate

Table 3.1: Energy consumption ($\mu J/B$) of smartphone

		3G	WiFi
Download	HTTP	11.3	4.92
	FTP	6.10	1.92
Upload	HTTP	14.56	2.20
	FTP	12.17	1.08

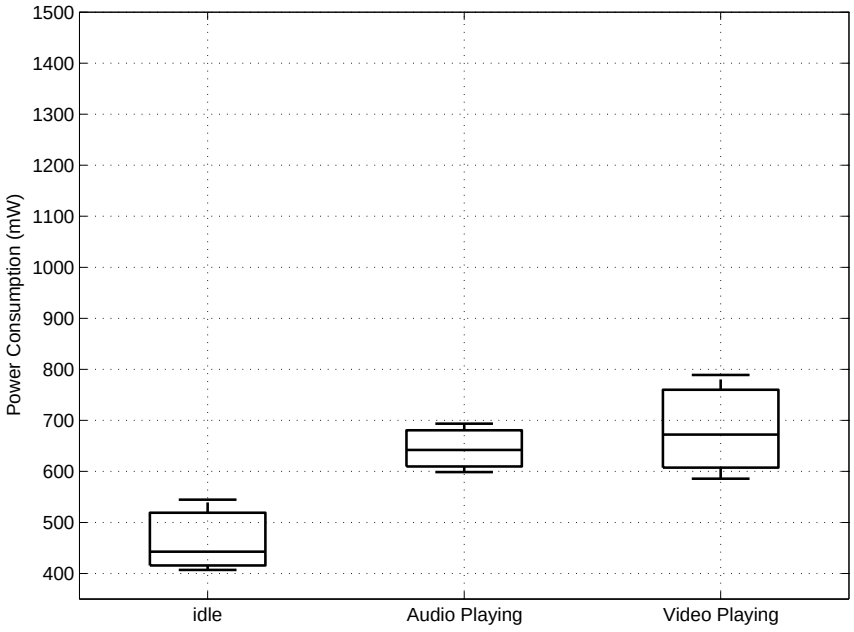


Figure 3.6: Power consumption for playing from the device

energy costs. At this point, the power consumption of playing the media is independent of the network interface. At playing stage, the power consumption of playing the media is shown in Fig. 3.6. We notice that the power consumption of this stage is similar for the playing when the file is on the device local storage. We show the idle case in Fig. 3.6 as a base for comparison purpose when the device is configured as the LCD is ON and the client application is OFF.

In the progressive download scenario, we configure the media client to start playing the video whenever there is enough data to start. After we measure the energy costs, the phases of this scenario can be distinguished easily as three phases. The first phase occurs when the client starts to download as much data as possible, which causes fluctuations in

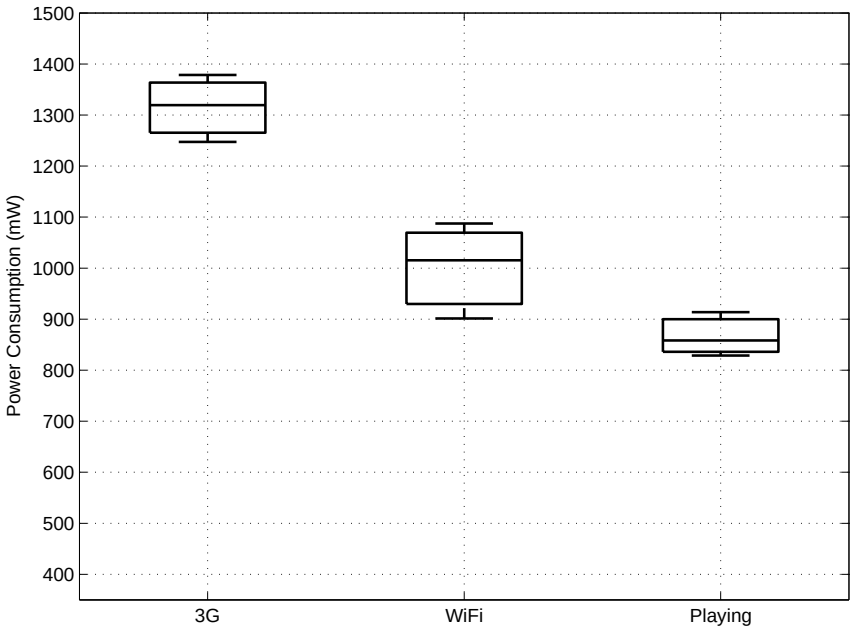


Figure 3.7: Power consumption in progressive download of a video file

the power consumption as shown in Fig. 3.1. The second phase occurs when the client downloads data at the same time of playing the video. Figure 3.7 shows the statistics of the power consumption by the network interfaces at this phase. Finally, the last phase of this scenario is when the device only plays the media after the download is completed. The power consumption of this phase is shown in Fig. 3.7 similar to the case of playing a video file regardless of the interface that has been used. The third phase consumes more power than the playing from the storage that shown in Fig. 3.6. The rise in the power consumption of the third phase may be due to the progressive download keeping the downloaded file in the cache memory of the device where this memory is a new power consumer element.

3.4 Experiments on Cloud Applications

The offloading to the cloud needs much evaluation with respect to the services that are provided to smartphones, in our case the service is in the form of energy saving. In fact, saving smartphone energy is necessary for all of the cloud services [85, 86, 87]. To investigate whether or not smartphones save energy by using the offloading, we conduct

further experiments on a real cloud. As we discussed before, video encoding requires heavy processing that drains smartphone battery if it is performed on the smartphone processor. On the other hand, the data exchange with the cloud over the Internet consumes energy for a smartphone to offload the encoding to the cloud. In these experiments, we investigate and measure the energy costs of using task offloading on *HTC Nexus One* smartphone. These experiments are conducted on the encoding service of the encoding cloud computing (*i.e., www.encoding.com*).

The Multimedia Cloud Computing (MCC), which is a special type of cloud computing, provides the smartphones with encoding functionality to a wide range of multimedia formats to convert from one format to another. The smartphones can provide the multimedia file to the cloud in many ways: *(i)* FTP or HTTP file link, where the file exists on a web server; *(ii)* Cloud file access link, where the file exists on a cloud; and *(iii)* direct upload, where the file exists on the user device. Once the file is uploaded, desired output format is specified and the conversion request will start. Finally, the MCC renders the converted file to the smartphone in many ways: *(i)* upload to FTP to HTTP server; *(ii)* cloud storage; and *(iii)* email link.

Most of the smartphones can play a narrow range of multimedia. For example, the *HTC Nexus One* can recognize and play *mp4, H.263,* and *H.264* video formats. If the user of this device wants to watch *Flash Video (flv)* video, he needs to convert it to another format supported by the same device. In particular, the *flv* format is very common to website hosting but no client, except a *YouTube* client, can play it on smartphones. The same difficulty applies to other video formats as well.

In the experimental setup, we consider all encoding scenarios for a multimedia file as depicted in Figs. 3.8 and 3.9. Figures 3.8 and 3.9 depict two broad experimental scenarios related to the location of the original file to be encoded. In Fig. 3.8, the original file is available on the smartphone itself. On the other hand, the original file is available in the cloud as in Fig. 3.9. For uploading and downloading files to and from the cloud, we consider the energy implications of: *(i)* using the HTTP and FTP protocols at the application level; and *(ii)* using the 3G and WLAN communications at the wireless access interface level. Using Fig. 3.8, we compare the energy cost of locally performing file encoding on a smartphone with the total energy cost of performing the same operation in the cloud, including the uploading and downloading communication costs. Similarly, using Fig. 3.9, we compare the energy cost of downloading an encoded file with the total energy cost of downloading the original file and performing encoding on a smartphone. Therefore, we perform experiments with eight scenarios for each of Figs.3.8 and 3.9. There are four scenarios related to the location of the original file to be encoded as follows.

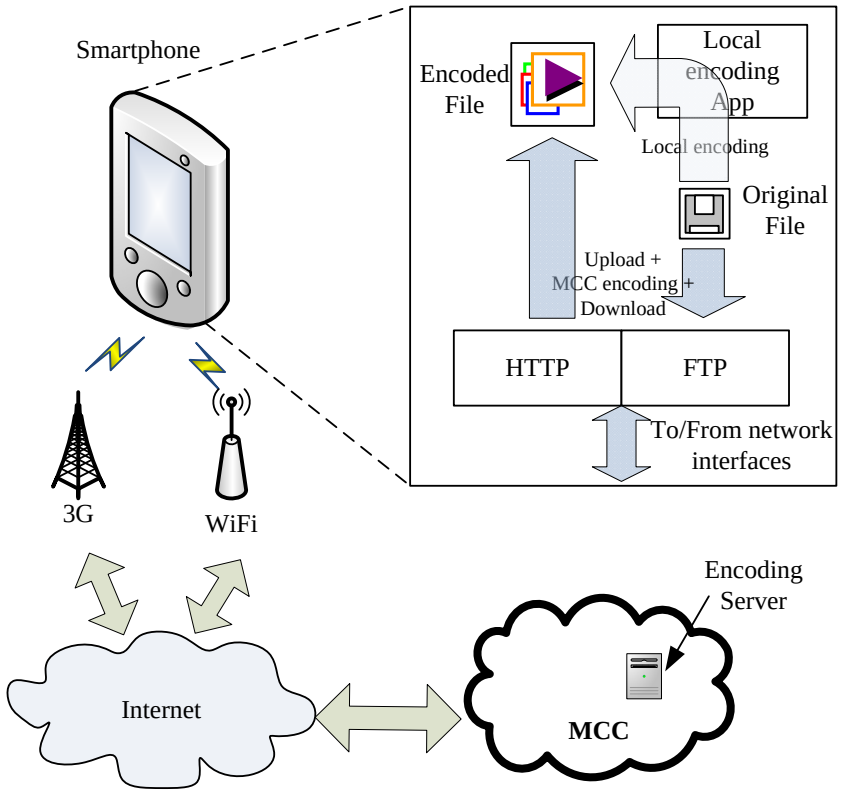


Figure 3.8: Encoding scenarios where the original file exists on the smartphone

1. The first scenario is the local encoding using the smartphone's encoding application (*i.e.,* Local encoding in Fig. 3.8).

2. The second scenario is the offloading technique by uploading the original multimedia file and doing the encoding by the MCC then downloading the encoded multimedia file (*i.e.,* Upload + MCC encoding + Download in Fig. 3.8).

3. The third scenario is the local encoding using the smartphone's encoding application but after download the original file from the MCC (*i.e.,* Download + Local encoding arrow Fig. 3.9).

4. The fourth scenario is the encoding using MCC after load the original file from cloud storage then downloading the encoded multimedia file (*i.e.,* MCC encoding + Download arrow Fig. 3.9).

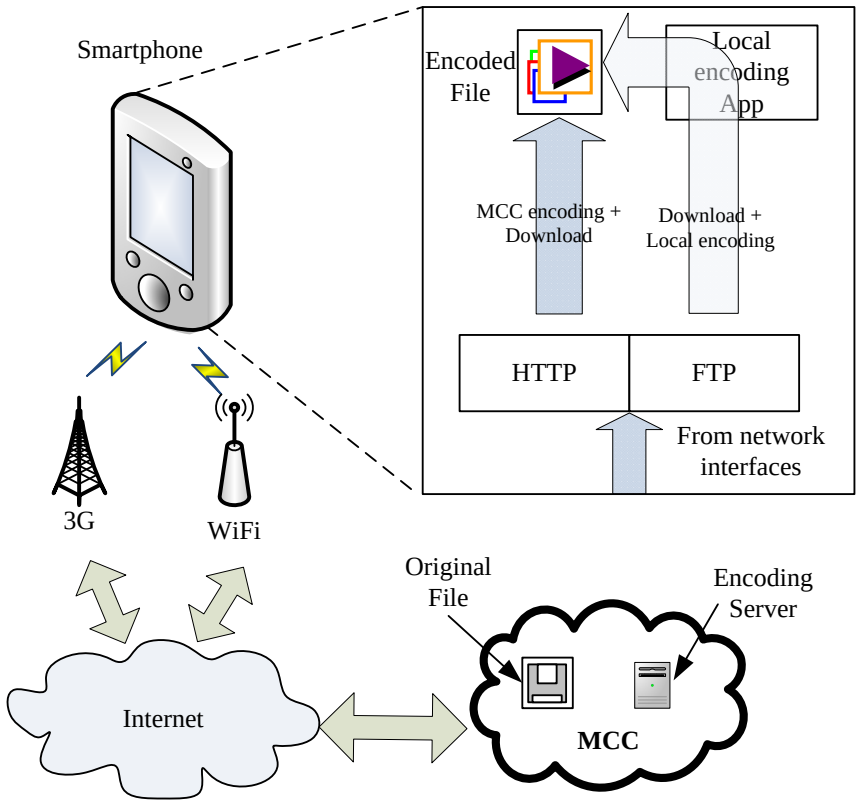


Figure 3.9: Encoding scenarios where the original file exists on the MCC

Table 3.2: Properties of the video files

Parameters	Original flv	Converted mp4 at settings	
		Default	Customized
Length (minutes:seconds)	3:31	3:31	3:31
File size (MB)	23.97	8.21	131.5
Frame width (pixel)	640	320	640
Frame height (pixel)	360	240	360
Data rate (Kbps)	909	258	5057
Frame rate (frame/s)	24	24	24
Audio bit rate (Kbps)	114	63	172
Channel	2	2	2
Audio sample rate (KHz)	44.1	44	44

In detail, we choose a song video in *flv* format and convert it into *mp4* video format. Table 3.2 lists the original and encoded video parameters. Each individual file format has unique performance parameters such as the data bit rate. In both of MCC and smartphone encoding application, the encoding parameters are kept to default settings. Unfortunately, the settings are different as shown in Table 3.2. The default and customized parameters are the parameters of encoded by MCC in default and customized setting, respectively. For fair comparison, we customized the encoding parameter for the MCC that match the parameters of the smartphone encoding application. Thanks to MCC that gives the opportunity to configure the conversion parameters since the smartphone application does not.

We present here the total energy costs for different network interfaces (*i.e.,* 3G and WLAN) and Internet protocols (*i.e.,* HTTP and FTP). In these results, if a network interface and an Internet protocol are used for uploading a file to the MCC, the same interface and protocol are used for download the converted file. Therefore, there is no permutation between network interfaces and Internet protocols in the uploading and downloading are presented here. This is because we belief that the users tend to use the same configurations at a time. Figure 3.10 reveals the capability of MCC to provide the cloud service that we called *Energy as a Service (EaaS)* for mobile devices. The MCC provides at least 60% reduction in energy when the 3G interface with HTTP protocol is used. In addition, this figure shows the effect of the network interfaces and the Internet protocols on the energy consumption. In general, the WLAN interface consumes less energy that the 3G interface for transmitting the same amount of data. Similarly, the FTP protocol consumes less energy than the HTTP protocol for transmitting the same amount of data.

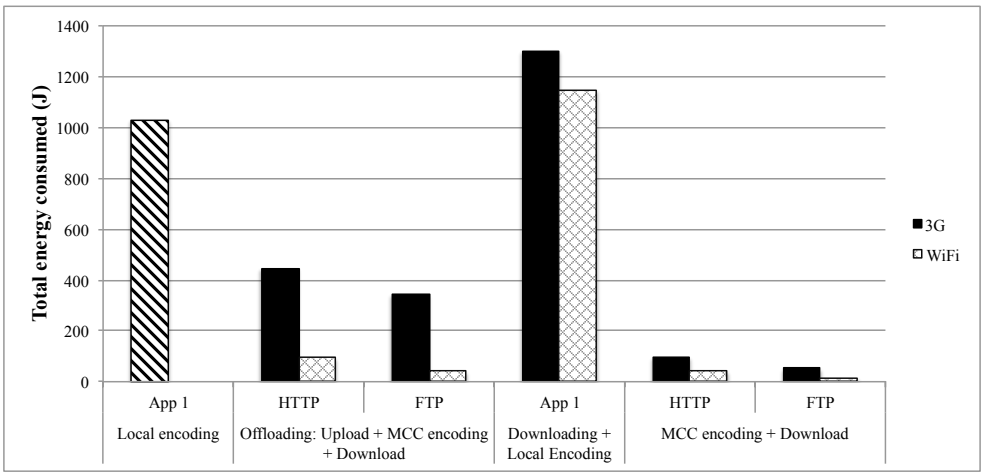


Figure 3.10: Total energy consumed by *App1* and the default MCC settings

Figure 3.11 shows the comparison when the MCC settings are identical as the settings of the smartphone application. This figure shows the impact of the file size in the offloading process since the customized settings required larger file size as show in Table 3.2. In this case, the offloading does not save energy when the 3G interface is the only available interface. From this figure, we can argue why we examine each of the network interfaces and the Internet protocols. It is worth mentioning that the customized configurations enlarge the encoded file without any noticeable improvement in the Quality of Experience (QoE) to the end user.

On the other hand, for the local encoding application, we test two versions of the converting application, which are PC-like applications, namely *App1*, on smartphone (*i.e., MP3 Media Converter, version 1.1.0.3, by Ansha Team*) and optimized version, namely *App2*, for the smartphone (*i.e., MP3 Media Converter (Neon), version 1.1.0.2, by Ansha Team*). The optimized version exploits the multimedia acceleration capability of the smartphone that is called Single Instruction Multiple Data (SIMD). All of these applications are available on the application market of *Android*. Similarly, Fig. 3.12 and 3.13 compare the total energy consumed in customized application (*i.e., App2*) and offloading to MCC with default and customized settings, respectively.

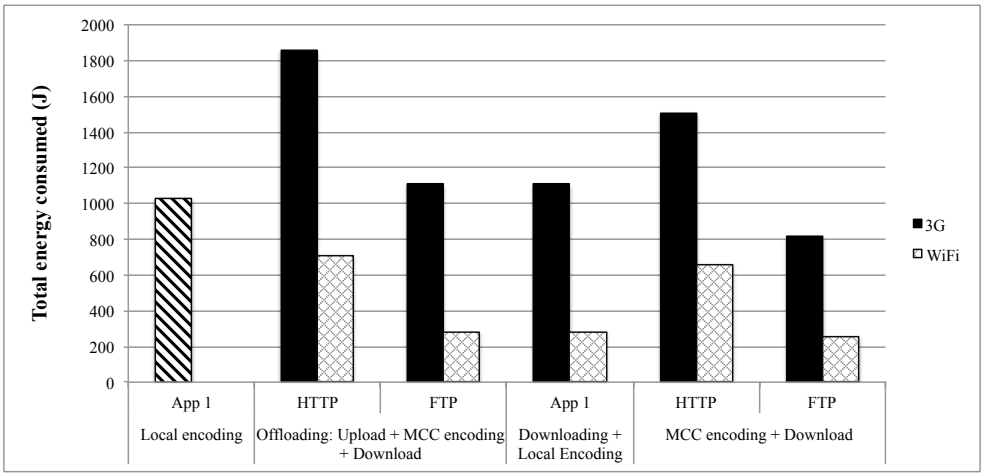


Figure 3.11: Total energy consumed by *App1* and the customized MCC settings

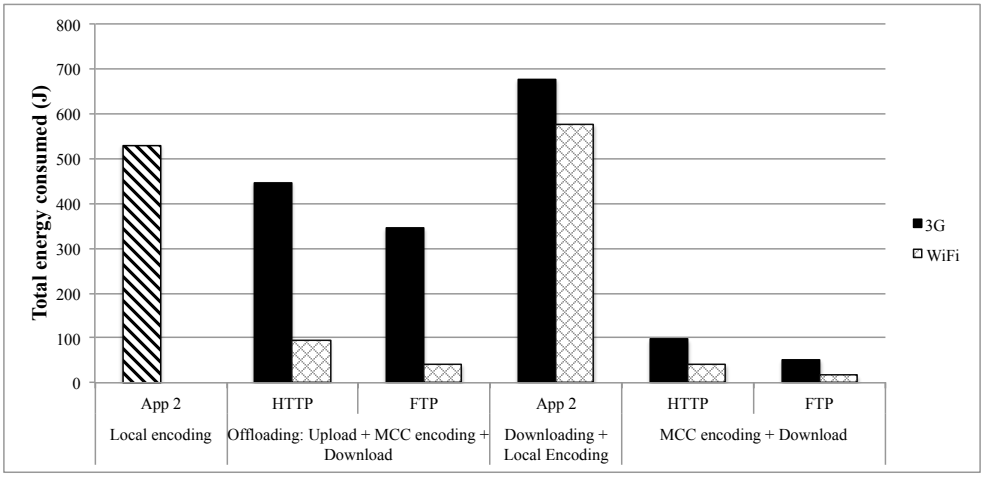


Figure 3.12: Total energy consumed by *App2* and the default MCC settings

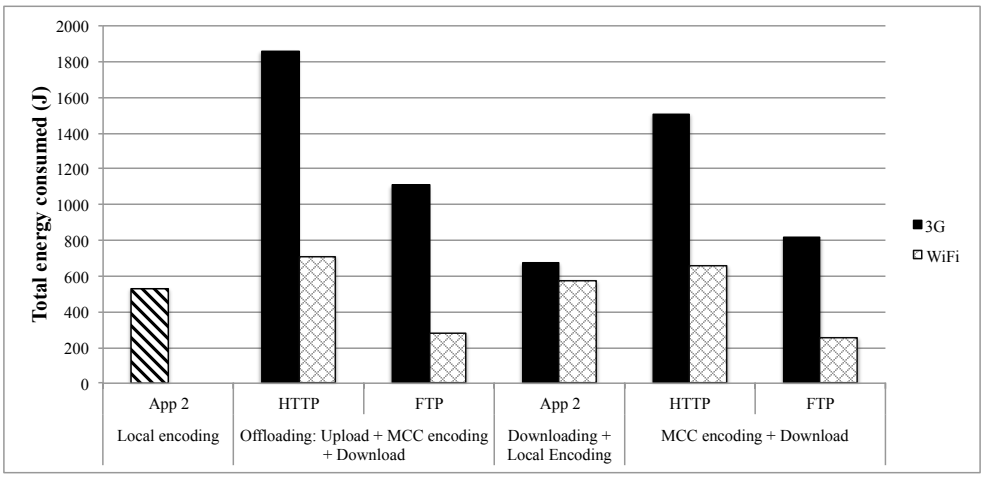


Figure 3.13: Total energy consumed by *App2* and the customized MCC settings

3.5 Limitations of Our Approach

Our experiments show that the NRA performance highly depends on the network interfaces and protocols. In general, it is shown that 3G interface consumes more power than the WLAN interface. This means offloading a task via 3G interface has to be avoided in the availability of WLAN interface. Moreover, the FTP protocol consumes less power than the HTTP protocol. Recommending a single interface is difficult because each one provides the end-users a unique experience. For instance, 3G interface supports a large range communication while WLAN supports short range. However, choosing interface decision depends on the NRA requirements such as having a specific data rate.

The experiment results show that some energy readings fluctuate widely caused by the data rate of the cloud servers or caused by the change in the wireless network conditions. This fluctuation also depends on the contents of the media as it has been proven in [88]. For example, the server provides a high data rate at the beginning of video progressive download to guarantee a certain level of quality of experience. In contrast, the server uses other procedure for the audio files. We limited our experiments to exclude the dependency of the energy cost on the cloud and the network conditions.

In our comparison experiments, we have been limited to the local applications that have fixed conversion parameters and produce a large file. For fair comparison, we customize the conversion parameters on the MCC to match the conversion parameters of the local applications. Therefore, the energy cost of downloading the converted file is quite high.

We should mention that the MCC converts a video file into a typical size, format, and quality for *Android* smartphones if the target device is specified. Our experiments reveal that the MCC auto conversion parameters would reduce the downloading energy costs for the smartphone from 90% to 95% where the custom configuration could save from 30% to 70%.

In our experiments, we use the Internet protocol *IPv4* that does not distinguish the information contents. If the smartphones support *IPv6*, the experience is likely to be different because this protocol offers *QoS* and multimedia priority procedure and could offer efficient energy solution and offloading techniques [89].

3.6 Summary and Discussion

Our study clearly indicates that offloading heavy applications, namely multimedia applications, from smartphones to MCC is beneficial. The MCC significantly reduces the energy consumption on smartphones. Moreover, the MCC enriches smartphones capabilities for multimedia applications.

At this time when the CC is in its infant state, the importance of evaluating the benefit of MCC to overcome smartphone constraints motivates us to conduct this study. A large number of experiments have been performed for common network interfaces (*i.e.,* 3G and WLAN) and protocols (*i.e.,* HTTP and FTP). The location of the original file has been considered. This chapter provides a wide range of comparison between possible encoding location, original file place, encoding configuration parameters, network interfaces, and Internet protocols. The results reveal the potential of the cloud by reducing smartphones energy consumptions on multimedia applications at least 30%.

At this point of our research, we argue that task offloading is beneficial and the need for offloading framework is fundamental. The framework performs the offloading decision accurately to whether or not the offloading is beneficial based on the system parameters. In this case, we need to develop estimation models to the energy consumption for each system components, which are the networking components and the computing components. The networking components are used for transferring the data between the device and the cloud in the case of the decision to offload the task. On the other hand, the computing components are used to execute the task on the device locally. Estimating the energy for these sets of mobile device components is the key for the offloading framework to make the decision. In the next chapters, we model the energy consumption of the networking components (*i.e.,* Chapter 4) and of the computing components (*i.e.,* Chapter 5).

Chapter 4

Modelling the Networking Energy Consumption

Task offloading from mobile devices to the cloud is a promising strategy to enhance the computing capability of mobile devices and prolong their battery life. However, task offloading introduces a communication cost for those devices. Therefore, consideration of the communication cost is crucial for the effectiveness of task offloading. To make task offloading beneficial, one of the challenges is to estimate the energy consumed in communication activities of task offloading. Accurate energy estimation models will enable these devices to make the right offloading decisions as to whether or not to perform task offloading, based on the energy cost of the communication activities. Simply put, if the offloading process consumes less energy than processing the task on the device itself, then the task is offloaded to the cloud. To design an energy-aware offloading strategy, we develop energy models for the WLAN, Third Generation (3G), and Fourth Generation (4G) interfaces of mobile devices. These models make mobile devices capable of accurately estimating the energy cost of task offloading. We validate the models by conducting an extensive sets of experiments on five smartphones from different vendors. The experimental results show that our estimation models accurately estimate the energy required to complete a task offloading. The contributions of this chapter are published in [9].

4.1 Preamble

In order to make the offloading beneficial, the energy cost of offloading for a given task should be estimated to compare it with the energy cost of executing the task locally. From

a smartphone point of view, the energy consumed during task offloading is mainly caused by the networking activities. The focus of this study is to develop energy models for task offloading caused by the networking activities. Specifically, we model the energy cost at the application level considering all the details of the network stack (*i.e.,* Transmission Control Protocol (TCP), Media Access Control (MAC), and Physical layer (PHY)).

In this chapter, we develop and validate mathematical models for the energy that smartphones consume during network activities for task offloading. We consider in our models the most common network interfaces: WLAN and 3G/4G. We conduct experiments on popular smartphones (*i.e., HTC Nexus One, LG Nexus 4, Samsung Galaxy S3, BlackBerry Z10, and Samsung Galaxy Note 3*) to validate our energy models. The experimental results reveal that our energy estimation models are able to estimate the energy accurately.

This chapter makes the following contributions:

1. introduce mathematical models to estimate the energy consumed in a smartphone to perform task offloading at the cases of:
   (a) file downloading using WLAN and 3G/4G network interfaces; and
   (b) file uploading using WLAN and 3G/4G network interfaces.
2. developed models so that provide an accurate estimation to the total energy consumed for task offloading by only taking the amount of data that the smartphone would transfer for task offloading as an input.
3. validate the energy models by means of implementation and measurement. In these experiments, we measure the actual energy consumed in the smartphones for each of the aforementioned network activities.

This chapter is organized as follows. We discuses the literature of modeling the energy cost of smartphone networking in Section 4.2. In Section 4.3 and Section 4.4, we provide the details of our energy estimation models for WLAN and 3G/4G networking, respectively. The validation of the models and the experimental results are discussed in Section 4.5. The discussion and summary are presented in Section 4.6.

4.2 Literature of the Networking Energy Modelling

4.2.1 Modelling Studies

Developing mathematical models for the energy consumption is essential to make task offloading beneficial with respect of energy cost. That is, the offloading decision depends on the estimation of the energy cost, which is modelled mathematically, for offloading the task to the cloud and for executing it locally. Modelling the energy consumption has been developed extensively in the literature for the use of energy saving techniques such as task offloading technique.

Zhang et al. [90] and Jung et al. [91] profiled the energy consumption of mobile device hardware components including the wireless interfaces. The profiling is developed by analyzing the access event of the system to the component and the change in the power state, which is provided from the Battery Monitoring Unit (*BMU*). Their mathematical models are built based on the analysis to the experimental results. As a result, the models lack for system analysis and the detail of the protocols. In addition, the *BMU* can not trace events that are shorter than *BMU* update rate as in the case of wireless interfaces. Therefore, the models are not accurate and not extendible for modelling the energy consumption of the wireless interfaces.

In contrast, Xiao et al. [92] presented an energy cost model for *IEEE 802.11g* networks. The model takes into account the impact of the transmission control protocol (TCP) and Internet traffic flow characteristics on the power consumption of smartphones running different operating systems. The model abstracts the detailed operation of the *IEEE 802.11g* protocol, such as the RTS-CTS packets exchange, the average back-off time, and the transmission of ACK packets. Our MAC (media access control) energy model accurately takes the detailed operation of *IEEE 802.11* into consideration where it is developed based on the *IEEE 802.11g* protocol parameters. Hence, it can be easily extended to other *IEEE 802.11* standards.

The wireless interface of a mobile device with 3G/4G radio consumes deterministic levels of power. These levels are associated with the radio resources that the interface was granted form the network. For instance, the interface consumes a specific amount of power during the data transfer period and another amount during signaling. Qian et al. [93] and Huang et al. [94] showed these distinct levels of power consumption by tracing the radio resources and power consumptions of the smartphones for 3G and 4G networks, respectively. We use this concept to develop our models. Rather than consider the power consumption of individual components inside the interface [95], we consider the overall

power consumption of the network interface, because we develop our models to be used at the upper system level where one only sees the total power consumption of the interface. This will simplify our models and reduce the parameters that are used for the offloading decision.

4.2.2 Networking Measurement Studies

In the field of energy measurements for the networking on the mobile devices, Xiao et al. [96] presented a case study of energy cost for mobile *YouTube* (*m.youtube.com*) on a mobile device (*Nokia S60*) using 3G and WLAN networks. Energy cost data is collected by the Nokia Energy Profile application that itself runs on the mobile device to measure the current and the voltage of the device battery. The analysis reveals that 3G consumes 1.45 times more energy than WLAN. Moreover, download-and-play consumes more energy than progressive download because the network modules continue to remain active for a while after the download is finished.

Abogharaf et al. [97] proposed an energy-efficient and client-centric algorithm based on experimental observations of data streaming. Their study shows the impact of communication parameters (*i.e.,* buffer size, low water mark, and socket-reading size) on the energy consumed during data streaming. The parameters affect the sleep behaviour of the wireless network interface controller (WNIC). The proposed algorithm tunes those parameters in an energy efficient way by utilizing the WNIC during the continuous active mode (CAM) and maximizing the use of power saving mode.

Albasir et al. [98] measured the energy cost of web browsing for different contents, and they observed that for web pages containing advertisements (ads) a smartphone consumes more energy than the same web pages without ads. Based on this observation, a client-server algorithm is proposed that saves energy by managing the web browsing contents. The server adapts the contents of the web pages based on smartphone requests, where the requests include battery-level and type of network connection.

The distinction between our work and the above work is that we consider the offloading decision problem in the application layer by taking into account the impact of lower layers on the energy consumption. We analyze the lower layer protocols to build reasonable and realistic models. In addition, we keep our models extendible to the next generation of wireless communication systems by developing our models based on the analysis to the networking layer standards. Furthermore, we develop fine-grained mathematical models first, and then we validate them experimentally. We do not drive the models from the experimental results by extracting the statistical properties and analyzing the observations.

In the experiments, we measure the actual energy using external measurement equipment to avoid measurements overhead such as *BMU* overhead, and to obtain very high precision readings.

4.3 WLAN Analytical Energy Model

We consider a single-channel *IEEE 802.11g* WLAN network. Following the carrier-sense multiple access with collision avoidance (CSMA/CA) protocol as described in the *IEEE 802.11* standard [99], if a node has a data packet to transmit and senses the channel to be idle for a period of Distributed InterFrame Spacing (DIFS), the node proceeds by transmitting an RTS packet. If the channel is busy, the node defers its transmission until an idle DIFS is detected and waits for a random backoff time in order to avoid collisions. The backoff time counter is chosen uniformly in the range [0,W_i - 1], where $i \in [0,\ m_b]$, m_b is the number of backoff stages, and W_i is the current contention window (CW) size in time slots. A time slot is the unit time in *IEEE 802.11*. The contention window at the first transmission of a packet is set equal to CW_{min}. After an unsuccessful transmission, the CW is doubled up to a maximum value

$$CW_{\max} = 2^{m_b} \times CW_{\min} \tag{4.1}$$

The backoff counter decreases at every slot time when the channel is sensed idle. The counter is stopped when the channel is busy and resumed when the channel is sensed idle again for more than DIFS. A station transmits the RTS packet when its backoff timer reaches zero. If the destination station successfully receives the RTS packet, it responds with a CTS packet after a short inter-frame space (SIFS) time interval. Upon the reception of the CTS packet, the sender sends the data packet. The receiver then waits for an SIFS time interval and transmits an acknowledgement (ACK) packet. If the ACK packet is not received within a specified ACK timeout interval, the data packet is assumed to be lost and a retransmission will be scheduled.

We assume a fixed packet size. The packet transmission time T_s is given by [100]:

$$T_s = T_{RTS} + T_{CTS} + 3SIFS + T_{ACK} + T_D + DIFS \tag{4.2}$$

and the packet collision time, which is the channel time wasted in a packet collision, is given by:

$$T_c = T_{RTS} + DIFS \tag{4.3}$$

Table 4.1: *IEEE 802.11g* system parameters

System Parameter	Value
Packet payload	L_{max} bits
MAC Header H_{MAC}	208 bits
T_{PHY}	26μs
T_{RTS}	$7.583\mu + T_{PHY}$
T_{CTS}	$5.583\mu + T_{PHY}$
T_{ACK}	$5.583\mu + T_{PHY}$
Slot Time (σ)	9 μs
Short Inter-Frame Space ($SIFS$)	10 μs
Distributed Inter-Frame Spacing ($DIFS$)	28 μs
Basic Rate	24 Mbps
Data Rate	$6 \leq R_{data} \leq 54$ Mbps
CW_{min}	32
Backoff stages (m_b)	5

The symbols T_{RTS}, T_{CTS} and T_{ACK} represent the transmission times for the RTS, CTS, and ACK packets as given in Table 4.1 [99], respectively; T_D is the data packet transmission time, which is constant for a fixed packet size.

We model the case of a single user in the WLAN network. Therefore, the probability that a node sends a packet at a random time slot can be give as [100]:

$$\tau = \frac{2}{CW_{\min} + 1} \tag{4.4}$$

We assume that the file size is B bytes and each TCP segment is carried only in one MAC frame. Therefore, the total number of MAC frames submitted to the AP by the node under study is $\frac{B}{F_s}$, where F_s is the MAC frame size in bytes.

In the following, we model the energy usage in two distinct cases, namely, file upload and file download. For simplicity, we assume that the mobile device transceiver uses only two power levels, namely, P_{RX} when it is idle, in backoff mode, or receiving and P_{TX} when it is transmitting.

4.3.1 File Download Case

In this case, the mobile device is mostly receiving. Here, we address first the general situation where there is no limitation on the file download rate from the cloud. Next, we address the situation where the cloud restricts the file download rate. For every MAC frame to be received, the mobile device has to send a CTS and an ACK frame. The mobile device has to send a TCP ACK for every received TCP segment. During downloading a file, a smartphone will be receiving a data frame for a time $T_D + 3SIFS + T_{PHY} + T_{RTS}$ and it has to wait for the AP backoff time $\frac{\sigma}{\tau}$ [100]. The smartphone also receives an acknowledgement for the TCP ACK it sends to the AP after receiving a data frame of the file being downloaded. On the other hand, the smartphone sends a TCP ACK using the basic access method (*i.e.,* only DATA-ACK) so it has to wait for an average backoff time of $\frac{\sigma}{\tau}$ [100]. It also has to send a MAC ACK and a CTS frame for each data frame it receives from the AP. Therefore, the total energy consumed in a file download can be obtained as

$$\begin{aligned} E_d &= \left\lceil \frac{F}{F_s} \right\rceil \times \left\{ \begin{array}{l} (T_{RTS} + T_{ACK} + 3SIFS + T_{PHY} + T_D + T_H) P_{RX} + \\ (T_{ACK} + T_{CTS}) P_{TX} \end{array} \right\} \\ &\quad + N_{d_{ACK}} (T_H + T_{PHY} + T_{TACK}) P_{TX} + \frac{\sigma}{\tau} P_{RX} \end{aligned} \tag{4.5}$$

where the TCP acknowledgement transmission time $T_{TACK} = \frac{ACK_{TCP}}{R_{data}}$ and $N_{d_{ACK}}$ is the number of TCP acknowledgements received by the smartphone in the download case, which is given as

$$N_{d_{ACK}} = \left\lceil \frac{F}{N_{d_{seg}} F_s} \right\rceil \tag{4.6}$$

where $T_D = \frac{L_{max}}{R_{data}}$, the transmission time for the MAC header $T_H = \frac{H_{MAC}}{R_{data}}$, the TCP acknowledgement transmission time $T_{TACK} = \frac{ACK_{TCP}}{R_{data}}$, and $N_{d_{seg}}$ is the number of TCP segments that can be sent without receiving an acknowledgement in the download case.

In fact, Eq. (4.5) estimates the consumed energy in downloading a file when the server hosting the file has no limitation on the download data rate. If there is a limitation on the download rate, there will be some idle time the mobile terminal will experience between downloading a TCP segment and the subsequent segment. This case can be taken into account by adding the term D_{ns} to Eq. (4.5) as follows:

$$
\begin{aligned}
E_{d_{ns}} \;=\; & \left\lceil \frac{F}{F_s} \right\rceil \times \left\{ \begin{array}{l} \left(\frac{\sigma}{\tau} + T_{RTS} + T_{ACK} + 3SIFS + T_{PHY} + T_D + T_H\right) P_{RX} + \\ \left(T_{ACK} + T_{CTS}\right) P_{TX} \end{array} \right\} \\
& + N_{d_{ACK}} \left(\frac{\sigma}{\tau} + T_H + T_{PHY} + T_{TACK}\right) P_{TX} \\
& + D_{ns}
\end{aligned} \tag{4.7}
$$

where R_s is the server download rate and

$$
D_{ns} = \left\{ \begin{array}{l} \left\lceil \frac{F}{F_s} \right\rceil \times \left(\begin{array}{l} \left[\begin{array}{l} \frac{L_{max}}{R_s} - \frac{\sigma}{\tau} - T_{RTS} - T_{ACK} - \\ 3SIFS - T_{PHY} - T_D - T_H \end{array} \right] - \\ \left(T_{ACK} + T_{CTS}\right) \end{array} \right) - \\ \left(\frac{1}{N_{d_{seg}}} \left(\frac{\sigma}{\tau} + T_{PHY} + T_H + T_{ACK}\right)\right) \end{array} \right\} P_{RX} \tag{4.8}
$$

The term D_{ns} takes into account the amount of energy consumed by the smartphone while in idle state within the inter-arrival time ($\frac{L_{max}}{R_s}$) of two consecutive TCP segments.

4.3.2 File Upload Case

Similar to the download case, we address first the general situation where there is no limitation on the upload rate to the cloud. Next, we address the situation where the cloud restricts the file upload rate. In the upload case, we take into account that the smartphone receives a TCP ACK for every frame of the uploaded file so it has to wait for the backoff time $\frac{\sigma}{\tau}$ that the AP takes to transmit the TCP ACK in addition to the TCP ACK transmission time. The smartphone also has to send a MAC ACK for each TCP ACK. For every frame the smartphone transmits, it receives a CTS, an ACK, and waits for $3T_{SIFS}$ and an average backoff time of $\frac{\sigma}{\tau}$. In addition to sending the data frame, the smartphone sends an RTS for every data frame of the uploaded file. Therefore, the total energy used for uploading a file can be given as

$$
\begin{aligned}
E_u \;=\; & \left\lceil \frac{F}{F_s} \right\rceil \times \left\{ \begin{array}{l} \left(\frac{\sigma}{\tau} + T_{CTS} + T_{ACK} + 3SIFS\right) P_{RX} + \\ \left(T_{RTS} + T_H + T_{PHY} + T_D + T_{ACK}\right) P_{TX} \end{array} \right\} \\
& + N_{u_{ACK}} \left(\frac{\sigma}{\tau} + T_{TACK} + T_H + T_{PHY}\right) P_{RX}
\end{aligned} \tag{4.9}
$$

where $N_{u_{ACK}}$ is the number of TCP acknowledgements received by the smartphone in the upload case, which is given as

$$N_{u_{ACK}} = \left\lceil \frac{F}{N_{u_{seg}} F_s} \right\rceil \tag{4.10}$$

where $N_{u_{seg}}$ is the number of TCP segments that can be sent without an acknowledgement in the upload case.

Similar to the download case, if there is a limitation on the file upload rate, there will be some idle time the mobile terminal will experience between uploading a TCP segment of the file and the subsequent segment. This case can be taken into account by adding the term U_{ns} to Eq. (4.9) as in the following.

$$\begin{aligned} E_{u_{ns}} \;=\; & \left\lceil \frac{F}{F_s} \right\rceil \times \left\{ \begin{array}{l} \left(\frac{\sigma}{\tau} + T_{CTS} + T_{ACK} + 3SIFS\right) P_{RX} + \\ \left(T_{RTS} + T_H + T_{PHY} + T_D + T_{ACK}\right) P_{TX} \end{array} \right\} \\ & + Nu_{ACK} \left(\frac{\sigma}{\tau} + T_{TACK} + T_H + T_{PHY}\right) P_{RX} \\ & + U_{ns} \end{aligned} \tag{4.11}$$

where

$$U_{ns} = \left\{ \begin{array}{l} \left\lceil \frac{F}{F_s} \right\rceil \times \left(\begin{array}{l} \left[\frac{L_{max}}{R_s} - \frac{\sigma}{\tau} - T_{CTS} - T_{ACK} - 3SIFS \right] - \\ \left(T_{RTS} + T_H + T_{PHY} + T_D + T_{ACK}\right) \end{array} \right) - \\ \left(\frac{1}{N_{d_{seg}}} \left(\frac{\sigma}{\tau} + T_{PHY} + T_H + T_{TACK}\right) \right) \end{array} \right\} P_{RX} \tag{4.12}$$

The term U_{ns} takes into account the amount of energy consumed by the smartphone while in idle state during the time ($\frac{L_{max}}{R_s}$) between uploading two consecutive TCP segments of the file under consideration.

4.4 Mobile Data Analytical Energy Model

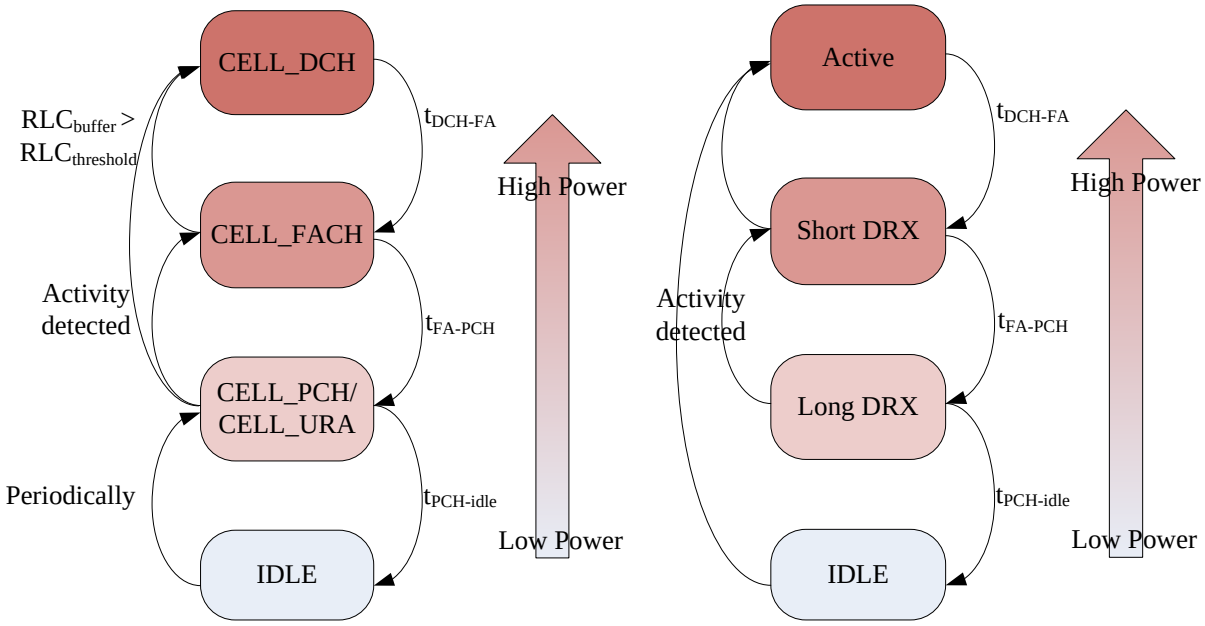
In this section, we present our models of energy consumption of a smartphone connected to mobile data networks. Specifically, we develop our models for 3G and 4G mobile networks.

4.4.1 Background

The 3G and 4G networks contain a Radio Resource Controller (RRC), which manages all communication between the user devices (*i.e.,* user equipment, UE) and provider networks. The aim of RRC is to provide high performance mobile connectivity by reducing signalling latency and UE power consumption, and enhance the network throughput. From here, the RRC has direct impact on the power consumption of the smartphone. Always-on keeps the latency low, but drains the UE battery quickly. In contrast to always-connected, only connected for data exchange minimizes the power consumption, but increases link setup signalling and consequently increases the network latency. The trade-off between these two cases is served by the RRC in which the radio resources take a specific status based on some conditions to change between these statuses. The mechanism of RRC and specifications are implemented and described in the 3rd Generation Partnership Project (3GPP) standards. In general, the RRC defines two major states for the radio connection, which are `RRC_IDLE` and `RRC_CONNECTED`. In the `RRC_IDLE` state, the radio is in a low-power state and there is radio resources are assigned to the UE. In this case, the UE tunes to the shared control channel where it listens to control traffic. In the `RRC_CONNECTED` state, the radio is in a high-power state and a data connection is established where dedicated radio resources are allocated to the UE. The transition to `RRC_CONNECTED` only occurs when the UE hears from the network broadcast that there is data to be received or the local buffer of transmission exceeded its threshold. At that time, the UE initiates a connection by sending connection request to the network through promotion signalling procedure [101]

In the 3G networks, the `RRC_CONNECTED` state is divided into two sub-states for further improvement, as depicted in Fig. 4.1(a). The `CELL_DCH` state is the state where a device is in a high-power state and network resources are assigned for data transfer. The `CELL_FACH` is an intermediate power state, where no dedicated network resources are assigned but a shared low-speed channel. At `CELL_FACH`, a device consumes significantly less power than at `CELL_DCH`. The buffer thresholds and the RRC timer govern the transitions among these states. If the buffer state is not changed, the UE does not change the power state to lower power state until the timer has expired. The timer keep the interface active waiting for possible next network activity to reduce the signalling, but if nothing happens it switch to lower power state. In fact, the UE wastes some UE energy called tail energy because of this timer.

Similarly, the `RRC_CONNECTED` is divided into three sub-states in the 4G networks as shown in Fig. 4.1(b). The Active state is similar to the `CELL_DCH` in the 3G. Similar to the `CELL_FACH` in the 3G, for better RRC performance the 4G networks use further sub-states called Short Discontinuous Reception (Short DRX) and Long Discontinuous



(a) 3G RRC status (b) 4G RRC status

Figure 4.1: 3G and 4G RRC status

Reception (Long DRX) in the downlink, and Discontinuous Transmission (DTX) and Long Discontinuous Transmission (Long DTX) in the uplink.

4.4.2 Energy Models

Based on the operation of the RRC states described above, the total energy consumed to transfer data consists of three parts: promotion signaling, data transfer, and tail energy [93, 94]. Figure 4.3 shows an example of these three parts in case of downloading data over a 3G network.Therefore, the general energy consumption model follows the following equation:

$$E_{3G/4G} = E_{ps} + E_{trx} + E_{tail} \tag{4.13}$$

where E_{ps}, E_{trx}, and E_{tail} are the energy consumed on promotion signaling, data transfer, and tail timer, respectively.

The energy consumed for a given task equals the power multiplied by the duration that the smartphone takes to finish the task, which is expressed as $E = P \times T$. Then, Eq.

(4.13) becomes

$$E_{3G/4G} = P_{ps} \times T_{ps} + P_{trx} \times T_{trx} + P_{tail} \times T_{tail} \tag{4.14}$$

As we discussed before that T_{ps} and T_{tail} are deterministic for each mobile operator, E_{ps} and E_{tail} will be constant for each given smartphone and mobile data provider. Therefore, these two terms are calculated independently and added to the data energy consumption. This addition is valid under the assumption that each data transfer establishes and uses only one connection at a time. Another assumption can be that signalling was already established for another data transfer and there is a second data transfer happening. The addition can be determined based on the current status of the network interface. To simplify these assumptions, the promotion signalling term is added if the interface is idle otherwise it is not. The addition of the tail energy needs further studies on the prediction of the data arrival.

The term $P_{trx} \times T_{trx}$ represents the total energy consumed for transfer the data, where P_{trx} is the power level of the mobile device adjusted by the Radio Link Control (RLC), and T_{trx} is the total time required to transfer the data over the network interface. As we discussed early, P_{trx} is constant for transferring any amount of data but the time T_{trx} depends on the amount of data (F) and the achieved data rate (R_{trx}) for the given network interface as expressed in the following equation:

$$T_{trx} = \frac{F}{R_{trx}}. \tag{4.15}$$

It is well known that wireless networks suffer from limited resources (*e.g.,* spectrum scarcity), high error rate, and higher delay compared to wired networks. Therefore, recent wireless networks target to increase spectrum utilization and reduce the delay as well [102]. Due to these limitations, especially high error rate, the TCP protocol experiences degradation of its performance. However, in the 3G and 4G mobile networks, new protocols called Automatic Repeat reQuest (ARQ) and Hybrid-ARQ are implemented into lower layers to recover from errors. As a result, performance of TCP is improved since it is almost isolated form wireless channel effect. Nevertheless, TCP is still limited in some cases by the delay occurring in the wireless networks because TCP is end-to-end control protocol, which mean it relay on some sort of information exchange between connection ends.

Based on the discussed characteristics of the wireless networks and TCP protocol, we can express the achieved data rate as

$$R_{trx} = \min\{R_{TCP}, R_{3G/4G}\}, \tag{4.16}$$

where R_{TCP} and $R_{3G/4G}$ are the rate limit due to TCP performance and the scheduler of the wireless networks, respectively. We believe that this expression is practical and simplifies the complex mathematical model developed in [103].

The rate of TCP is defined by the effective TCP Congestion Window (CWD), and the Round-Trip Time (RTT) as expressed in the following equation:

$$R_{TCP} = \frac{CWD}{RTT}. \tag{4.17}$$

The rate $R_{3G/4G}$ is the rate achieved at the TCP layer, which is limited by the rate of the lower layers (*i.e.,* $PDCP$, MAC, PHY). In 3G/4G networks, the rate is adaptive to the channel condition to maximize spectrum utilization. The adaptation is implemented for each Transmission Time Interval (TTI). In the adaptation process, different Modulation and Coding schemes (MCS) are used, taking into account the Received Signal Strength (RSS) and Signal to Noise ratio (SIN). The receiver reports to the transmitter the current channel condition using Channel Quality Index (CQI), which is calculated using RSS and SIN. Then, the transmitter selects the MCS according to the mapping from the reported CQI and the user equipment (UE) category to MCS. This mapping is out of the focus of this work. We use in our calculation the achieved data rate $R_{3G/4G}$ at the TCP layer. However, we conducted a set of field tests to do this mapping and we have the same observation claimed in [104] that the TCP throughput is not affected by the SIN after some threshold of SIN However, we conducted a set of field tests to do this mapping, which will be our future work. The mapping goes from the RSS, SIN to CQI, and then UE category to MCS, and then to $R_{3G/4G}$.

In the TCP protocol, part of the congestion control is the slow-start at the beginning of the connection from Initial Window size (IWD) until it reaches CWD. As we discussed earlier, the power level does not depend on the data rate. Therefore, the mobile device will consume the same power during the TCP slow-start as the power at high rate, as depicted in Fig. 4.3. Hence, Equation (4.15) becomes

$$T_{trx} = T_{ss} + \frac{F - F_{ss}}{R_{trx}} \tag{4.18}$$

where T_{ss} is the time for the slow-start to reach CWD, and F_{ss} is the amount of data transferred during the slow-start stage. Their values can be calculated using the following equations:

$$T_{ss} = RTT \times log_{\gamma}(CWD/IWD), \tag{4.19}$$

$$F_{ss} = TCP_{segment\,size} \times log_{\gamma}(CWD/IWD), \tag{4.20}$$

where γ is the exponential growth of the window size, usually it takes a value of 2.

4.5 Experimental Validation

In this section, we set up and conduct a set of experiments to validate the energy models. We measure the actual energy consumed in five different smartphones in real circumstances and a real cloud.

4.5.1 Methodology

We set up our experiments as depicted in Fig. 4.2. In this setup, we use five different types of smartphones: *HTC Nexus One, LG Nexus 4, Samsung Galaxy S3, BlackBerry Z10* and *Samsung Galaxy Note 3*. These smartphones can access WLAN, 3G, or 4G networks. With these networks, the smartphones upload and download files to and from the cloud. The power supply can simultaneously power the smartphone and record the power consumption. The power readings during the experiments are recorded on a laptop designated for this purpose. To reduce the errors in the reading, we duplicate some experiments using two types of power supplies: *Keithly* and *Monsoon*. We conclude that the error is less than 1%.

The total energy consumed in a smartphone for a communication task is the sum of the energy consumed by several system parts as given in the following equation:

$$E_{total} = E_{WNI} + E_{OS}, \tag{4.21}$$

where E_{total} is the total energy consumed for a communication task, E_{WNI} and E_{OS} are the energy consumed by the wireless network interface (WNI) while transferring data, and by the operating system (OS), respectively.

Our models are developed to calculate the energy consumed for data transfer as represented by E_{WNI}. The aim of our experiments is to validate our energy models. However, in our experiments, the overhead energy consumed by the operating system is unavoidable. The E_{OS} term is determined experimentally, and consequently, we distinguish the energy consumed for transferring data from the total measured energy during the communication. Figure 4.3 shows the real time power consumption of a smartphone when the system is idle (low power consumption) and when a data block is transferred (the high power consumption). Hence, to compare our model with the experimental measurements, we need to add the energy consumed by the operating system to the models. Throughout this section, the comparison between experimental results and models is presented with respect to the energy consumed in the wireless interface E_{WNI}.

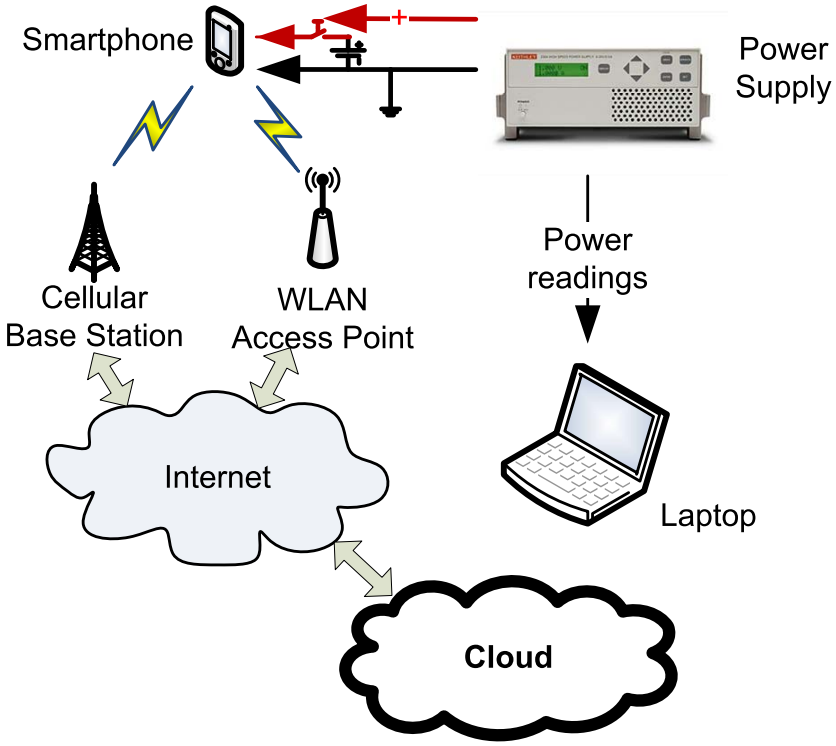


Figure 4.2: Experiments setup

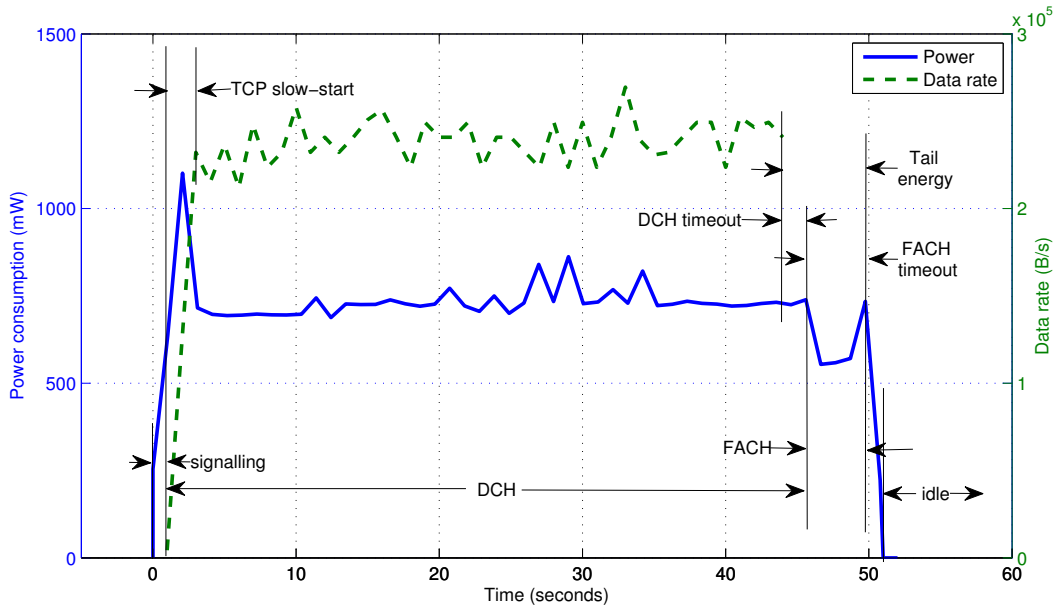


Figure 4.3: Example for power and data rate for different TCP and RRC status

For these experiments, we choose a 10 *Megabytes (MB)* file to represent average multimedia files. We use the same file for all of the experiments to keep the results consistent. Figure 4.4 shows a comparison between the cumulative energy consumption of a smartphone obtained from experiments (total) and the total energy calculated by our models (Model). Figure 4.4 also shows the energy consumed by the operating system (E_{OS}) after we separated it from the total energy experimentally, and the energy consumed by WNI (E_{WNI}) after we calculated it using Eq. (4.7). The results shown in Fig. 4.4 reveal that our energy estimation model is very accurate. This figure shows the energy consumed in system parts to demonstrate our methodology for our experiments. Hereafter, we only show the total energy obtained by the experiments for the wireless interface and by the mathematical models.

As the Internet traffic is bursty, bursts keep the wireless interface in the inactive mode, or in the idle mode (*i.e., power saving mode*) if the waiting time for a traffic exceeds a threshold amount [92]. To accurately measure the energy consumed during traffic exchange, bursts traffic is avoided. One way to tackle this problem is to limit the traffic rate at the server. For this purpose, we conduct set experiments on bursty traffic and non-bursty traffic, then we compare their TCP traces, as shown in Fig. 4.5. This figure depicts the

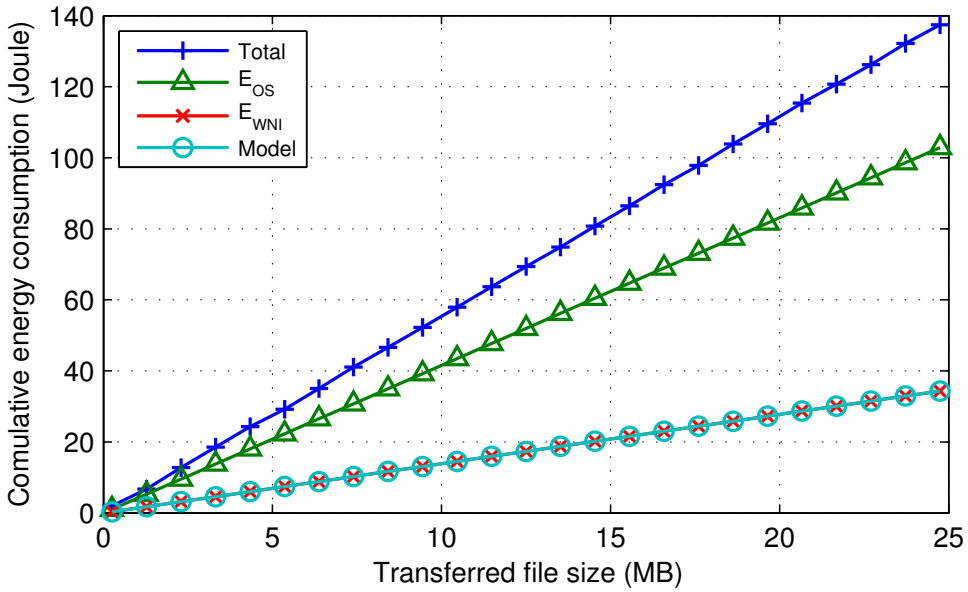


Figure 4.4: Total energy consumption for file downloading over WLAN

TCP trace, where packet arrival time is shown on the x-axis and the amount of transferred packets on the y-axis. The Bursty-Traffic line represents the flow of a bursty traffic. We notice that most of the packets arrive at relatively short time, which is called bursts, and few packets arrive on much longer time, which causes the interface to use power saving mode. Moreover, we notice that the time between receiving bulk of packets is random. This observation explains why the bursty traffic leads to inaccurate energy estimation, because it keeps the wireless interface idle for random amount of time. To reduce the impact of bursty traffic on energy consumption, we make sure there is no idle period during the network activities when we measure the energy consumption. We obtain this by running a set of download and upload tests and monitor the burst of the traffic using network analysis software *"Wireshark"*. The resulting traffic is smooth as shown using the network analysis software, by the Smooth-Traffic line in Fig. 4.5. This line shows that packets arrival is uniformly distributed over the transfer time, where there is no idle time for the wireless interface. This leads to accurate energy measurement for data transferring. For a more detailed examination of the impact of traffic burstiness, see [105].

In our experiments, we perform more than 30 sets of experiments involving file downloading and uploading over a WLAN network, and again for file downloading and uploading over 3G and 4G networks. Each set of experiments is repeated between three to five times.

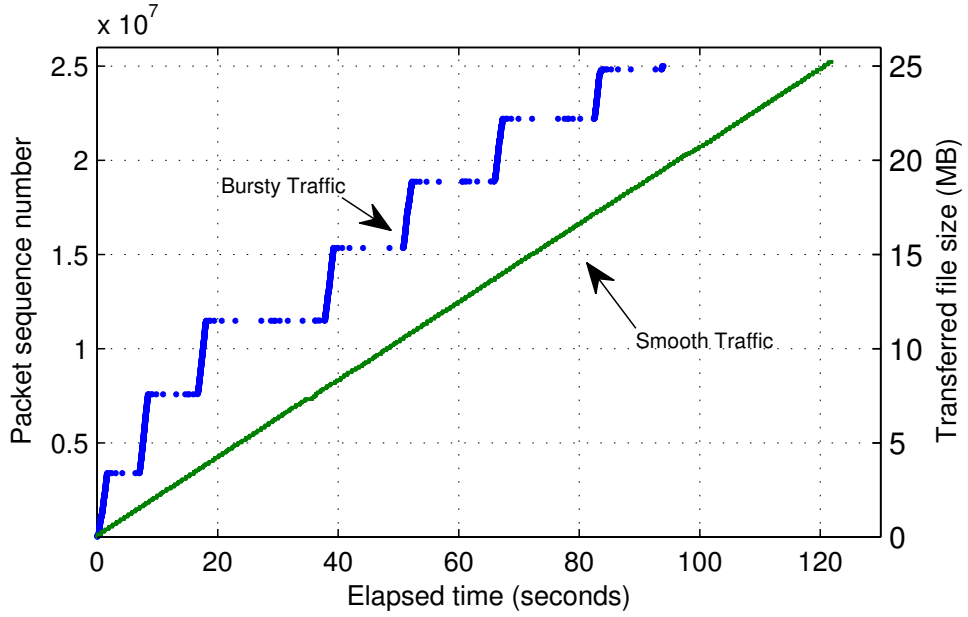


Figure 4.5: TCP trace: Time versus file size

The results of our experiments reveal that all tested devices have the same behaviour of energy consumption during network activities. We obtain consistent results among the devices, which emphasizes that our models are device independent and applicable to a wide range of devices. The only difference is the amount of power consumption, as summarized in Table 4.2. Since all devices behave similarly, we present an extensive statistics of the experimental results only for the most modern device that we have at the time of our experiments, namely, *Samsung Galaxy Note 3*. Moreover, we will not present the statistics of the results for all tested devices due to limited space. On the other hand, all devices achieve similar TCP throughput, which we show in this section.

4.5.2 File Transfer over WLAN Networks

We conduct our experiments for real circumstances and we confirm some parameters from our experiment settings. Table 4.3 lists the values that we obtained from the experiments for the parameters used in Eq. (4.1) to Eq. (4.11) and not listed in Table 4.1.

In the first set of experiments, we measure the total energy consumed by the smartphone during downloading a large file ($10\,MB$) over a WLAN network to validate our

Table 4.2: Average power consumption (mW)

		Smartphone				
Network	Activity	UE1	UE2	UE3	UE4	UE5
WLAN	Download	485	580	670	1010	1044
	Upload	830	780	850	1140	1280
3G	Download	730	700	1080	950	730
	Upload	750	711	1125	1025	750
4G	Download	NA	NA	1100	965	1250
	Upload	NA	NA	1130	1220	2300

UE1: *HTC Nexus One*, UE2: *LG Nexus 4*, UE3: *Samsung Galaxy S3*, UE4: *BlackBerry Z10*, and UE5: *Samsung Galaxy Note 3*

Table 4.3: Parameters obtained from the experiments

Parameter	Value
B	25 MB
R_{data}	54 $Mbps$
L_{max}	1448 $Bytes$
F_s	1448 $Bytes$
$N_{d_{seg}}$	3
$N_{u_{seg}}$	8
ACK_{TCP}	32 $Bytes$

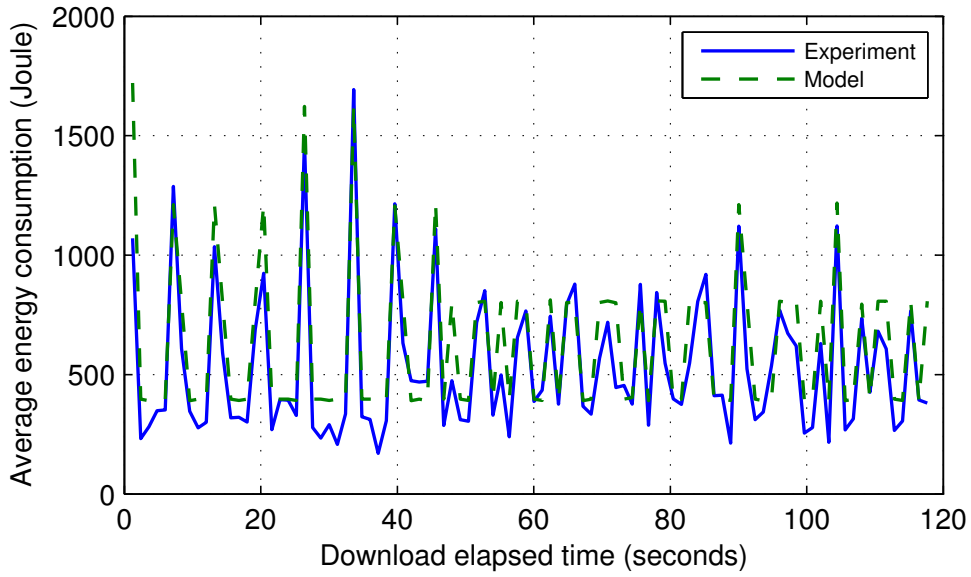


Figure 4.6: Experiment measurements and estimation model

energy estimation model in Eq. (4.7). Figure 4.6 shows a comparison between real time experimental measurements and our energy estimation model for downloading a file over a WLAN network. The cumulative energy is the sum of consumed energy for a task from the beginning of the task to a given time. However, the cumulative energy consumed during downloading a file is actually what is drained out of the smartphone battery. For that, we compare the cumulative energy obtained from our experiments and our energy estimation models that we developed in Eq. (4.7) for the download case. Figure 4.7 shows the cumulative energy obtained from our model. The small vertical bars represent the 95% confident interval of the experimental results around the models.

In the second set of experiments, we conducted similar experiments, but for file uploading to validate Eq. (4.11). Figure 4.7 shows a comparison between the cumulative energy measure in the experiments and the cumulative energy calculated from our model for file uploading case.

4.5.3 File Transfer over 3G and 4G Networks

We conducted a set experiments to validate the energy estimation model for 3G and 4G networks introduced in Eq. (4.13) for file transfer. We used the *Wireshark* software to

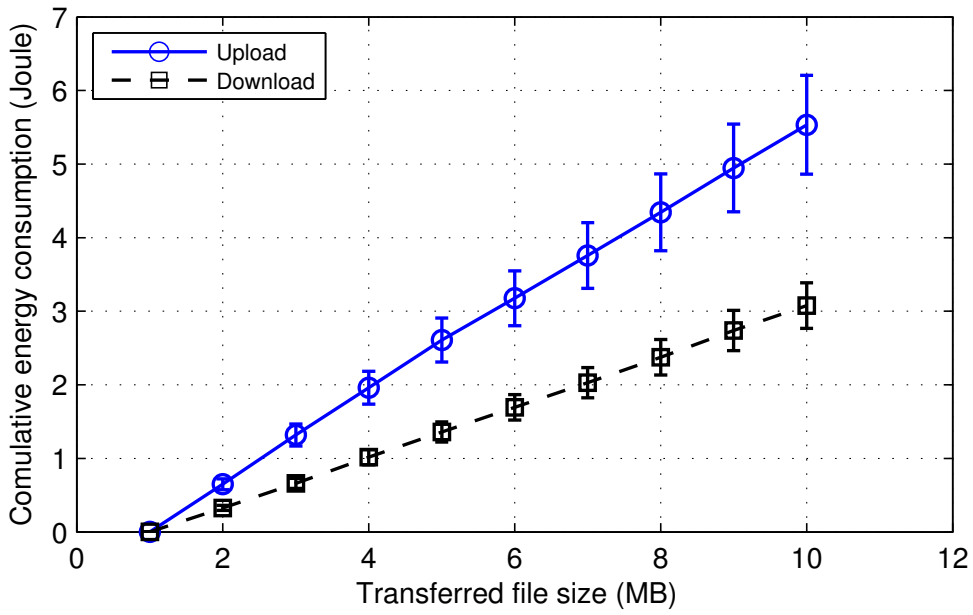


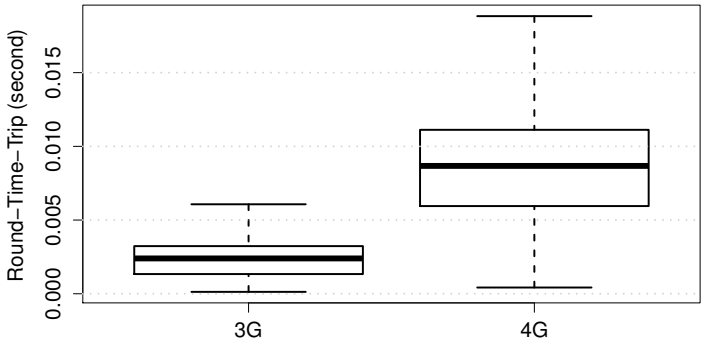
Figure 4.7: Energy consumption for WLAN versus file size

determine experimentally the value of TCP throughput, RTT, IWD, CWD, and RCC timers. Table 4.4 lists the parameters of RRC that we obtained experimentally.

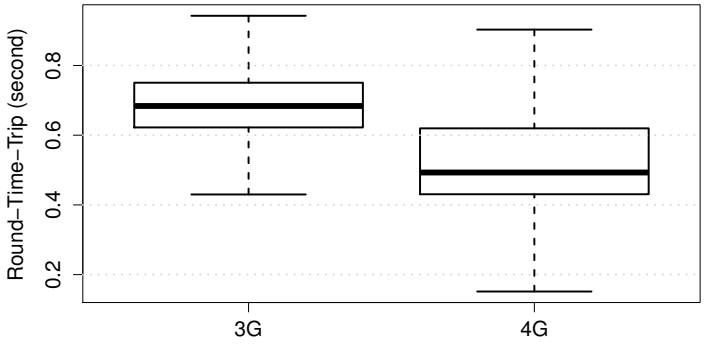
Figures 4.8, 4.9, and 4.10 show the experimental statistics of RTT, TCP throughput, and power consumption, respectively. In Fig. 4.8, we notice that the values of RTT in the upload cases are much higher than the values in the download cases. Therefore, the data rate of the uploading cases is limited by the TCP rate due to high RTT. In contrast, the data rate is limited by the network rate for the download cases.

Figures 4.11 and 4.12 show the energy consumed for transferring different amount of data using 3G and 4G networks, respectively. The solid lines show the energy calculated using our proposed models, where the bars represent the amount of energy that the experimental results deviate from the models with 95% confidence interval.

The standards of 4G networks adopted multiple-input and multiple-output (MIMO) to be used whenever a UE has the MIMO capability to enhance the performance of the wireless links. For this reason, we examined the MIMO capability on all of our devices and found that only UE5 has this capability. In the case of using 4G networks, Fig. 4.12 depicts a comparison between the cumulative energy consumption for UE5 with MIMO capability and without it, which is called single-input and single-output (SISO).



(a) Downloading RTT



(b) Uploading RTT

Figure 4.8: RTT statistics

Table 4.4: RRC parameter values

	Parameter	Value
3G	$t_{signalling}$	$\simeq$ 1 s
	$t_{DCH-FACH}$	6.3 s
	$t_{FACH-PCH}$	3.7 s
	E_{ps}	0.56 J
	E_{tail}	6.61 J
4G	$t_{signalling}$	$\simeq$ 1 s
	$t_{Active-ShortDRX}$	2.5 s
	$t_{ShortDRX-LongDRX}$	10.5 s
	E_{ps}	0.45 J
	E_{tail} download	7.1 J
	E_{tail} upload	9.31 J

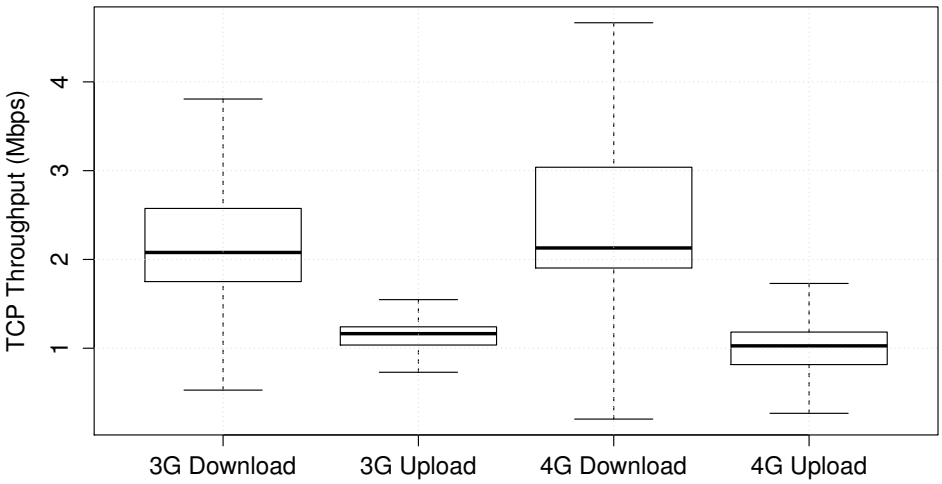


Figure 4.9: Statistics of TCP throughput

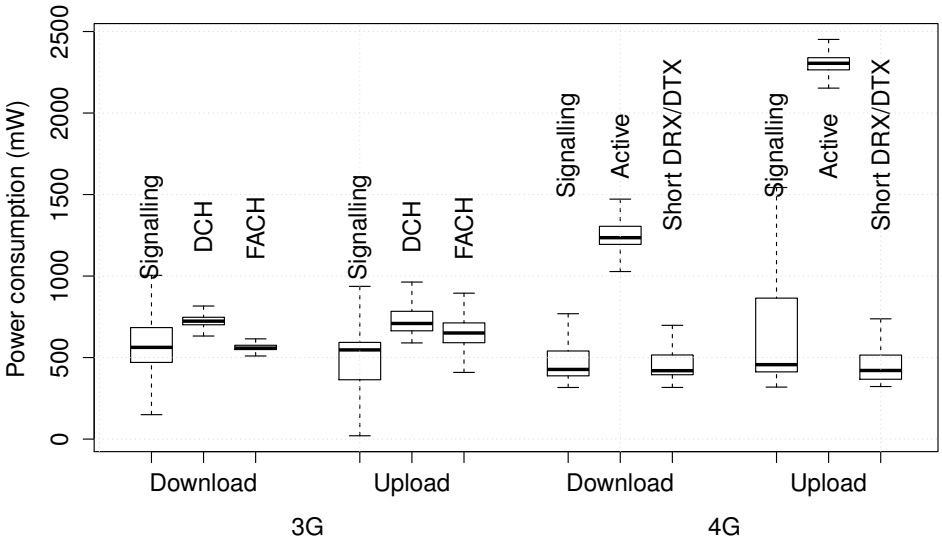


Figure 4.10: Statistics of mobile power consumption

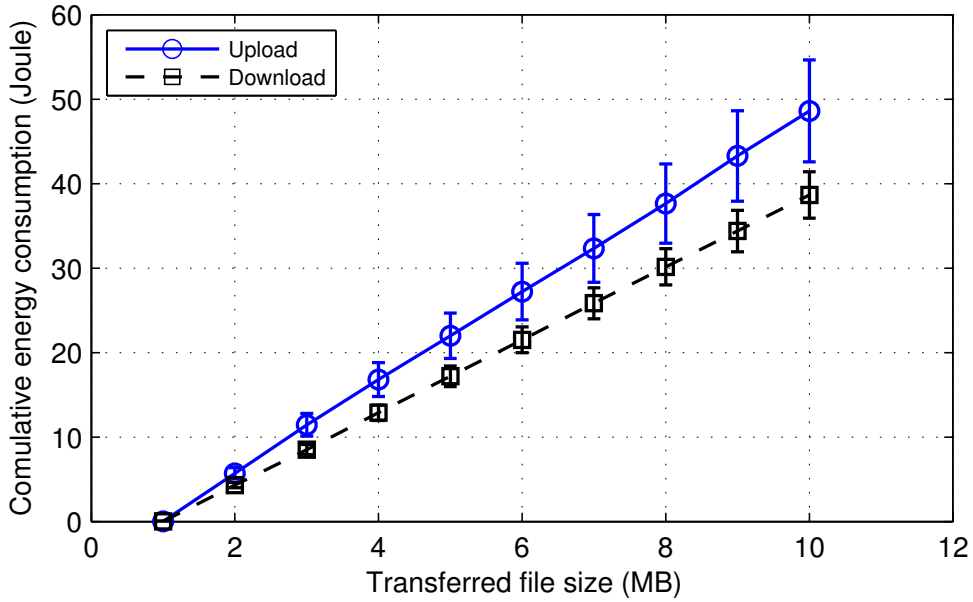


Figure 4.11: 3G energy consumption

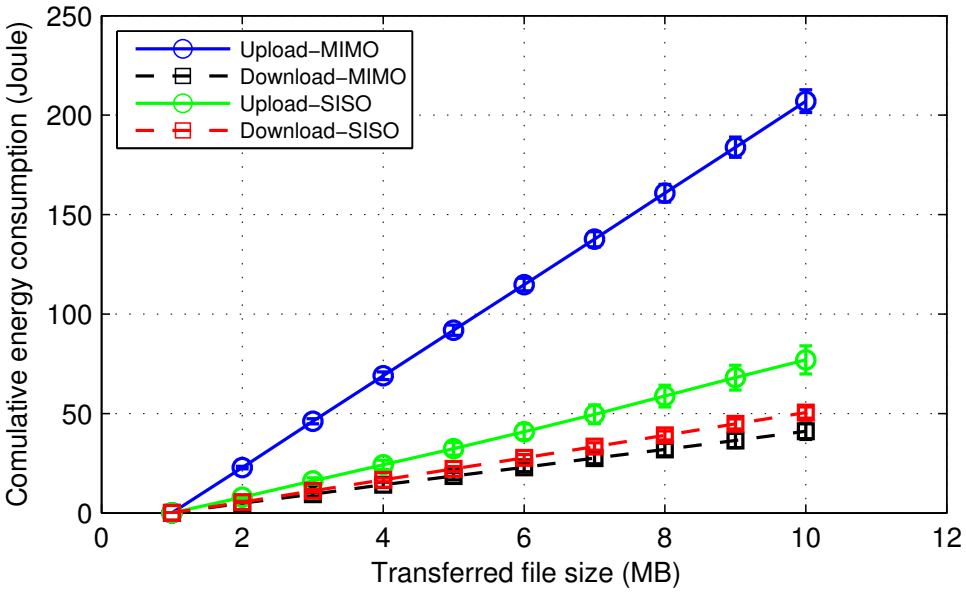


Figure 4.12: 4G energy consumption

4.5.4 Offloading Case Study

In this subsection, we examine the energy estimation models in case of task offloading. As a case study, we consider the second offloading scenario (S2), which is depicted in Fig. 3.8, because it involves file uploading and downloading. Therefore, we have this scenario as a benchmark of our models to show their accuracy. Robust estimation of this scenario leads to make reasonable offloading decisions; specially, decide between scenario S1 and scenario S2.

The estimated energy is computed by only knowing the transferred file size (F) using Eq. (4.7), Eq. (4.11), and Eq. (4.13). Based on the models, we study scenario S2 for offloading a task, which encodes a video from one video format to another. This scenario involves uploading a 23.97 MB video clip in *flv* video format, doing the encoding in the cloud from *flv* to *mp4* video format, and then downloading a 8.21 MB video clip in *mp4* format. The details of encoding the video files are presented in [8]. Since the size of the transferred files is known, we can use our energy estimation models to calculate the energy cost on a smartphone that is consumed to perform the encoding offloading.

Figure 4.13 shows a comparison between experimental results and estimation models for WiFi and 3G networks. This figure presents the total amount of energy consumed during

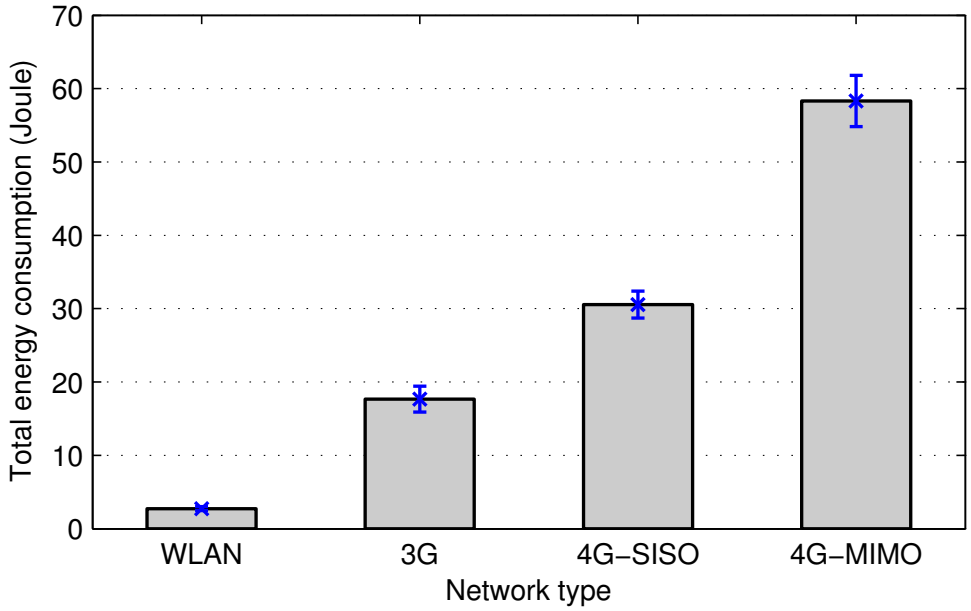


Figure 4.13: Total energy consumption for an offloading case study

23.97 MB file uploading, 8.21 MB file downloading, and total task offloading. The vertical bars represent the deviation of our models from the experimental results. Note that the offloading involves both the uploading and downloading activities. As a result, the total energy consumed in offloading is the sum of the energy consumed in both of uploading and downloading activities. These results indicate that our models accurately estimate the energy required for complete a task offloading. In addition, the results emphasize that our models realistically estimate the energy consumed in the smartphone, which can reach a correct offloading decision.

4.6 Summary and Discussion

The proposed energy models of WLAN, 3G, and 4G interfaces will allow smartphones to make correct offloading decisions. We considered the details of the network stack from lower level up to the transport level. This work estimates the energy cost by simple computations without overhead energy cost that could be caused by complex mathematical calculations. The models just need to know the amount of transferred data and some system parameters, and they can provide good estimations of energy cost. Nevertheless, the introduced

models accurately estimate the task offloading energy consumption for 3G/4G and single user WLAN networks. Moreover, our models not only helps for task offloading but also opens new door for energy solutions that require predicting the energy consumption. We emphasize that our models are sufficiently accurate, simple, and extendible.

In fact, we consider the simple energy model, which equals the transmission power times expected transfer time, but when it is possible such as for the 3G/4G networking. This is because; the power level takes a specific value at each state of RRC. However, this is not the case in the WLAN (*i.e., IEEE 802.11g*) since the power level is different for different data rate and it is change every time-slot. Therefore, we consider the details for the MAC timing and the probability of transmission for each time-slot.

We limit the WLAN models to the *IEEE 802.11g* standard but we are able to model for *IEEE 802.11n* standard in the same approach and analysis we used for *IEEE 802.11g*. However, one of the main features in *IEEE 802.11n* is the MIMO spatial diversity that are missing in all of current smartphones in the WLAN interface. They are only feature by single WLAN antenna, which degrades the system to work as *IEEE 802.11g*. We have experimentally approved this at the early stage of our work. For that reason, we defer our work on *IEEE 802.11n* to the future work. In contrast, we consider the case of MIMO in the 4G modeling since the 4G interface is featured with multi-antenna (*e.g., Samsung Galaxy Note 3* has two 4G antennas).

We would mention to the fact that the issue of burst traffic is only for the WLAN networking. In the 3G and 4G networking, there is no burstiness experienced due to the protocols of these networks that assign a dedicated data channel for each device during data transferring. We developed our models to estimate the energy consumption for file transferring. Therefore, it is intuitively that our modeling was developed to compute the energy per bytes. Regardless of the shape of the traffic, our models predict the energy consumed for any given transferred data. However, we use smooth just for the case of WLAN and just for experimental purpose. As we elaborated, we smooth the traffic to avoid the impact of the power saving mode, which could occur in the time between the bursts. Moreover, the time between the bursts is random and modeling the randomness of this time is out the scope of our work. Xiao et al. [92] discuss this issue and show the impact of the burst traffic.

We would emphasize that our model can also compute the energy consumption of burst traffic by considering the energy for the bursts and the time between them. The bursts energy is computed similar to compute for smooth traffic but with consideration of the data rate of the bursts. On the other hand, the energy consumed during the time between the bursts can be computed by multiply the idle power by the time duration of that period.

The problem is that the time is not deterministic, which is most likely statistical, and if it is long enough the power saving mode will be triggered. In this case, experimentally examine the models will be complex, where we need to extract the power saving mode parameters such as the mode timeout.

Todays WLAN system is a multi-user, multi-channel system supporting data transfer with PCF (Point Coordination Function) and DCF (Distributed Coordination Function) modes of operation (with hand-shake and without hand-shake) in an environment with varying channel quality. Ideally, there is a need to develop energy models that encompass the full range of capabilities and constraints of the WLAN system(s). However, we began developing a model for the simplified case with the objective of incrementally developing a model for the general case.

Chapter 5

Modeling the Hardware Energy Consumption

Modelling the energy consumed in the task execution is crucial to help the developers to build energy efficient applications. Therefore, the major challenge in the modelling approach is to accurately estimating the energy consumed for an application by the hardware components, such as CPU, memory, storage unit, and network interfaces.

In this chapter, we develop and validate hardware and software profiling models and procedures. We profile the smartphone CPU, where we consider multi-core CPUs and the impact of Dynamic Voltage and Frequency Scaling (DVFS) mechanism on the power consumption. In addition, we profile the smartphone storage unit by taking into account the writing and reading rate to the unit. Moreover, we experimentally validate these profiles on two diverse smartphones with different versions of the operating system. These profiles empower mobile developers to estimate the energy consumption of an application on a given hardware configuration. Thus, we demonstrate on a real case study how the developers perform the profiling. In a case study, we profile an example application, apply the developed CPU and storage unit profiles, and experimentally validate them. The experimental results reveal that our profiles are able to estimate the application energy accurately. The contributions of this chapter are published in [10].

5.1 Preamble

In the recent years, the problem of limited energy capacity of mobile devices became significant after the emergence of a wide range of applications, many of them being not

energy efficient. The energy aspects of the devices in many cases have not been addressed. For instance, some online video streaming and web browser applications are not energy efficient as in the finding of Abogharaf et al. [97] and Albasir et al. [98]. The problem is that the developers have no clue about the energy consumption of their application on a particular device. As a result, many of today's mobile applications do not utilize the device resources efficiently. The cause of the problem is the existence of a knowledge gap between hardware and application developers, where no energy models of the hardware are provided to the application developers. Moreover, the social networking encourages mobile users to use more multimedia than before especially with the capability to of the devices to produce and process multimedia contents. These contribute to the concern regard the energy of the devices.

Developing energy-aware applications needs information about the energy consumption of the hardware components at the application level [106, 107]. This is achievable by means of developing hardware and application profiles. The hardware profile contains a mathematical description of the power consumption for the hardware components at different operating levels (*e.g.,* idle, standby, and active). On the other hand, the application profile is the mathematical description of the use of the hardware components by a given application. In the profiling method, the developers use the hardware profile to estimate the total energy consumed for running an application on a given hardware as depicted in Fig. 5.1. The energy profiling not only helps the developers to estimate energy consumption for their application but also opens new door for energy solutions that require prediction of the energy consumption. For example, task offloading from a mobile device to the cloud needs to predict the energy consumed for the task before making decision whether or not to offload the task. In this case, profiling the hardware allows the device to predict the energy consumed by the device on the task in the cases of executing the task on the device or offloading the task to the cloud; and consequently, the decision is made based on whether or not the offloading saves energy.

To profile the energy consumption of a mobile device, it is crucial to identify the most hardware components consume energy. In the modern mobile devices, it has been validated that the LCD screen, networking interfaces, memory, CPU, and data storage unit (*i.e.,* sd-card) are the most energy consumers [108]. The energy consumed by the LCD screen is linearly proportional to the level of the brightness and the displayed color [109, 20, 109]. However, the energy consumed by the LCD screen does not impact the energy consumed due to running an application. Therefore, we exclude LCD screen from our profiling procedure. To determine the energy consumed by the memory, the memory activities are associated with the CPU activities. Figure 5.2 shows the maximum power consumption of the hardware components: CPU, Storage Unit (SU), and Memory (MEM). This figure

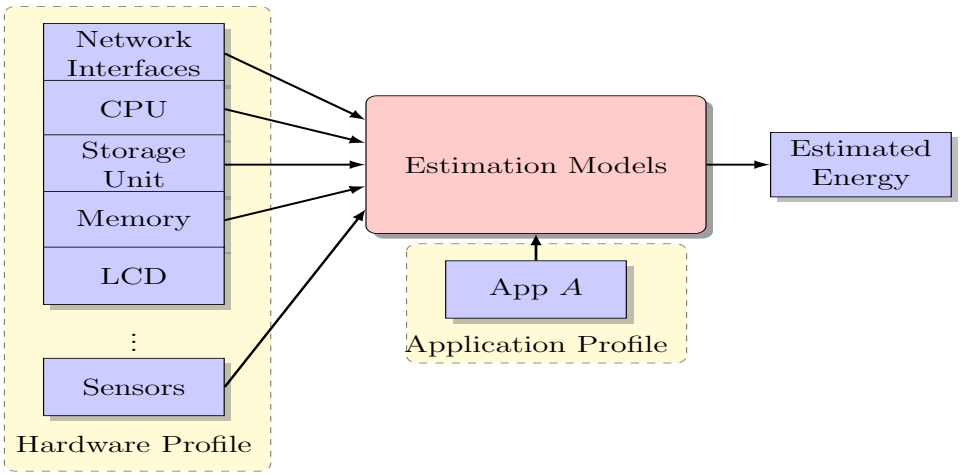


Figure 5.1: Profiling overview for energy estimation

reveals the memory energy is less than 3% compare to the CPU power consumption. Therefore, the energy consumed by the memory is considered as a part of the energy consumed by the CPU. In fact, the energy consumed by the memory is very small compared to the energy consumed by the CPU, which usually be ignored [108]. Profiling the energy consumed by the CPU and the storage unit is the objective of our work in this chapter.

In this work, we present a procedure to profile the mutli-core CPU and the storage unit of smartphones. We develop the procedure to be applicable to many smartphones. The result of the procedure is mathematical models to profile applications without running additional tests. These models accurately describe the hardware energy consumption. To summarize, this chapter makes the following contributions:

- Profile multi-core CPU and storage unit of smartphones, where we model the power consumption in these hardware components;
- Consider the impact of the *Dynamic Voltage and Frequency Scaling (DVFS)* mechanism and the number of online cores on the CPU profile by modeling the CPU power consumption as a function of the CPU clock frequency, and the number of online cores, respectively;
- Model the power consumption of the storage unit as a function of the system clock frequency and the rate of writing and reading tot eh unit.

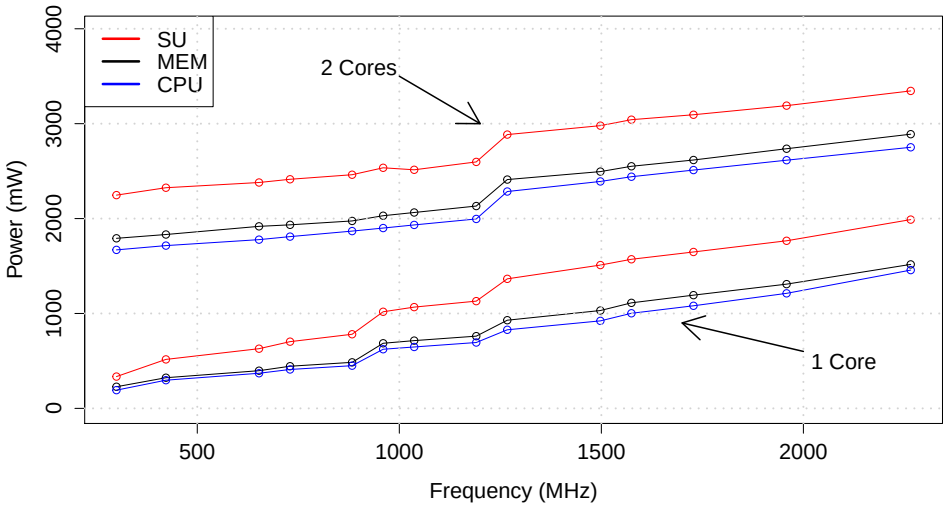


Figure 5.2: Power consumption of the mobile hardware components

- Profile an application as a case study to demonstrate the use of hardware components; and
- Validate experimentally the profile models and procedures.

This chapter is organized as follows. The literature of modelling and profiling the energy consumption of the hardware components are discussed in Section 5.2. In Section 5.3, we explain and develop profiling models and show the profiling procedure that we follow to extract the models parameters. In Section 5.4, we present a case study for our profiling procedure, and show how to apply the energy estimation models. Some summary and concluding remarks are given in Section 5.5.

5.2 Hardware Profiling Literature

As the problem of limited energy in mobile devices becomes more acute, more research is needed to tackle this problem. Multipurpose systems, like the ones on mobile devices, are very complex; consequently, it is difficult to predict their power consumption [110]. The power profiling approach is a promising one to tackle the problem [111].

In the literature, there are two approaches to profiling: post-profiling and pre-profiling. Post-profiling provides information about the power consumption of an application on a given device after running the application. In this profiling approach, the power information is only available for the post processing. In contrast, pre-profiling predicts the power consumption of an application on a given device before the application is started. In this approach, the prediction is based on the power profiles expressed as lookup tables or mathematical equations. This approach is important for making an offloading decision such as whether or not to run an application to save energy.

Bugu is a post-profiling approach that drives the relationship between events and power consumption on mobile devices to profile an application [112]. The system consists of two parts: a *Bugu* server and *Bugu* client. The *Bugu* server collects the application power information and provides it to *Bugu* clients after analyzing the power and the events from the device. On the other hand, the *Bugu* client monitors the application power consumption. *Bugu* does not breakdown the total power consumption to the power consumed by the hardware components. Moreover, this approach does not consider hardware configurations like online CPU cores and DVFS. *PowerScope* is a post-profiling system that profiles both the hardware components and the application [113]. The energy consumption of the hardware components is measured by external power meters. At the same time of measure the power, software performs a statistical analysis to the system activities. The hardware profiling is computed off-line by combining the statistical analysis with the measure power. To profile an application, *PowerScope* analyzes the application and build mapping from the application structure to the system events that statistically provided from the hardware profiling.

In contrast to *Bugu*, *Devscope* [91] and *PowerBooter* [90] are pre-profiling approaches that profile the hardware components of a mobile device by analyzing the access to the components and the change in the power state [91]. The power state is provided from the *Battery Monitoring Unit (BMU)*. The output profiles are expressed in the form of lookup tables and equations. The used method would be accurate if it uses an accurate power meter since *BMU* is known to update at a very low rate and provides readings with very low accuracy. It cannot trace very short events that are shorter than *BMU* update rate as in the case of the Forward Access Channel *FACH* change in mobile networking where the results show the error to be higher than 30% [91]. In addition, this system has to be installed on a modified version of *Android OS* to give the ability of profiling the applications dynamically. This work also does not consider the power consumption of the storage unit or multi-core CPUs. Since the hardware components consume a fixed amount of power at a given state, there is no need for a dynamic system and one-time measurements is enough to profile a mobile device. In addition, the BMU always need

celebration and battery readings become unreliable overtime. The authors argue to use dynamic system. Therefore, one time measurements is absolutely enough

Appscope [114] can profile an application using the models provided by *Devscope*. *Appscope* uses the same concept of *Devscope* by monitoring the system events at the kernel level for the call of the hardware components by the application. Similarly, *pTop* [115] and *Eprof* [116] estimate the energy consumed for an application by tracing system calls for the application. The energy of an application is computed by using the information of the power consumption of the hardware components that the application uses, and the time the application needs these components that is mathematical expressed as $E_{app} = \sum_{i \in n} P_i * t_i$, where E_{app} is the total energy consumption for the application *app*, P_i is the average power consumption of the component i, t is the total usage time for the component i by the application, and n is the set of hardware components accessed by the application *app*. For instant, *Devscope* provides P_i while *Appscope* measures t_i to calculates E_{app}. *eCalc* [117] and *eLens* [107] use the same approach of *Appscope* but it analyzes the application at the code-level. The application profiles in *eCalc* and in *eLens* show how much of the code uses each hardware component by tracing the code at the development environment. Then, an energy cost function estimates the energy consumed for each type code instructions to profile the hardware components. The code is analyzed and the code instructions are counted by the code analyzer. Based in that, the total energy is the sum of the calculated energy from the energy cost functions. If the models from Devscope are accurate, the result of Appscope will be accurate too. Therefore, we emphasis that Appscope is accurate using our profiling models.

In our profiling methods, we use an accurate approach that precisely measures the power consumption at any system event. In addition, we consider the impact of *DVFS*, multi-core CPU, and storage unit of mobile devices. The only study, to the best of our knowledge, considers the *DVFS* mechanism is the study presented in [118]. However, this study is implemented for BitsyX platform and not for mobile system, where the goal of is to determine the optimal operating frequency at which the CPU consumes minimum energy for a given task.

5.3 Profiling Models

In this section, we present our profiling models and procedures for smartphone multi-core CPUs, storage units, and applications.

5.3.1 CPU Profile

Modern mobile devices are loaded with many types of sensors and multimedia producers, such as GPS, accelerometer, compact camera, and microphone. Consequently, there is a growth in multimedia applications on mobile devices especially with the emergence of social networking, which encourage mobile users to use more multimedia than before. Therefore, many of today's mobile devices are equipped with mutli-core CPU to accommodate the growing computing need. However, these CPUs consume much more energy than single core CPUs. Therefore, the CPU of mobile device have been given the ability to turn-off some cores as needed in contrast to the desktop computers, which usually keep all cores on. The manufactures of the hardware or the assembler usually develop a CPU driver to control the CPU cores. This driver enables the *Operating System (OS)* to control the CPU cores. Thus, the decision of how many cores are needed is implemented in the *OS*. To demonstrate the effect of multi-core CPU on the power consumption, we conducted experiments on playing video streamed from a remote server.

Figure 5.3 shows the total power consumption over the time of playing a *YouTube* video on a 4-core smartphone, namely *Samsung Galaxy Note 3*. This figure shows the power consumed for different number of online cores for playing the same video after we enforce the *OS* to the desired number of online cores. Moreover, we compared that with the power consumed after we let the *OS* selects the best number of online cores based on its own algorithm. The *OS* algorithm increases the number of online cores if the *OS* foresees high computation tasks, as it is shown in the figure at the beginning of the buffering stages. Then, the *OS* reduces the number to 1 online core when the computation is low. We notice from the figure that the power consumption goes through a deterministic cycle of states based on the number of online cores. The total energy consumed are: 764, 808, 813, 839, 787 Joules, for the case of 1,2,3,4 online cores, and dynamic OS configuration, respectively. In this work, we target a multi-core CPU since they will become a common on the mobile devices.

To model the energy consumption of the CPU, we should mention to two important facts about the power consumption of a modern CPU in general. First, the power consumption is a function of the CPU clock frequency; and second, the CPU power cost is proportional to the CPU usage. In this subsection, we show and describe these relationships in detail.

First, the total power consumed by a CPU is the sum of the static and the dynamic power [119]. The static power (P_s) is the dissipated power caused by the leak current. This power is constant for each CPU chip design. On the other hand, the dynamic power is the dissipated power cause by the switching activities of transistors. This power (P_d) linearly depends on the capacitance (cp) being switched inside the CPU chip, the square of the

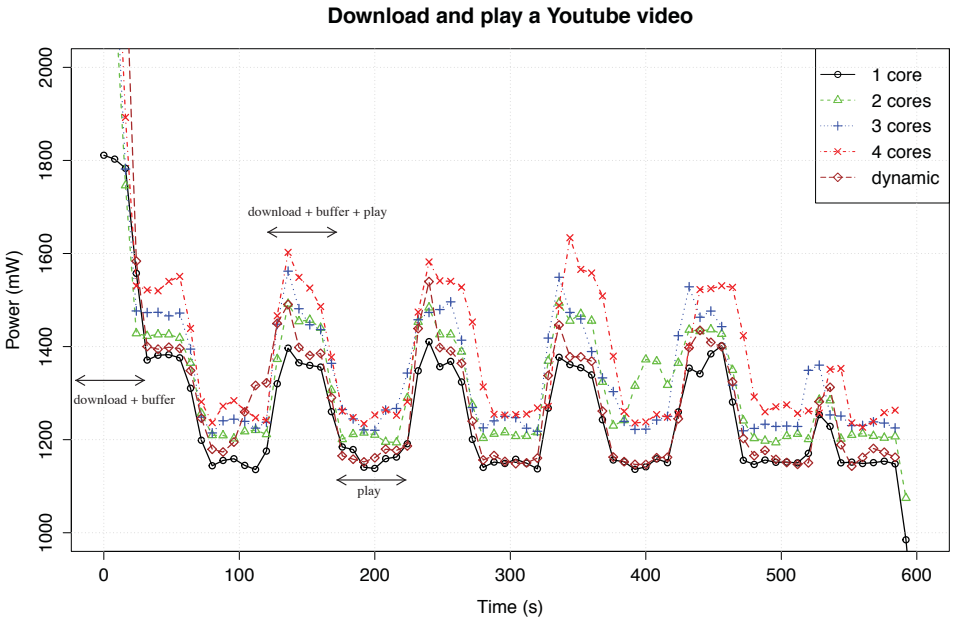


Figure 5.3: Power consumption for playing a *Youtube* video in a multi-core CPU

CPU supply voltage (v), and the CPU clock frequency (f) [120, 119]. The mathematical expression for the dynamic power is expressed as:

$$P_d = cpv^2 f. \tag{5.1}$$

The value of the capacitance is constant for each individual CPU, which depends on the CPU chip design. On the other hand, the values of the supply voltage and the clock frequency have been set up to be constant, but modern CPU technologies adopt to set these values dynamically through a technique called *Dynamic Voltage and Frequency Scaling (DVFS)*. The advantage of adjusting the voltage or the frequency dynamically is to reduce the CPU power consumption. This technique is widely deployed in current CPUs, especially for the mobile devices, where energy is a scarce resource. A programmable clock generator is used to change the frequency and a *DC-DC* converter to change the voltage. The mechanism to change the voltage is a built-in feature of the CPU, whereas the frequency can be controlled by the *OS*.

Since we do not have control on the voltage, we combined the capacitance and the voltage notations in Eq. (5.1) into one notation called α. As a result, the total power dissipation of the CPU follows the following general equation:

$$\begin{aligned} P &= P_d + P_s \\ &= \alpha f + \beta \end{aligned} \tag{5.2}$$

where β is a constant representing the static power and the power consumed independently of the clock frequency, which is the power to activate the circuits.

Equation (5.2) considers only a single core CPU. In fact, a multi-core CPU acts as a combination of single core CPUs. Hence, the total power consumption of a multi-core CPU denoted by P_{mc}, will also be a combination of the power consumed by each core, as represented in Eq. (5.3).

$$P_{mc} = P_b + P_c \times n \tag{5.3}$$

where P_b is the base power to activate a multi-core CPU, P_c is the power consumed in each individual core, and n is the number of active cores. We multiply P_c by n because CPUs are identical cores. Consequently, the multi-core version of Eq. (5.2) is written as:

$$P_{mc} = \alpha_b f + \beta_b + [\alpha_c f + \beta_c] \times n \tag{5.4}$$

Another fact concerning power consumption is that the CPU consumes power as a function of the CPU usage (*i.e.,* utilization) [121]. The CPU usage is defined as the function of time for which a CPU is executing instructions. The relationship between the CPU power consumption and the CPU usage is a linear relationship as expressed in Eq. (5.5).

$$P_{cpu} = P_{min} + (P_{max} - P_{min}) \times U \tag{5.5}$$

where P_{min} and P_{max} are the power consumed by the CPU at zero and 100% utilization, respectively, and U is the CPU utilization at a given time when the power is calculated. This observation validated after we conducted and experiments to measure the average power at different CPU utilizations on *Samsung Galaxy Note 3*. We obtained exact power values that match Eq. 5.5, where $P_{min} = 400\,mW$ and $P_{max} = 1600\,mW$. Figure 5.4 shows the real-time power consumption of the CPU at several levels of CPU utilizations.

To calculate the total power consumed by the CPU, we use Eq. (5.5), where P_{min} and P_{max} are calculated using Eq. (5.4) at zero and 100% utilizations.

5.3.2 Storage Unit Profile

The storage unit consumes power based on the amount of data written/read into/from the unit, because the unit only activates the cells that are targeted for the writing or reading

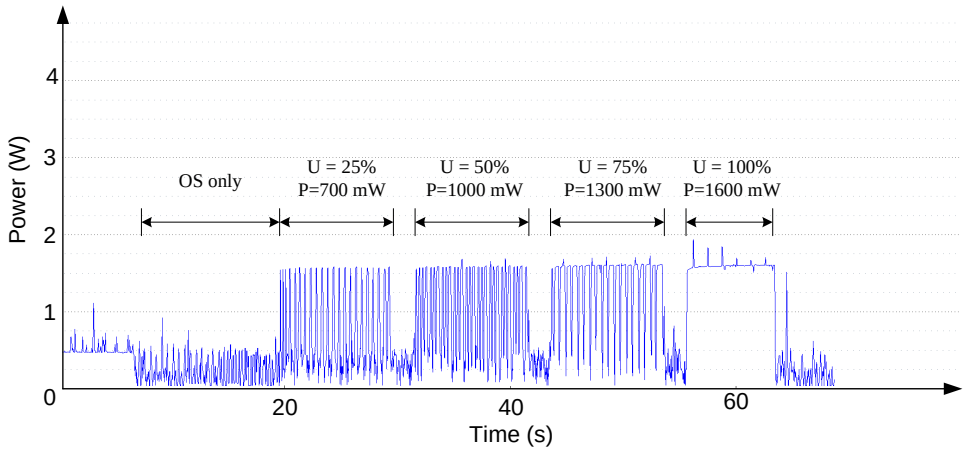


Figure 5.4: CPU power consumption at different utilization

activity. In the case of solid-state storage units, namely flash storage (*e.g.,* sd-card), the power to just turn-on the unit is negligible, since there is no need for a motor. However, the speed of the bus that activates the cells for writing/reading determines the power consumed by the unit. Usually, this speed takes the value of the CPU frequency of the embedded system, such as modern smartphones. As a result, the power consumption of the storage unit will be a function of the data rate and the system frequency as expressed in the following equation:

$$P_{su} = \gamma \times R_{su} \times f, \tag{5.6}$$

where P_{su} is the average power consumed by the storage unit, R is the data rate of the writing/reading activities, and f is the system frequency. A base cost is absent in the equation because the amount of power to activate the unit is negligible.

5.3.3 Application Profile

In this subsection, we want to deploy an application profile model. The aim of application profiling is to model the usage of the hardware components by an application. We consider the energy aspects of the application that help application developers to estimate the total energy consumed by the application on a particular device. In this work, we profile the hardware by taking into consideration the CPU and the storage unit components. For applications that use networking interfaces, we refer the reader to our work on network profiling [9]. Therefore, the focus of this work is on applications that do not need networking activities.

We profile the application by studying the impact of the application on the CPU and the storage unit. For the impact on the CPU, we need to know how much the application could consume CPU time, which reflects on the CPU utilization and consequently the power consumption expressed in Eq. (5.5). Moreover, the expected execution time is important to calculate the total energy of the application. The expected execution time is calculated as:

$$T = I \times cc \times \frac{1}{f}, \tag{5.7}$$

where T is the total execution time, I is the total number of instructions of the application for a given task, cc is the number of CPU cycles, and f is the CPU frequency [122]. The first term (I) is program dependent, which is a constant for a given application doing a specific task. Similarly, the second term (cc) is CPU architecture dependent, which is constant for a particular CPU. The third term ($\frac{1}{f}$) implies that the execution time depends on the CPU frequency in a reciprocal relationship. It is obvious that increasing the CPU frequency reduces the execution time.

On the other hand, the impact of an application on the storage unit can be determined by specifying the application throughput. Here, throughput is the total amount of data read from and written to the storage unit per second by the application. Therefore, the throughput is given by:

$$R^A = \frac{Data_{read}}{T} + \frac{Data_{write}}{T} \tag{5.8}$$

Substituting T with its value from Eq. (5.7), we get

$$R^A = \delta f, \tag{5.9}$$

which represents the throughput as a first degree polynomial function of the frequency, and where δ is the relationship constant.

5.4 Experimental Validation

In this section, we validate our developed profiles by conducting experiments on real smartphones and applications. We perform the experiments on two different smartphones feature with multi-core CPUs and different versions of the *OS*. The purpose of the experiments is to evaluate empirically the value of the constants appear in profiling equations. For the CPU profile, we experimentally identify the values of α_b, β_b, α_c, and β_c in Eq. (5.4), where this equation is used for both of P_{max} and P_{min} in Eq. (5.5). In the storage unit profile,

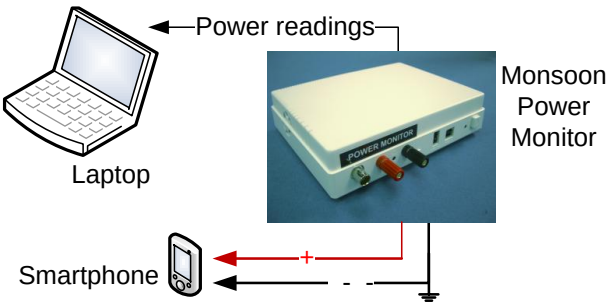


Figure 5.5: Experiments setup

we experimentally determine the value of γ in Eq. (5.6) For the application profile, we experimentally extract the value of δ in Eq. (5.9), the total time T in Eq. (5.7), and U in Eq. (5.5).

In the following subsections, we present and explain the experimental set up, the profiling procedure, and the results.

5.4.1 Experimental Setup

We set up our experiments as depicted in Fig. 5.5. In this setup, we use *Samsung Galaxy Note 3* and *Samsung Galaxy Nexus* smartphones, which features a four-core and two-core CPUs, named *Quad-core 2.3 GHz ARMv7* and *Dual-core 1.2 GHz CPU ARMv7*, respectively. In addition, these devices have *Android 4.3* with the *Kernal Version 3.4.0* and *Kernel Version 3.0.72*, respectively, as the operating system. We choose *Android OS* because it allows us to easily access many features as we describe in the following subsections; and it has the biggest market share in the smartphone industry. The power supply simultaneously powers the smartphone and records the power consumption as a time series. The power readings during the experiments are recorded on a separate laptop.

5.4.2 Experimental Results

In the experiments, we write several scripts that change the system parameter at the kernel level to profile the CPU, the storage unit, and the application. We use the predefined operating points of the system frequency, which is available in the file `scaling_available_frequencies` under the directory `/sys/devices/system/cpu/`. The scripts initialize the desired frequency in the file `scaling_setspeed` under the same directory to force the CPU

to use that frequency. Moreover, *Linux* kernel controls the number of online cores by writing in the file `online` the ID number of the core that the kernel wants to bring it online.

CPU Profile

In the CPU profiling, the scripts write the ID number of the core and the desired frequency in the kernel files as discussed above. The scripts generate load for the CPU by making its utilization 100%. At this time, the average power consumed by the device is P_{max} in Eq. (5.5). In the same way, we measure P_{min}. The scripts disable all applications in the user domain and only the *OS* is run on the CPU to minimize the CPU usage. At the same time of running the profiling scripts, the power meter records the power readings. In reality, the recorded power is not P_{min} since the *OS* was running on the CPU and consuming some of CPU time. We overcome this difficulty of directly measure P_{min} by writing scripts that record the CPU usages at the time of recording the power consumption where only the *OS* is running. After knowing the value of P_{max} and the recorded U, we can apply Eq. (5.5) to realize the value minimum power consumed by the CPU (P_{min}). These steps are repeated for each operating frequency and different number of online cores.

Figure 5.6 and 5.7 show the change in P_{min} and P_{max}, respectively, as a function of CPU frequency. Each point depicts the obtained power at the corresponding operating frequency points. The dotted lines represent the curve fitting that we expressed in Eqs. (5.10) and (5.11) as a corresponding to the models in the previous section. We obtain the value of α_b, β_b, α_c, and β_c in Eq. (5.4) by using curve fitting tools on MATLAB that use linear regression approach. We configure the fitting to the first degree polynomial. The result values are given for the models in Eqs. (5.10) and (5.11), where the obtained models have multiple R-squared value between 94-99. To reduce the number of scheduling interruption, all applications in the user domain are disabled and only the *OS* is run on the CPU. Moreover, disabling the applications allows us to control the CPU usage.

$$P_{min} = \begin{cases} 0.03f + 51, & n = 1 \\ 0.2f - 30, & n > 1 \end{cases} \tag{5.10}$$

$$P_{max} = 0.47f - 1340 + [0.175f + 1310] \times n \tag{5.11}$$

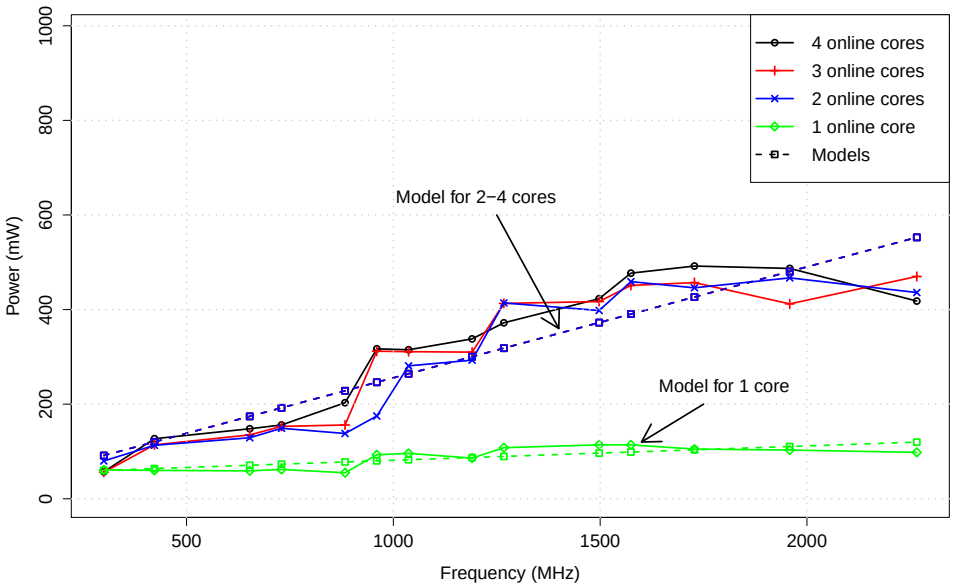


Figure 5.6: P_{min} for CPU profiling of *Samsung Galaxy Note 3*

Storage Unit Profile

In storage unit profiling, we use similar scripts but we add a workload to write into and read from the storage unit. We obtain the power consumed by the storage unit by subtracting the power that we know the CPU consumed from the total power consumed by the device. The obtained power is

$$P_{su} = 10^{-3} R f, \tag{5.12}$$

where P_{su} is the power consumed by the storage unit in mW, R is the writing/reading rate to the unit in kB, and f is the clock frequency in MHz.

Application Profile

In the application profile, we also use scripts to force the system for a specific number of online cores and working frequency. After that, another set of scripts trace the system parameters such as CPU usages, writing and read rate to the storage unit, and total execution time. These parameters are used to calculate the total energy consumed by the application using the profiling models. The CPU utilization is obtained from the kernel

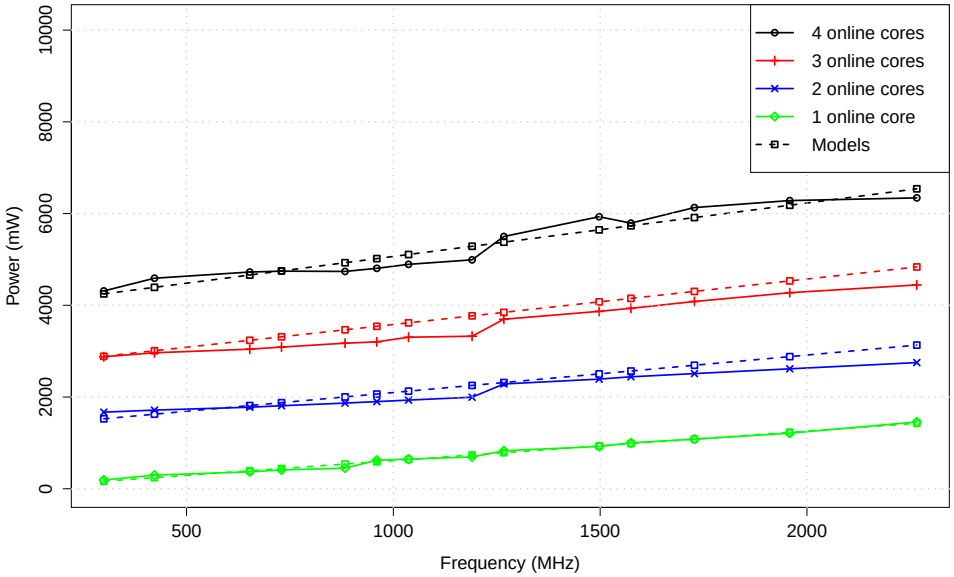


Figure 5.7: P_{max} for CPU profiling of *Samsung Galaxy Note 3*

file `/proc/stat`. The writing and the reading rate can be traced from the application log. The total execution time is retrieved from the kernel file `/proc/uptime`.

The application that we use in this case study is *FFmpeg* encoding. It was compiled for *Android* platform and installed on the device. The most important feature of this application is the supporting of multi-threading, which can utilize multi-core CPU when we test it on our device. We use it to encode a *23.97 MB flv* video file into a *20.94 MB mpeg* video file. The same input file and encoding parameters are used for consistency in our experiments. The experiments show that the encoding application utilizes the CPU at 100%, 70%, 50%, and 40% for 1,2,3, and 4 online cores, respectively. Moreover, the application throughput is shown in Fig. 5.8, the dotted lines represent the fitting curves that we expressed in Eq. (5.13). However, the linear fitting does not perfectly match the actual measurement lines because there is a small curvature on the measurement lines in the cases of more than one online core. The curvature increases as the number of online cores increases. For better results, we correct this by adding second-degree terms to give a better match as expressed in Eq. (5.13).

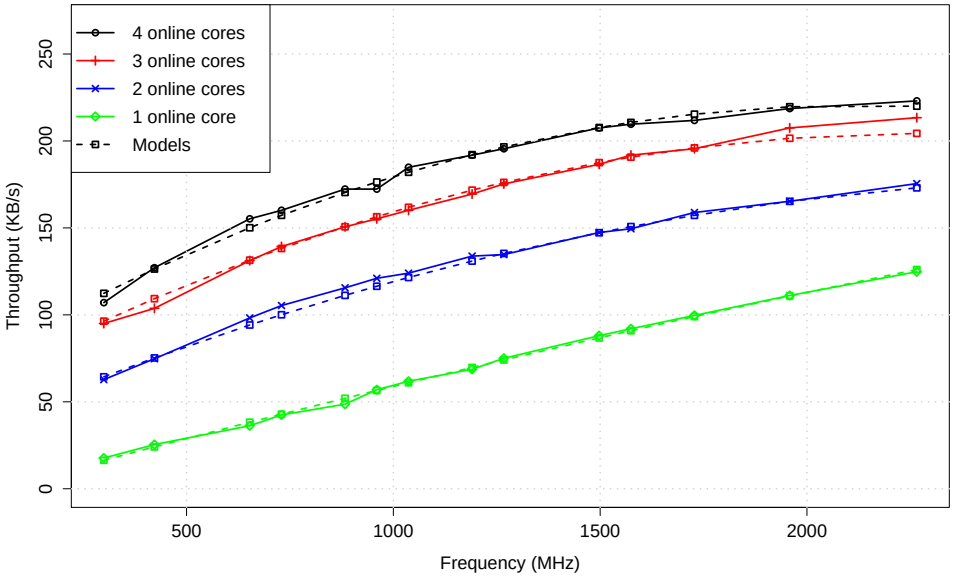


Figure 5.8: *FFmpeg* application profiling (KB/s: Kilo Bytes per second)

$$\begin{aligned} R^A = {} & 3 \times 10^{-6} f^2 + 48 \times 10^{-3} f - 22 \\ & + \left[-7 \times 10^{-6} f^2 + 18 \times 10^{-3} f + 19 \right] \times n \end{aligned} \tag{5.13}$$

5.4.3 Applicability Validation

When we developed our profiling, we kept in mind that the best profiling method is the one that is device independent and can be performed on a large number of devices with different versions of *OS*. Therefore, we aimed at making our profiling applicable to all Android devices. We conducted another set of experiments on another *Android* device and another version of *Android*, named *Samsung Galaxy Nexus*, and obtain the results on the same way we used with *Samsung Galaxy Note 3*. We used similar scripts that we used before for *Samsung Galaxy Note 3*, but we took into account the hardware configuration range such as the number of cores and the clock frequency range. The number of predefined operating points for this device is four points, namely, 350, 700, 920, and 1200 MHz.

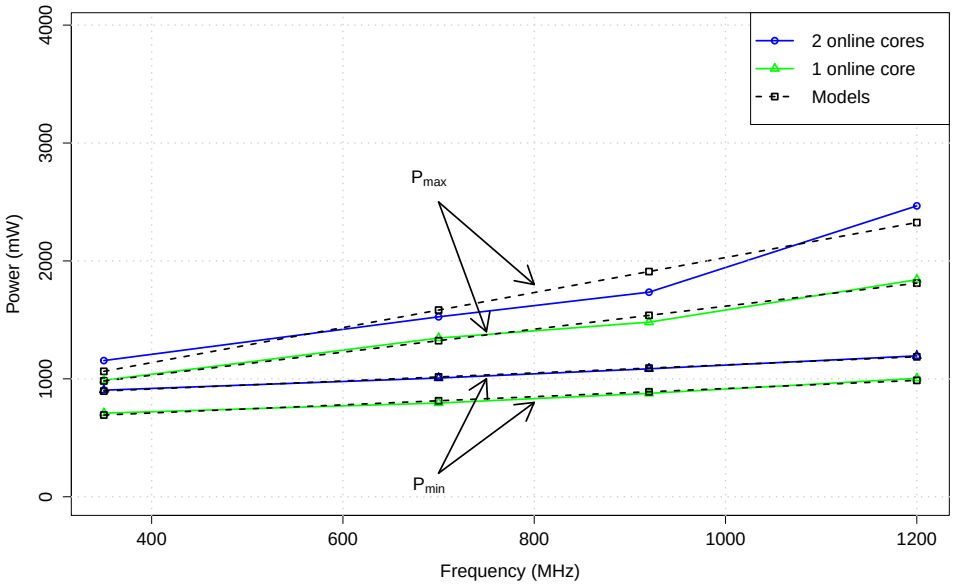


Figure 5.9: P_{min} and P_{max} for CPU profiling of *Samsung Galaxy Nexus*

The first set of experiments is conducted to profile the idle power P_{min} as depicted in Fig. 5.9. The models for the lines in Fig. 5.9 are expressed in Eq. (5.14).

$$P_{min} = 0.35f + 1921 + \left[4 \times 10^{-3} f + 205\right] \times n \tag{5.14}$$

Similarly, the second set of experiments are to profile the maximum CPU power P_{max}. The results are presented in Fig. 5.9 and their models are expressed in Eq. (5.15).

$$P_{max} = 0.462f + 736 + [0.5f - 96] \times n \tag{5.15}$$

The storage unit for this device is profiled using storage unit profiling scripts. The obtained expression for the power consumption of the storage unit is given in Eq. (5.16).

$$P_{su} = 0.4 \times 10^{-3} Rf, \tag{5.16}$$

To profile the application, we used the application profiling scripts that we used before, but again take into account the hardware configurations. The application throughput on

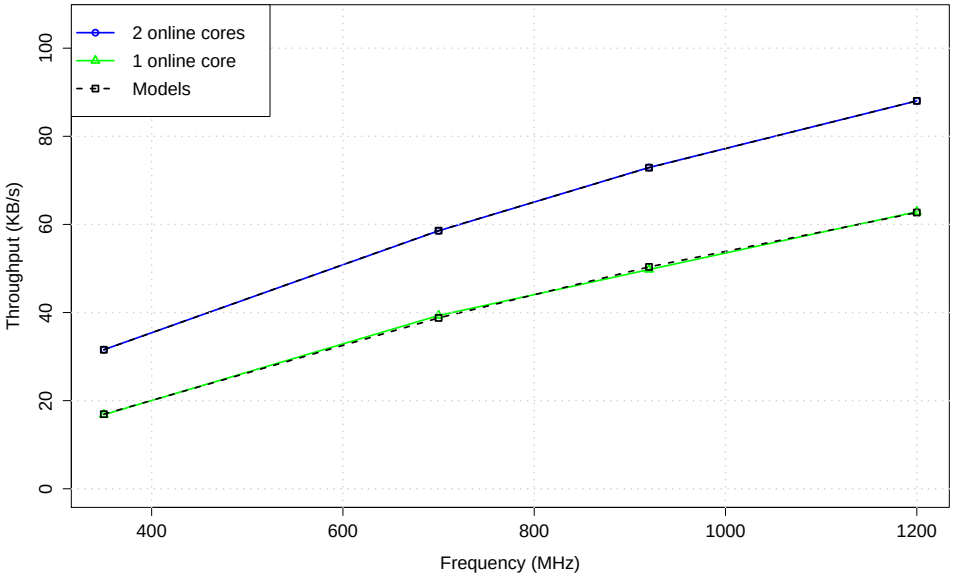


Figure 5.10: *FFmpeg* application profiling on *Samsung Galaxy Nexus*

this device for different number of online cores are shown in Fig. 5.10, and their fitting models are expressed in Eq. (5.17).

$$\begin{aligned} R^A = & -13 \times 10^{-6} f^2 + 60f - 17 \\ & + \left[-4 \times 10^{-6} f^2 + 20f + 8.3\right] \times n \end{aligned} \tag{5.17}$$

5.4.4 Application Total Energy

In this subsection, we show how the developers use the profiles to calculate the total energy consumed by a smartphone for an application. The total application energy is calculated by multiplying the total power consumption (*i.e.*, CPU and storage unit) by the total execution time. Figure 5.11 shows the comparison between the energy consumption obtained from the experiments (the solid lines) and from the profile models (dotted lines). This figure further corroborates the fact that for a fixed task the total energy is constant regardless of the CPU frequency [122]. This is because the execution time decreases as the

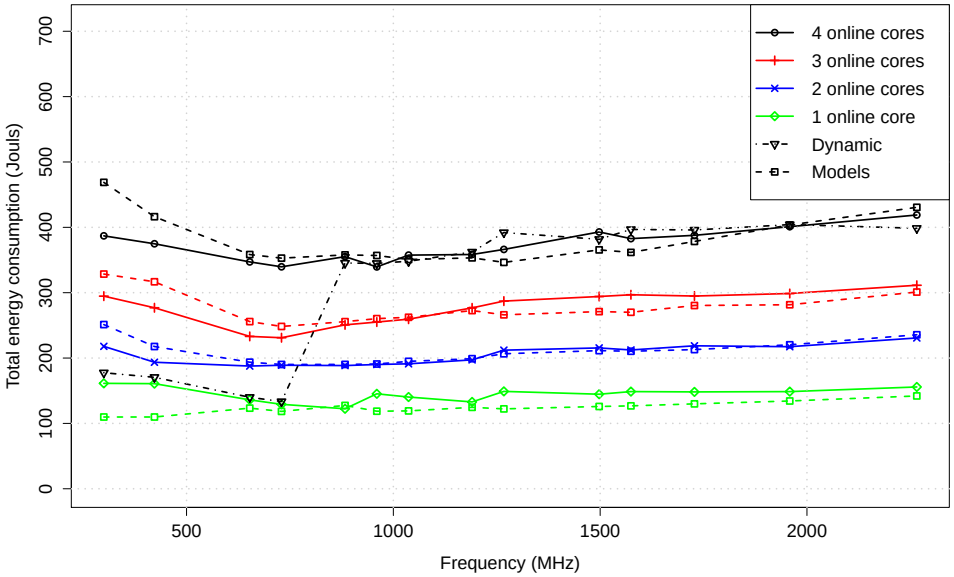


Figure 5.11: Total application energy consumption on *Samsung Galaxy Note 3*

frequency increases as expressed in Eq. (5.7). The difference between the energy levels of the online cores is due to the energy consumed to activate the cores, similar to the difference between P_{min} and P_{max} for different online cores. Figure 5.12 depicts the statistical total energy consumed for a fixed task execution in a multi-core CPU.

We conducted another set of similar experiments, but we let the *OS* choose the required number of online cores. The experimental results from this set is plotted in Fig. 5.11 for the comparison with forced online core selections (*i.e.,* 1,2,3, and 4 cores). The line for the dynamic core selection clearly illustrates that after 960 MHz the *OS* turns-on the four cores while it activates just one core otherwise. Figure 5.13 shows obtained from the experiments conducted on *Samsung Galaxy Nexus*.

One can notice from Fig. 5.11 that one core is the best option to execute any task. This observation scarifies the benefit of using multi-core CPUs. However, another important factor we should consider is the total execution time for the task. Figure 5.14 shows the total execution time for the same encoding task versus clock frequency of the CPU. This figure reveals the benefit of using multi-core CPUs since more cores mean less time is needed to finish the task. In the case of one core at low clock frequency, the execution time is very long because most of the CPU time is for the *OS* at these system configurations.

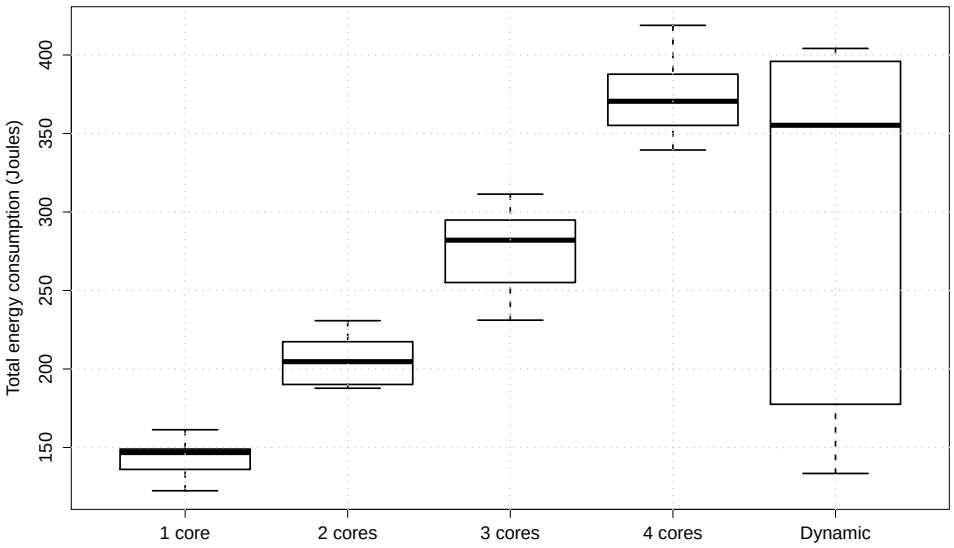


Figure 5.12: Total application energy consumption statistics

We notice in this figure that the execution time has a reciprocal relationship to the clock frequency as expressed in Eq. (5.7).

5.5 Summary and Discussion

The energy aspects of the mobile devices in many cases have not been addressed well. The problem is that the developers have no clue about the energy consumption of their application on a particular device. In this chapter, we develop hardware and software profiling models and procedures. These profiles empower mobile developers to estimate the energy consumption of an application on a given device. Moreover, we described how to profile modern multi-core CPUs, storage units, and applications under the *Android* platform. The results of the real experiments on two different smartphones reveal the validation and accuracy of our profiling procedures.

However, the *OS* overhead on the modelling is experimentally unavoidable. Therefore, we keep the *OS* at the lowest operating point where we disable unnecessary *OS* services. In this case, the changing on the usage for the hardware components is caused by the test

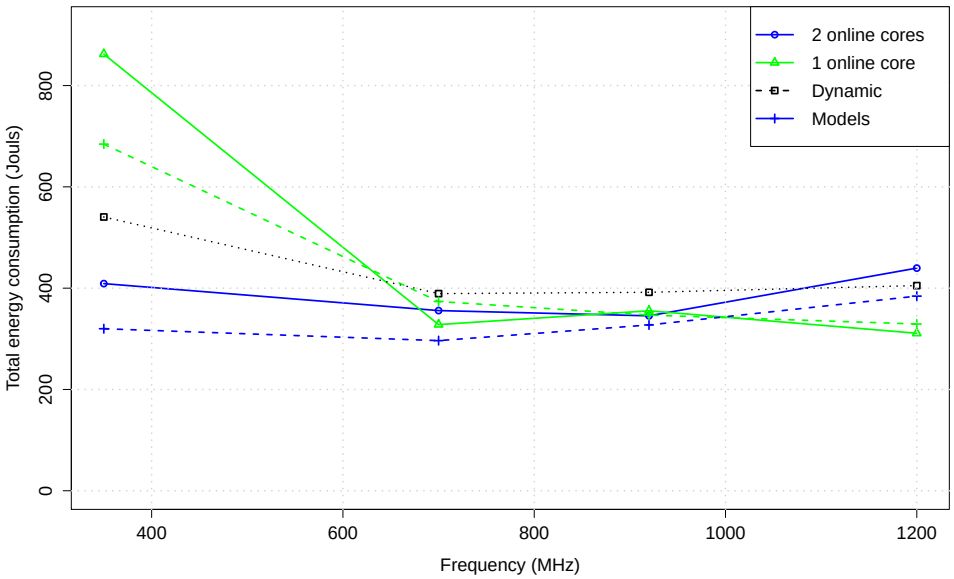


Figure 5.13: Total application energy consumption on *Samsung Galaxy Nexus*

application. As a result, we can model the application based on this change. Another way to over come this overhead is by assuming the *OS* as separate application. Because the energy consumption of the applications is additive, as we modelled in Subsection 1.2.2, we substitute the energy of the *OS* for measuring the energy of the test application.

In modelling the computing activities, we ignore the energy consumption of the memory because it is relatively small compared with other components. But, that could have impact on the system for applications that need long execution time. On the other hand, we developed the model for each hardware component individually but in some cases the dependency between components cannot be ignored, which depends on the characteristics of the application. Additionally, the behaviour of the system could be different when more than one application simultaneously running on the shared CPU affected by of the *OS* scheduler. This impact the total execution time but not the energy required for an application because we modelled the energy based on the used CPU timing (*i.e.,* CPU load) for the application.

In the application profiling, we avoid the complexity of application branching and we model the application for one task. To extend our models and make it applicable to multi-task application, the application can be divided into tasks where each task is modelled using

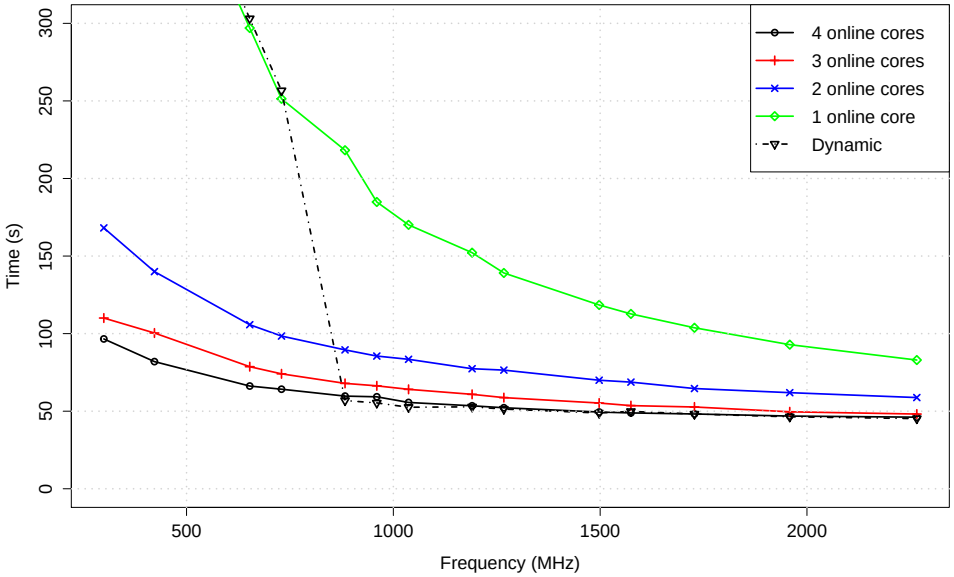


Figure 5.14: Total execution time for encoding a video file

similar approach that we used. Thus, We emphasize the energy model for application that we developed in 1.2.2 is the starting point for any energy modelling of an application.

Chapter 6

The Proposed Offloading Framework

Task offloading from the mobile device to the cloud is essential for enhancing their computing capabilities, and saving their battery energy at the same time. However, task offloading introduces a new challenge due to the impact of system parameters on the offloading performance. This challenge is to make an offloading decision correctly before executing the offloading processes. An accurate offloading framework allows the mobile devices to make the correct decision as to whether or not to perform the task offloading based on the system parameters at the time of making the decision.

In this chapter, we tackle this challenge by developing an offloading framework that gives mobile devices the ability to make the correct offloading decision. The framework accomplishes the correct decision after it collects the required system parameters that have a direct influence on the offloading decision. We consider all sources of system parameters and classify them as profiles to build the framework based on these profiles. We implemented and validated these profiles experimentally on real-life scenarios for a real smartphone, a smartphone application, and cloud. The experimental results show that our proposed framework is practical to accomplish the appropriate offloading decisions. The contributions of this chapter are published in [11] and [12].

6.1 Preamble

Task offloading is a promising solution to overcome many device limitations, especially the energy limitation [4]. As current mobile devices feature Internet connectivity with fast wireless networks, reaching remote computing resources is practical. In the era of cloud

computing, remote computing resources are accessible anywhere and anytime. Cloud computing provides its users with virtually unlimited computing resources from its data centres [5]. Based on these observations, a mobile device is able to offload specific computing tasks to the cloud for remote task execution on the cloud and receive the results with less energy consumption than executing that task on the device itself [6, 7].

In our previous work that presented in Chapter 3 and published in [8], we examined the feasibility of offloading from smartphones to cloud, where we focused on energy as the performance metric. We conducted a set of experiments for four real-life scenarios and evaluated the energy cost on smartphones in the case of using the offloading technique over different network interfaces and Internet protocols. The results reveal that offloading is beneficial in most scenarios as presented in the result figures shown in Chapter 3.

However, offloading is a critical technique that depends on the system parameters of both the mobile device and cloud. For example, offloading is not beneficial if the mobile device is connected to the Internet using a 3G network interface and the HTTP Internet protocol, as shown with our experimental results in Chapter 3. Therefore, we realized that to make offloading beneficial for mobile devices, an offloading decision is vital. Consequently, we developed an offloading decision engine on a real smartphone and cloud that performs the offloading decision as to whether or not the mobile device will save energy by offloading a given task [11].

The aim of this chapter is to design an offloading framework that is suitable for implementation in mobile devices and applicable to the environment of the cloud computing, which provides the offloading capability to mobile devices. In this framework, we take into account all of the system parameters that affect on offloading decision. We consider all sources of system parameters and classify them as profiles, and we build the framework based on these profiles. These profiles let the proposed framework accurately make the correct offloading decision to save energy on mobile devices. This chapter makes the following contributions:

- developing a general offloading framework to offload tasks from mobile devices to cloud computing;
- classifying the system parameters based on their sources to build accurate framework;
- developing an offloading procedure for making the offloading decision using these profiles; and
- validating experimentally the proposed framework on real device, application, and cloud.

The remaining sections of the chapter are organized as follows. Section 6.2 represents the system parameters and the formulation for the proposed offloading framework. The description of the proposed offloading framework is presented in Section 6.3. We illustrate the implementation of our proposed offloading framework in Section 6.4. This chapter is concluded in Section 6.5.

6.2 System Parameters

Offloading a task from a mobile device that requires sending the task load (*i.e.,* information, specifications, and data) to a powerful computing machine – in our case the cloud – for executing the task remotely, and then receiving the results of the remote execution. The offloading decision for a given task is affected by the system parameters. We identify these parameters based on the need for actually making an offloading decision. In this section, we describe these parameters, and in the next section we show how these are used to make an offloading decision.

Simply put, an offloading decision is made based on the result of a comparison between the estimated energy cost for task offloading and the estimated energy cost for local execution of the task [11]. Estimating the energy needs to perform some calculations using system parameters, where these parameters come into play. In this framework, we consider and describe these parameters based on the following eight sources:

1. user,
2. application,
3. content,
4. hardware,
5. battery,
6. network,
7. location, and
8. cloud.

We classify the parameters from each source in a separate profile to build a general and expandable offloading framework. In the following, we explain these profiles and their parameters.

6.2.1 User Profile

The user profile includes the parameters that are user dependent. The user of the mobile device controls and manages the parameters in this profile. In fact, mobile device users configure their devices for a set of configurations suitable for their needs. For example, the user could enable or disable the networking interfaces according to the needs and their location. In the context of task offloading, we limit the user profile to two parameters. The first parameter describes the status of networking, which is either enabled or disabled. We represent this parameter by Networking Status (NS) binary variable. The second parameter (T_u) represents the maximum time that the user allows for the offloading [123].

6.2.2 Application Profile

In this profile, we include the parameters that are application dependent. These parameters are keys for computing the total energy consumed by the application, if the task executed on the mobile device. Assume that a mobile application A consumes energy equal to E^A and provides the user with some sort of quality of service denoted as QoS^A. To compute E^A, we need to know how the application interacts with the hardware components since the energy consumption is due to the application's usage of some hardware components. For instance, A utilizes the CPU with amount of U^A (*e.g.,* 40%), and writes into and reads from the storage unit with a rate equal to R^A. We identify the QoS in the application profile to let our offloading framework having the ability to do offloading without sacrificing the quality of the task to saving energy. Ideally, the developer of an application knows the application profile at the implementation time. In this case, we assume that the application profile parameters are already included in the application package.

6.2.3 Content Profile

This profile contains the parameters that describe the task content affects the energy consumption of the task. In fact, with respect to the energy consumption, the only parameter we need in this profile is the size of the content. We assume the user wants to process a content file of F bytes using A. Based on the value of F, the offloading framework calculates the energy consumption for this task in case of execute it locally on the device or remotely on the cloud. The content profile can be known easily by reading the metadata of the content file.

6.2.4 Hardware Profile

In this profile, we show the parameters that indicate how the hardware components consume energy as these interact with an application. Assume that the mobile device consists of hardware components and each component consumes a specific amount of power P_c to serve mobile applications. For instance, the CPU as a hardware component consumes some power (P_{CPU}) to process the computation task given by an application. Similarly, if the mobile device sends or receives data over the wireless interface, then the wireless components consume some power for this task. The energy consumed E_c by a component c is given by

$$E_c = T_c \times P_c \tag{6.1}$$

where T_c is the period of time when the component is in use, and P_c is the power consumption of the component. The value T_c is application and content dependent as we explain later in this chapter. The value of P_c usually is constant but sometimes it depends on other hardware configurations. For example, the power consumed by the CPU, P_{CPU}, is a function of the CPU frequency [10]. The manufacture and device maker usually precisely specify the hardware profile of each of component in the device they produce.

The total energy consumed for an application A is the sum of the energy consumed by each component used by this application, which is calculated as

$$E^A = \sum_{C^A} E_c \tag{6.2}$$

where C^A is the set of hardware components that A uses.

6.2.5 Network Profile

In this profile, we explain the parameters that are network dependent. Each wireless network has its own purpose and characteristics; therefore, different types of wireless networks implement different communication protocols and radio interfaces. As a result, during wireless communication, the energy consumption of the mobile device is highly influenced by the chosen type of the wireless network, see Chapter 4 and [9]. The energy consumed in networking E_{NT} is equal to the active time of the wireless component (T_{NT}) multiplied by the power consumption of the wireless component (P_{NT}) as given in Eq. (6.3).

$$E_{NT} = T_{NT} \times P_{NT} \tag{6.3}$$

For the wireless component, the active time is the time required to transfer a certain amount of data at the rate of R bytes per second. In the offloading technique, the transferred data is the amount of data (*i.e.,* F bytes) to be offloaded. Since different wireless interfaces consume different level of power while supporting different data rates, considering only the power consumption or the supported data rate is not enough to have an insight into the energy cost of the communication modules. We introduce a metric that helps tackle this issue called the energy efficiency by a communication module e in bytes per Joule; e is equal to the supported data rate over the power consumed of a communication module.

The energy consumption for a communication module given the amount of data and the supported data rate is given as in Eq. (6.4).

$$\begin{aligned} E_{NT} &= T_{NT} \times P_{NT} \\ &= \frac{F}{R} \times P_{NT} \\ &= \frac{F}{e} \end{aligned} \tag{6.4}$$

6.2.6 Battery Profile

This profile is developed to allow our offloading framework to consider the capacity of the battery and ensure the completion of the offloading process in case the offloading engine decides to offload the task. We denote the remaining energy on the device battery as E^B at any time instant. If the offloading energy is estimated to be E^{off}, then the remaining energy on the battery should be greater than or equal to E^{off} to make sure that the device is able to complete the offloading. This constraint is presented as

$$E^{off} \leq E^B. \tag{6.5}$$

It is worth noting that the battery profile is already developed on all current mobile devices and can be obtained from the operating system.

6.2.7 Location Profile

We consider the location profile to determine the available network infrastructure and the reachable cloud from the current location. One example of the need for such a profile is that many Internet providers limit their coverage to a specific geographical region. Moreover,

even though the cloud is accessible worldwide, some countries restrict the access to limited destinations. As a result, task offloading to the cloud has to consider this limitation. The parameters of this profile are binary in nature. For example, the network parameter is $NT_i = 1$, if the network i is available and $NT_i = 0$, otherwise. Similarly, the cloud parameter is $CC_j = 1$ and $CC_j = 0$ for an available and unavailable cloud, respectively, from the current location of the mobile device. The parameters of this profile are:

$$NT_i = \begin{cases} 1 & \text{if the network } i \text{ is available} \\ 0 & \text{if the network } i \text{ is unavailable} \end{cases} \tag{6.6}$$

$$CC_j = \begin{cases} 1 & \text{if the cloud } j \text{ is available} \\ 0 & \text{if the cloud } j \text{ is unavailable} \end{cases} \tag{6.7}$$

The location profile can be obtained by sending probe packets to all known clouds from the current location through any available network such as WLAN or cellular data. Examining the network interfaces at the OS level can retrieve the availability of the networks.

6.2.8 Cloud Computing Profile

In this profile, we show the parameters that describe a cloud with respect to task offloading. The cloud profile consists of the status of the cloud and the supported Quality of Service (QoS) at current time. The status in our case is the queue of processing requests. The waiting time of the offloading request in the cloud queue T^{cc} must not exceed offloading threshold T^{off}. Moreover, the supported cloud QoS (QoS^{cc}) should be greater than or equal to the minimum QoS required by the application (QoS^A).

$$T^{cc} \leq T^{off} \tag{6.8}$$

$$QoS^{cc} \geq QoS^A \tag{6.9}$$

The cloud profile is known by including the cloud status in the response of the location probe.

6.2.9 Profiles Summary

Finally, we summarize the profiles as in Table 6.1, which lists all notations and symbols used in our profiles formulation.

Table 6.1: List of symbols used in profiles formulation

	Profile	**Parameter**	
		Symbol	**Description**
1	User	NS	Network Status
		T_u	User maximum time for the offloading to finish
2	Application	E^A	Total energy consumed by the application A
		QoS^A	Quality of service provided by the application A
		U^A	CPU utilization due to app. A
		R^A	Application throughput
3	Content	F	The size of the task content
4	Hardware	E_c	Total energy consumed by a component c
		T_c	Total time of using a component c
		P_c	Power consumed in a hardware component c
5	Network	E_{NT}	Total energy consumed in networking
		T_{NT}	Total networking time
		P_{NT}	Average power consumption in networking
		R	A Network data rate
		e	The energy efficiency of a communication module
5	Battery	E^{off}	Total offloading energy
		E^B	Remaining energy on the device battery
7	Location	NT_i	The availability binary variable for the network i
		CC_j	The availability binary variable for the cloud j
8	Cloud	T^{cc}	Total time of the task in the cloud
		T^{off}	Offloading threshold time
		$QoS^c c$	Cloud application supported Quality of Service

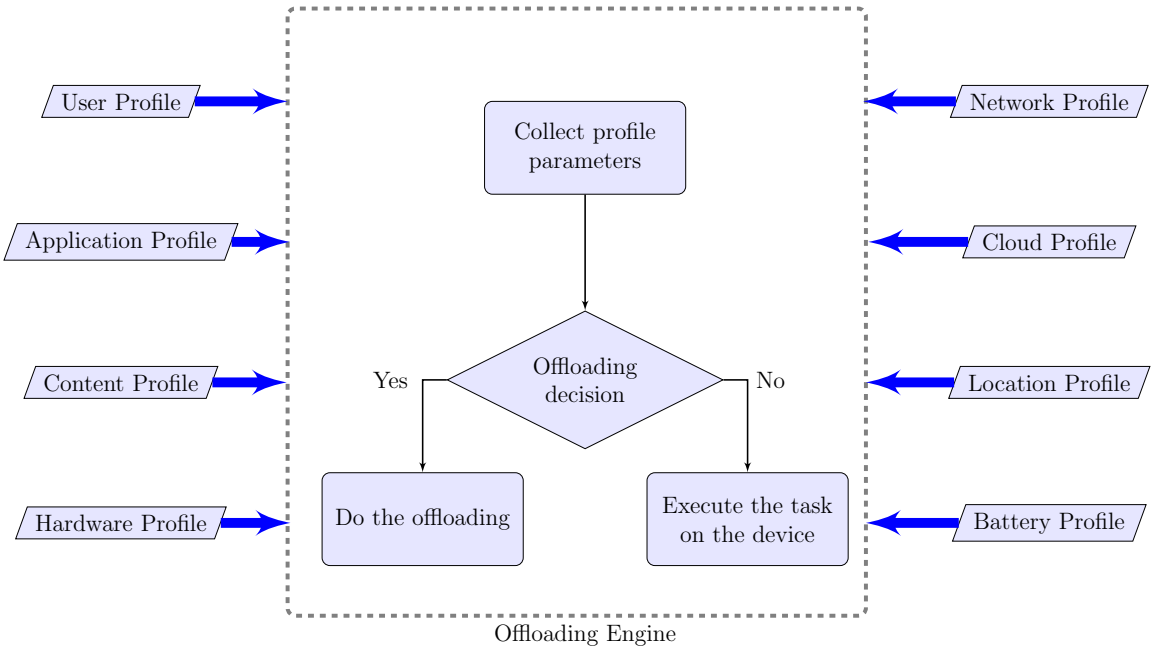


Figure 6.1: Proposed offloading framework

6.3 Offloading Framework and Decision Procedure

In the following, we describe how previous profiles are integrated together to build our proposed offloading framework, as depicted in Fig. 6.1. It shows how the profiles are used to make an offloading decision.

In this section, we describe how the proposed framework works. The offloading framework performs the following procedure, which is represented as a flow chart in Fig. 6.2. The step numbers in the following procedure correspond to the box numbers in the decision flow chart.

1. When an application A is started and the user wants to perform some tasks, the offloading process is started immediately.

2. The engine calls for the user profile to check if any wireless network is enabled. The user profile contains user configurations, which is stored on the smartphone system files. Sometimes, the user disables networking feature, such as roaming, to avoid

extra charge. If the networking feature is disabled, then there is no point in going further in the process so end the offloading process (*i.e.,* Case 1).

3. If the networking is enabled, the engine calls the location profile to determine the available networks and online clouds. This calling step includes paging for all enabled networks and pinging all known clouds. The results are built into sets NT and CC.

4. Based on the CC set, the engine finds the set CC^A, which is the cloud that has a clone for A. At the same time, the selected cloud should provide QoS^{cc} similar to or better than what the smartphone application could provide. The value of CC^A, which takes on value 0 or 1, and the value of QoS^{cc} can be included in the response to the request for cloud profiles. A request is sent to all clouds in CC.

5. If the set CC^A is empty, the offloading process stops trying to offload and ends the procedure (*i.e.,* Case 2).

6. If there is at least one cloud that has a clone for A, then select the network that provides lower energy consumption per byte e.

7. Now, the engine calculates the expected energy E^{off} needed to offload the task, and the expected time to complete it.

8. An important step in the offloading engine is to check if the battery has enough energy to complete the offloading. This step is accomplished by evaluating the condition $E^{off} \leq E^B$. Moreover, the time to complete the offloading should not exceed the time limit T_u defined by the user, where $T^{off} = T^{NT} + T^{CC}$. Otherwise, the engine terminates the decision procedure (*i.e.,* Case 3).

9. If offloading can be done by using the remaining battery capacity and within the time limit, the last step is to calculate the total energy consumed E^A, if the task is performed by the local application.

10. Based on this energy, a decision can be made as to whether the offloading will be beneficial to the smartphone or not. Calculate the gain G as the difference between the energy consumed for the application and for the offloading, which is given as

$$G = E^A - E^{off}. \tag{6.10}$$

11. The final decision is made based on gain G; if the gain is positive, the offloading will be beneficial to the mobile device, when the offloading starts (*i.e.,* Case 5 and Case 4 if G is negative).

12. In any case where the decision procedure is terminated, the engine lets the device execute the task locally.

6.4 Proof-of-Concept Implementation

In this section, we present the applicability of our proposed framework to real life scenarios. The state of the art in the framework implementation is presented in subsection 6.4.1. Then, we show a case study for the implementation of our proposed offloading framework in a real life scenario.

6.4.1 Practical Implementation

In this subsection, we discuss and explain the practical methods to implement our framework and the state of the art in the implementation. We address the implementation of the boxes 2 through 10 in Fig. 6.2 and the possible decisions and computations. The summary of the state of the art is listed in Table 6.2.

We discuss the implementation of the boxes in the following corresponding points:

2. For this step, the user profile is a matrix that contains all of the user current configurations, such as screen brightness and speaker volume [20]. This profile parameters can be specified easily using an approach similar to the one described by Shye et al. [124].

3. The location profile can be obtained using a service discovery protocol to all known clouds from the current location through any available network, such as WLAN or cellular. In fact, examining the available networks is already implemented in the operating systems of modern mobile devices and it can be retrieved via system calls [125]. Moreover, the service discovery protocol has a response from the cloud that is reachable from the current mobile location [126, 127]. Based on the probe responses, the offloading engine builds the sets CC and NT, which contain the reachable clouds and available networks, respectively.

4. In this step, the characteristics of the application can be known in the application design and implementation stages. Application developers determine the characteristics of their application with the Software Development Kits (SDK) [128, 129]. In

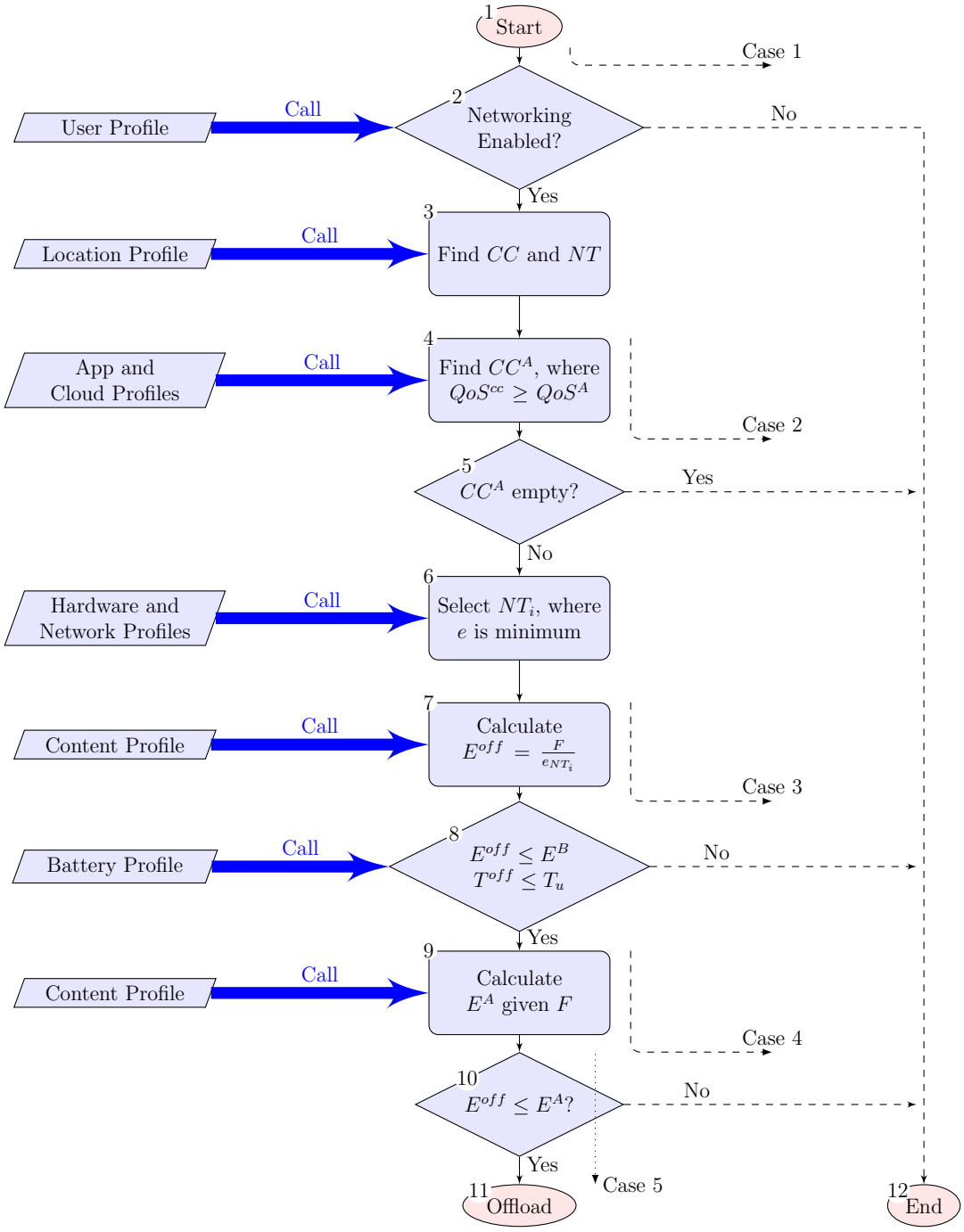


Figure 6.2: Offloading decision flowchart

this case, we assume that the application profile parameters are already included in the application package. On the other hand, the cloud profile is known by including the cloud status in the response of the location probe packets that has been performed in the previous step. Giving the QoS provided the application and the cloud, the engine constructs the set CC^A that contains all clouds that provide QoS^{cc} equal to or greater than the QoS^A of the application.

5. This step examines whether the result CC^A is empty or not.

6. In this step, we assume that the makers for each hardware component specify power consumption of their components, which is usually provided in the components' datasheets. Otherwise, the literature shows a tremendous amount of methods to experimentally profile the power consumption of the hardware components [8, 9]. Therefore, we assume that these parameters are stored in a specific file to be used our offloading engine.

7. The offloading engine calculates the expected energy (E^{off}) consumed to offload the given task, given the task parameters from the content profile. For instance, the content profile used in this step is the size of the data (F) that has to be transferred during the offloading, which can be obtained from the meta-data of the task file.

8. The battery profile has been already developed on all current mobile devices. For instance, the remaining energy on the battery (E^B) is accessible by the operating system [130, 131]. As a result, this profile can be easily determined. In this step, the offloading engine only compares the obtained battery energy with the calculated offloading energy from the previous step.

9. Several approaches have been presented in [9, 132, 133, 117], and one of them can be adopted to calculate the energy (E^A) consumed by the mobile components to run a given application (A).

10. In this step, the offloading engine examines whether or not the offloading is beneficial to the mobile device. The offloading is only beneficial if the energy consumed to offload the given task is less than or equal to the energy consumed to execute the given task locally.

Table 6.2: Boxes implementation

Box number	State of the art
2	Use the profiling methods [20, 124]
3	Use service discovery protocol to build the sets CC and NT [125, 126, 127]
4	The provided QoS is included in the application package [128, 129]
5	A comparison
6	Use the models developed by [8]
7	Use the models developed by [9]
8	Use battery profiling that already developed on the mobile device [130, 131]
9	Estimate application consumed energy by [132, 133, 117]
10	A comparison

6.4.2 A Case Study

In this subsection, we present a real life case study of the implementation of our proposed framework. We implement and validate it experimentally with the help of previous Chapters (Ch. 4 and Ch. 5). We describe in detail the experimental setup and the profile's implementation procedure. Moreover, we show how the offloading decision is performed experimentally.

Experimental Setup

The aim of the experiments is to retrieve the profiles for a smartphone for this case study. In this case study, we profile an encoding application, a video file as the task content, hardware components of a smartphone, and several wireless networks. We set up our experiments as depicted in Figs. 4.2 and 5.5. In this setup, we use a *Samsung Galaxy Note 3* smartphone, which features a four-core CPU, called *Quad-core 2.3 GHz ARMv7*. In addition, the device is powered by *Android 4.3* with the *Kernal Version 3.4.0* as the operating system. We choose *Android OS* because it allows us to easily access many features as we describe in the following subsections; and it has the biggest market share in the smartphone industry. This device accesses the Internet through WLAN, 3G, or 4G networks. Using these networks, it can upload and download files to and from the cloud. The power supply simultaneously powers the smartphone and records the power

consumption as a time series. The power readings during the experiments are recorded on a separate laptop.

Experimental Results

The experimental results show the profile of an application, task content, hardware components of a smartphone, and wireless networks. The methods and the approaches have been extensively presented and explained in detail in the previous chapters. The summary of the obtained profiles is briefly listed in Table 6.3. The value on this table obtained from the experiments that we conducted similar to the experiments explained in the previous chapters.

6.4.3 The Offloading Decision

Without loss of generality, we assume that the parameters of the remaining profiles are known such as the user enable the networking, the battery has enough energy to offload the task, all known clouds are reachable through all wireless networks from the current location, and cloud has a clone for the application. Therefore, we omit cases 1 to 3 in this analysis. Box 10 is the key in the offloading decision in which the engine compares the energy consumed for local execution of the task and the energy for offloading the task. For a given device and a specific application, the profiling parameters are constants as we can see in Table 6.3. However, the only variables are the size of the task data (F), the operating frequency of the CPU (f), the number of online CPU cores (n), and the used wireless interface in the case of offloading the task.

Figures 6.3 – 6.4 show the computed gain in Joules from Eq. (6.10) using Eqs. (4.5) – (4.20) for WLAN, 3G, and 4G networks, and Eqs. (5.5) – (5.9) for *FFmpeg* application. *FFmpeg* is an open source package that have libraries and programs for handling multimedia data. The white area on the surface represents the operating points where the task is not to be offloaded. On the other hand, the dark area represents the operating points where an offloading will be beneficial. Points in the dark regions mean more energy saving due to the offloading. In these figures, we limit the range of F to show the critical areas in the offloading decision. The offloading engine calculates the gain at only one point given the value of F, F^{cc}, f, n, and the available network with highest energy efficiency.

For example, given a task that encodes a 25 MB *flv* video file ($F = 25\,MB$) into a 25 MB *mpeg* video ($F^{cc} = 25MB$), the operating frequency of the CPU is 300 MHz ($f = 300\,MHz$), and all 4-cores are online ($n = 4$), then the total estimated energy consumed

Table 6.3: Experimentally obtained profile parameters

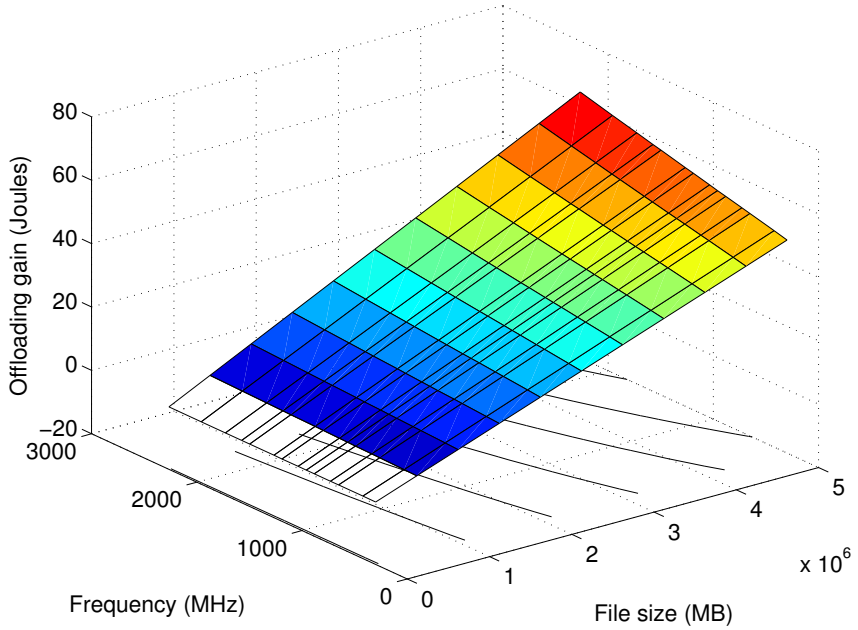
Profile	Parameter name	Parameter symbol	Value	
Hardware	CPU clock frequencies	f	(300, 422.4, 652.8, 729.6, 883.2, 960, 1036.8, 1190.4, 1267.2, 1497.6, 1574.4, 1728,1958.4, 2265.6) MHz	
	number of online cores	n	1,2,3, or 4	
	Upload power consumption	P_{up}	WLAN	1280 mW
			3G	750 mW
			4G	2300 mW
	Download power consumption	P_{down}	WLAN	1044 mW
			3G	730 mW
			4G	1250 mW
	CPU power	P_{cpu}	P_{max}	$0.47f - 1340 + [0.175f + 1310] \times n$
			P_{min}	$\begin{cases} 0.03f + 51, & n = 1 \\ 0.2f - 30, & n > 1 \end{cases}$
	Storage unit power consumption	P_{su}	$10^{-3} R^A f$	
Applications	CPU utilization	U^A	1, 0.7,0.5, and 0.4 for n=1,2,3, and 4, respectively	
	Throughput	R^A	$\begin{cases} -4 \times 10^{-6} f^2 + 66 \times 10^{-3} f - 3, & n = 1 \\ -10 \times 10^{-6} f^2 + 24 \times 10^{-3} f + 26, & n > 1 \end{cases}$	
Contents	File size	F	23 MB	
Network	Signaling energy	E_{ps}	WLAN	nun
			3G	0.56 Joules
			4G	0.45 Joules
	Tail energy consumed	E_{tl}	WLAN	nun
			3G	6.61 Joules
			4G	9.31 Joules
	Data rate	R_{up}	WLAN	6.75 MB/s
			3G	1.16 MB/s
			4G	1.03 MB/s
		R_{down}	WLAN	6.75 MB/s
			3G	2.08 MB/s
			4G	2.13 MB/s
	WLAN Download energy	E_{down}	$\left\lfloor \frac{F}{1448} \right\rfloor \{339.5P_{RX} + 63.17P_{TX}\} + \left\lceil \frac{F}{4344} \right\rceil \{34.59P_{TX}\} + 148.5P_{RX}$	
	WLAN Upload energy	E_{up}	$\left\lfloor \frac{F}{1448} \right\rfloor \{241.7P_{RX} + 309.5P_{TX}\} + \left\lceil \frac{F}{11580} \right\rceil \{183.1P_{RX}\}$	

by the application (E^A) equals 480 $Joules$ and the total estimated energy consumed to offload the task (E^{off}) over 3G network equals 18 $Joules$. As a result, the computed gain equals 462 $Joules$, which means the offloading is beneficial since it is positive.

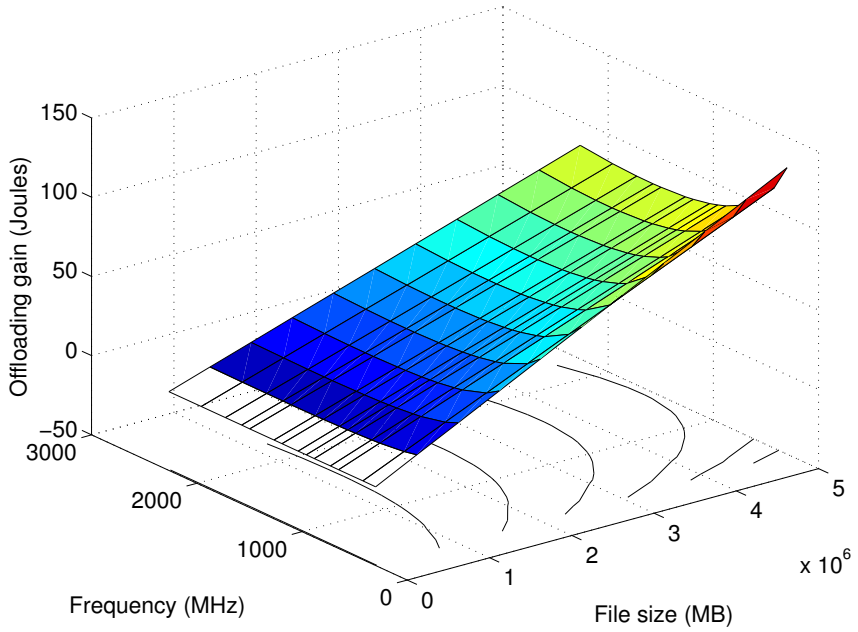
In term of battery life, the energy that the offloading saves for the mobile device equals the gained energy over the power consumption of the device. If we know that the power consumption of the device equals $35\,mW$ at the idle state, then the gain in term of battery life is given as $t_G = \frac{G}{P_{idle}} = \frac{462\,Joules}{35\,mW} = 220\,minutes$. That means the device will stay at idle state for 3.6 $hours$ longer if it offloads that task to the cloud. The overhead is negligible since we only need to compute two linear simple equations as we demonstrated above.

6.5 Summary and Discussion

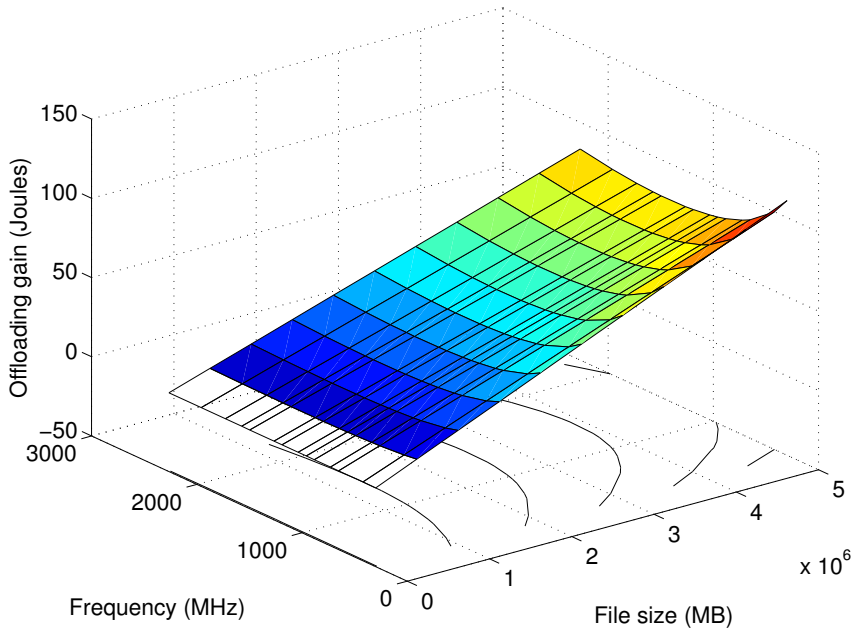
Extending the capabilities of these devices beyond their battery energy capacity is possible by offloading tasks to the cloud. However, making a correct task offloading decision is crucial to making the offloading beneficial, which happens only when the energy consumed in the offloading process is less than the energy consumed without it. Therefore, the major challenge in task offloading is to accurately make the offloading decision. In this chapter, we developed a comprehensive framework for task offloading and we introduce a set of parameters that are necessary to perform the task offloading. The experimental validation shows the performance of our proposed offloading framework on several real life scenarios. It should be emphasized that our proposed framework is realistic and applicable to all mobile cloud computing structures. We would extend our work to consider more system parameters such as cloud waiting time and security parameters.



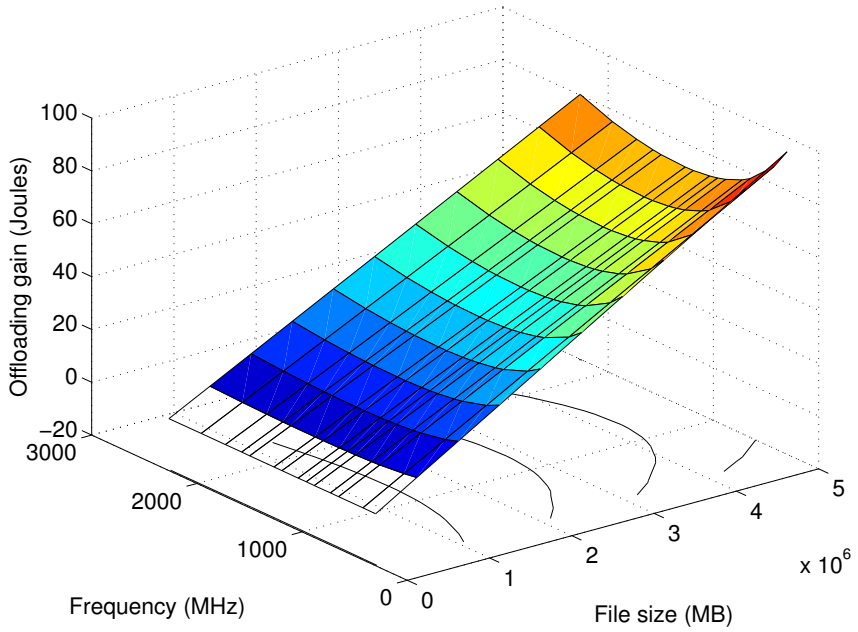
(a) 1 Core



(b) 2 Cores

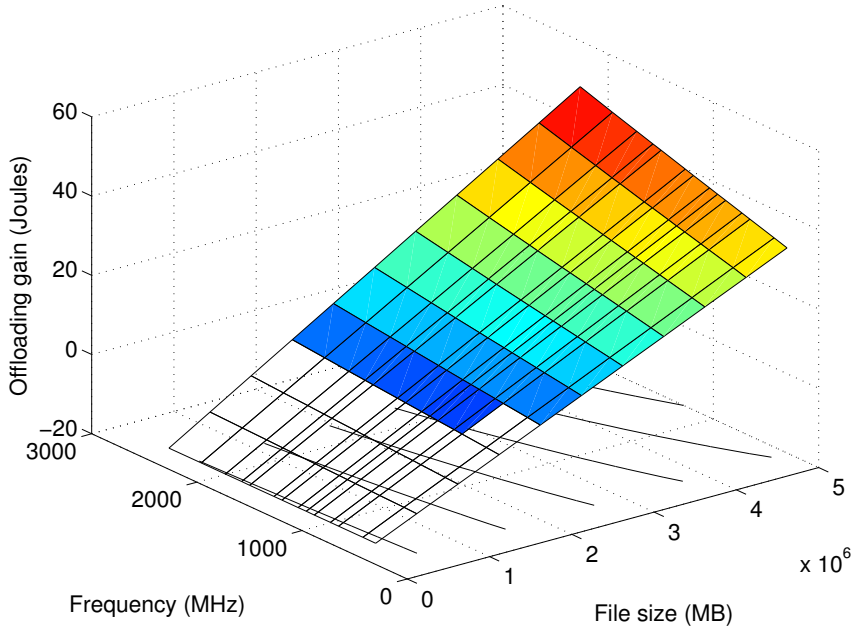


(c) 3 Cores

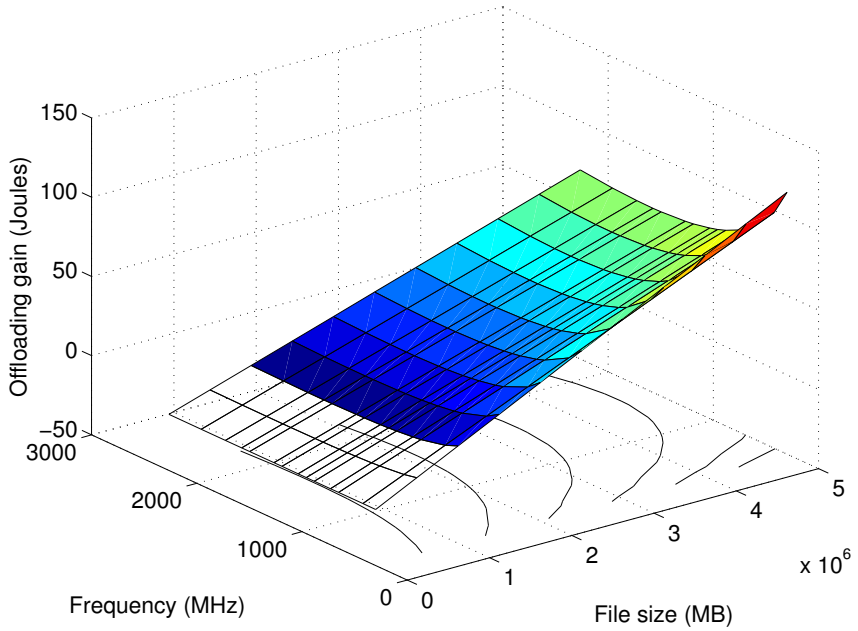


(d) 4 Cores

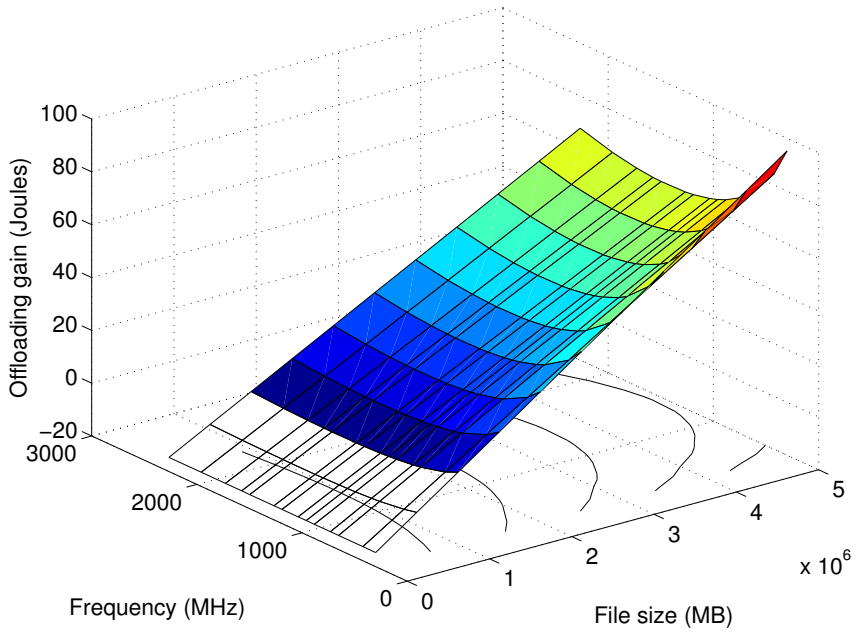
Figure 6.3: Offloading gain with WLAN interface



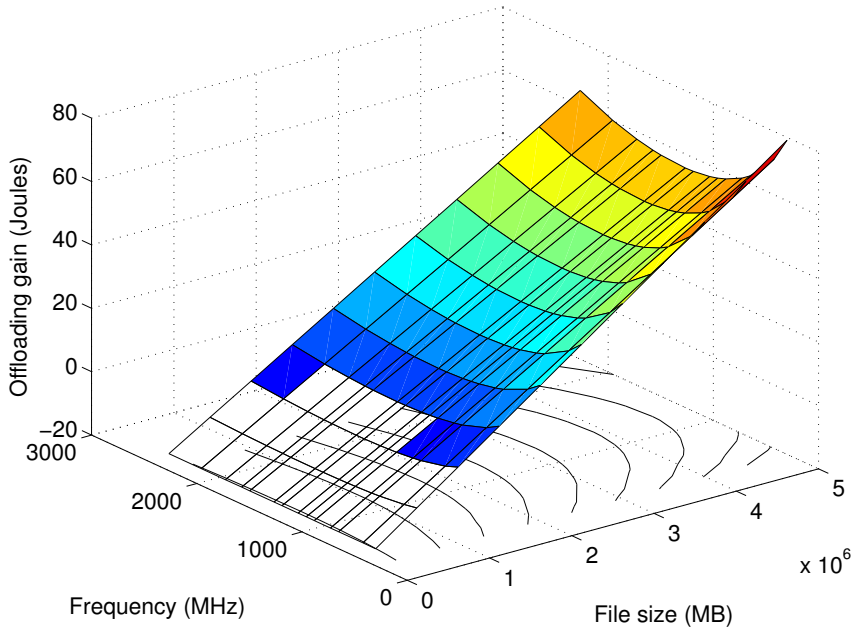
(a) 1 Core



(b) 2 Cores

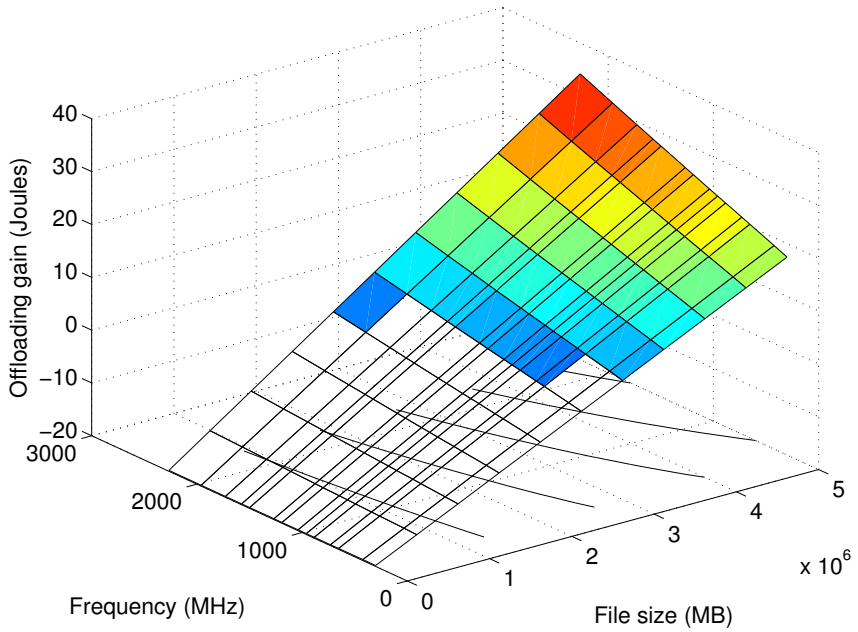


(c) 3 Cores

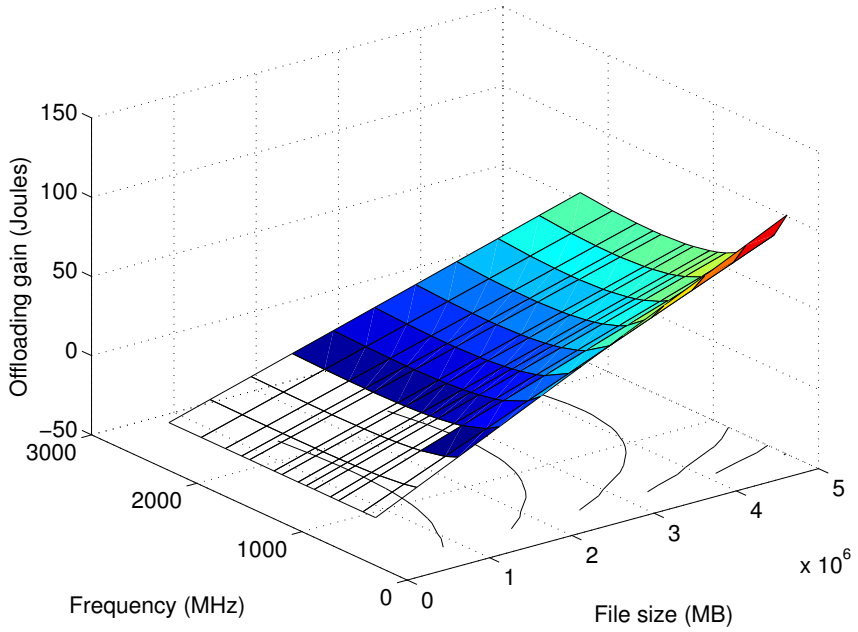


(d) 4 Cores

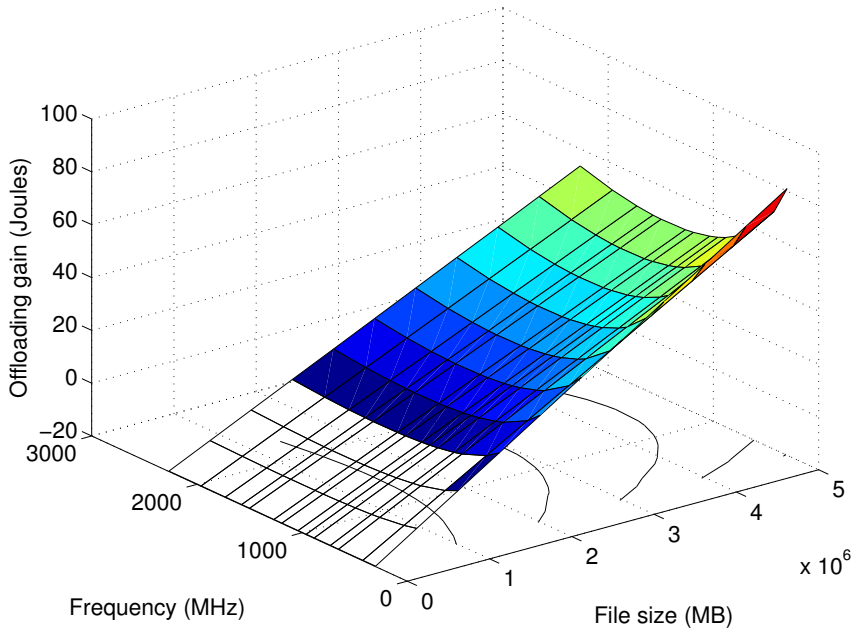
Figure 6.4: Offloading gain with 3G interface



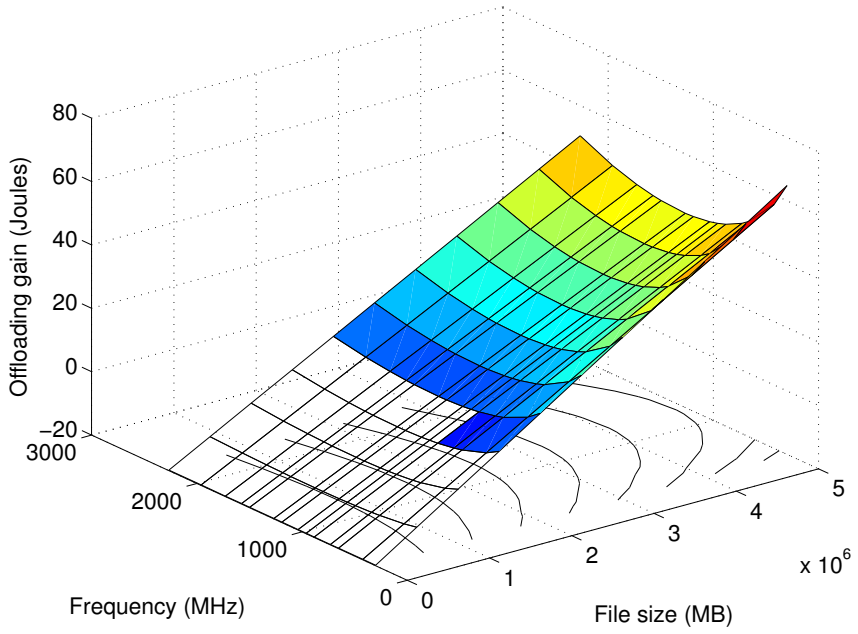
(a) 1 Core



(b) 2 Cores



(c) 3 Cores



(d) 4 Cores

Figure 6.5: Offloading gain with 4G interface

Chapter 7

Conclusions and Future Research

7.1 Conclusions

In recent years, the smartphone become one of the most essential devices on the hand of many people. The advances in the Internet, wireless communication, and the semiconductor technologies contribute to the popularity of the smartphones. However, the limited energy capacity of smartphones slows and limits the growth of smartphones capabilities. In this dissertation, the bright future of task offloading motivates us to proposed offloading framework that saves mobile devices energy with the aid of the cloud computing. The offloading technique promises to save energy and enriches the computing functionality of mobile devices. Since the offloading to the cloud is in its infant state, an in-depth study is required to evaluate and develop it.

Our evaluation study clearly indicates that offloading heavy applications such as multimedia applications from mobile devices to the cloud is beneficial. We conducted a large number of experiments for common network interfaces (3G/4G and WLAN) and Internet protocols (HTTP and FTP). The results reveal the potential of the cloud by reducing the energy consumptions at least 30%. Nevertheless, in some system configuration no energy is saved by the offloading. Therefore, the need offloading decision is crucial to offload a task whenever it is beneficial. In fact, the offloading framework needs energy estimation models to make the right decision.

Hence, we developed energy models for WLAN, 3G, and 4G wireless networks to allow the offloading framework to make correct offloading decisions. In these energy models, we considered the details of the network stack from lower level up to the transport level. The

models just need to know the amount of transferred data and some system parameters, and they can provide accurate estimations of energy cost.

For the computing energy consumption, we developed hardware and software energy models. These models empower mobile developers to estimate the energy consumption of an application on a given device. Moreover, we described how to profile modern multi-core CPUs, storage units, and applications under the *Android* platform. The results of the real experiments on two different smartphones reveal the validation and accuracy of our profiling procedures.

Finally, we used the developed models for the computing that networking to develop an offloading framework. In this framework, we introduced a set of parameters that are necessary to perform the offloading decision. We experimentally validated our offloading framework on several real life scenarios. The validation study emphasizes that our proposed framework is realistic and applicable to the mobile and cloud augmentation.

7.2 Future Research

This dissertation raises a number studies merit further investigation. The proposed energy models of WLAN, 3G, and 4G not only helps for task offloading but also opens new door for energy solutions that require predicting the energy consumption. Our developed mathematical models for WLAN can be extended to recent WLANs and broadband network standards such as *IEEE 802.11n* and *802.11ac* networks. Moreover, the impact of the number of WLAN network users on the energy consumption is needed because most of todays WLANs are multi-user networks. In addition, the impact of device mobility is worth examining for the networking models.

On the other hand, we avoid the application branching and we model the application for only one task. To extend our models and make it applicable to multi-task application, the application can be divided into tasks where each task is modelled using similar approach that we used. Furthermore, the real-time application and the effect of the user interfaces and user interactions with the application would be considered.

Finally, the offloading framework can be extended to consider more system parameters such as cloud waiting time and security parameters. Similarly, the offloading decision can be made in the cloud after uploading the system parameters as we proposed in Fig. 1.9. These highlight the importance of our study and the challenge for future research.

References

[1] N. Jones, "Mobile Web Trends 2007 to 2011," Gartner Group, Tech. Rep. G00148175, Jun. 2007. 1, 26

[2] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20112016," Feb. 2012. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf 1

[3] K. Naik, "A Survey of Software Based Energy Saving Methodologies for Handheld Wireless Communication Devices," Dept. of ECE, University of Waterloo, Waterloo, ON, Canada, Tech. Rep. 2010-13, 2010. 1, 14, 36, 37

[4] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010. 1, 14, 31, 32, 33, 34, 37, 38, 105

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009. 2, 27, 106

[6] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, "When Mobile Terminals Meet the Cloud: Computation Offloading as the Bridge," *Network, IEEE*, vol. 27, no. 5, pp. 28–33, 2013. 2, 31, 106

[7] M. Altamimi and K. Naik, "The Concept of a Mobile Cloud Computing to Reduce Energy Cost of Smartphones and ICT Systems," in *Proceedings of the First international conference on Information and Communication on Technology for the Fight against Global Warming (ICT-GLOW'11)*. Springer-Verlag, Aug. 2011, pp. 79–86. 2, 3, 36, 106

[8] M. Altamimi, R. Palit, K. Naik, and A. Nayak, "Energy-as-a-Service (EaaS): On the Efficacy of Multimedia Cloud Computing to Save Smartphone Energy," in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 764 –771. 3, 18, 34, 39, 79, 106, 117, 118

[9] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy Cost Models of Smartphones for Task Offloading to the Cloud," to appear in IEEE Transactions on Emerging Topics in Computing (TETC). 3, 18, 55, 92, 109, 117, 118

[10] M. Altamimi, K. Naik, P. Srivastava, and B. Plourde, "A Hardware Profiling Procedure for Smartphone App Developers to Estimate Energy Cost," submitted to IEEE 81st Vehicular Technology Conference (VTC2015-Spring). 3, 18, 83, 109

[11] M. Altamimi and K. Naik, "A Practical Task Offloading Decision Engine for Mobile Devices to Use Energy-as-a-Service (EaaS)," in *2014 IEEE World Congress on Services (SERVICES)*, Jun. 2014, pp. 452–453. 3, 18, 105, 106, 107

[12] ——, "A Framework to Offload Tasks from Mobile Devices to Cloud," submitted to ACM Transactions on Modeling and Performance Evaluation of Computing Systems. 3, 18, 105

[13] A. Shye, B. Scholbrock, and G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec. 2009, pp. 168 –178. 6, 9, 11

[14] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in Smartphone Usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 179–194. 6, 9, 14

[15] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A First Look at Traffic on Smartphones," in *Proceedings of the 10th annual conference on Internet measurement*, 2010, pp. 281–287. 6, 14

[16] M. Viviani, N. Bennani, and E. Egyed-Zsigmond, "A Survey on User Modeling in Multi-application Environments," in *Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC), 2010 Third International Conference on*, Aug. 2010, pp. 111 –116. 9

[17] C. Liang, "User Profile for Personalized Web Search," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 3, Jul. 2011, pp. 1847 –1850. 9

[18] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice, "Exhausting Battery Statistics: Understanding the energy demands on mobile handsets," in *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, ser. MobiHeld '10. ACM, 2010, pp. 9–14. 9

[19] R. Arya, R. Palit, and K. Naik, "A Methodology for Selecting Experiments to Measure Energy Costs in Smartphones," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, Jul. 2011, pp. 2087 –2092. 9, 40

[20] R. Palit, R. Arya, K. Naik, and A. Singh, "Selection and Execution of User Level Test Cases for Energy Cost Evaluation of Smartphones," in *Proc. of the 6th Int. Workshop on Automation of Software Test*, 2011, pp. 84–90. 9, 40, 42, 84, 115, 118

[21] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010, p. 21. 11

[22] G. Perrucci, F. Fitzek, and J. Widmer, "Survey on Energy Consumption Entities on the Smartphone Platform," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, May 2011, pp. 1 –6. 11

[23] J. Zhang, D. Wu, S. Ci, H. Wang, and A. Katsaggelos, "Power-Aware Mobile Multimedia: a Survey," *Journal of Communications*, vol. 4, no. 9, pp. 600–613, 2009. 14

[24] HTC Nexus One Google: Technical Specs. HTC. Accessed Apr. 2012. [Online]. Available: http://www.htc.com/us/support/nexus-one-google/tech-specs/ 14

[25] A. P. Miettinen and J. K. Nurminen, "Energy Efficiency of Mobile Clients in Cloud Computing," in *Proc. of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10)*, 2010, p. 4. 14, 33

[26] J. Paradiso and T. Starner, "Energy Scavenging for Mobile and Wireless Electronics," *Pervasive Computing, IEEE*, vol. 4, no. 1, pp. 18 – 27, January-March 2005. 14

[27] S. Robinson, "Cellphone Energy Gap: Desperately Seeking Solutions," Strategy Analytics, Tech. Rep., Mar. 30 2009. 14

[28] A. Brydon and M. Heath, "Will 3G Networks Cope?: 3G traffic and capacity forecasts, 2009-2014," Unwired Insight Limited, Tech. Rep., Sep. 2009. 22

[29] J. De Vriendt, P. Laine, C. Lerouge, and X. Xu, "Mobile Network Evolution: a Revolution on the Move," *IEEE Communications Magazine*, vol. 40, no. 4, pp. 104–111, 2002. 22, 23, 25, 26

[30] F. Bonomi, "The Future Mobile Infrastructure: Challenges and Opportunities," *IEEE Wireless Communications Magazine*, vol. 17, no. 5, pp. 4–5, 2010. 23

[31] A. H. Khan, M. A. Qadeer, J. A. Ansari, and S. Waheed, "4G as a Next Generation Wireless Network," in *Proc. Int. Conf. Future Computer and Communication*, 2009, pp. 334–338. 23

[32] J.-Z. Sun, J. Sauvola, and D. Howie, "Features in Future: 4G Visions from a Technical Perspective," in *Proc. IEEE Global Telecommunications Conf. GLOBECOM '01*, vol. 6, 2001, pp. 3533–3537. 23

[33] A. Akan and C. Edemen, "Path to 4G Wireless Networks," in *Proc. IEEE 21st Int Personal, Indoor and Mobile Radio Communications Workshops (PIMRC Workshops) Symp*, 2010, pp. 405–407. 23

[34] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A Survey of Mobile Phone Sensing," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 140–150, 2010. 25

[35] A. Dillistone, R. Moreno, and C. Voisey, "Market Perspective: Commercial Mobile Video," in *Proceedings of the 3rd workshop on Mobile video delivery*, ser. MoViD '10, 2010, pp. 69–72. 26

[36] AirPrint. Wikipedia.org. Accessed Nov. 2014. [Online]. Available: http://en.wikipedia.org/wiki/AirPrint 26

[37] (2011, Jan. 15) Sony Unveil Smartphone Gaming Solution. Cloud Computing Online. 26

[38] R. Cheng. (2011, Jun. 2) New Smartphone Targets Avid Gamers. Online. The Wall Street Journal. 26

[39] Z. Stern. (2010, Dec. 1) Process Credit Cards Anywhere: 5 Smartphone Alternatives. PCWorld. [Online]. Available: http://www.pcworld.com/businesscenter/article/211924/ 26

[40] B. Reed. (2010, Nov. 16) Big telcos aim to replace credit cards with smartphones. Network World. [Online]. Available: http://www.networkworld.com/news/2010/111610-isis-smartphones-credit-cards.html 26

[41] I. Union, "The World in 2010: ICT Facts and Figures," Oct. 20 2010. [Online]. Available: www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf 26

[42] ARMONK. (2007, Aug. 22) IBM Consumer Survey Shows Decline of TV as Primary Media Device. Online. IBM. 26

[43] K. Algar. (2011, Apr. 11) Faster internet speeds will increase consumer time online: Nielsen report. [Online]. Available: http://www.current.com.au/2011/04/11/article/26

[44] S. Zhang, S. Zhang, X. Chen, and S. Wu, "Analysis and Research of Cloud Computing System Instance," in *Proc. Second Int. Conf. Future Networks ICFN '10*, 2010, pp. 88–92. 27

[45] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, Jan. 2011. 27, 37, 38

[46] O. Manifesto, "Open Cloud Manifesto," *www.opencloudmanifesto.org*, vol. 20, pp. 1–7, 2009. 27, 30

[47] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *Proc. 24th IEEE Int Advanced Information Networking and Applications (AINA) Conf*, 2010, pp. 27–33. 27, 28, 29

[48] I. Bojanova and A. Samba, "Analysis of Cloud Computing Delivery Architecture Models," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, Mar. 2011, pp. 453 –458. 27

[49] Cloud Computing. Wikipedia.org. Accessed Nov. 2014. [Online]. Available: http://en.wikipedia.org/wiki/Cloud_computing 27, 30

[50] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, Dec. 2008. 27

[51] P. Mell and T. Grance, "The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology," *Nist Special Publication*, vol. 15, no. 800-145, pp. 1–3, Sep. 2011. 28, 29

[52] "Cisco Cloud Computing - Data Center Strategy, Architecture, and Solutions," White Paper, Cisco, 2009. 28

[53] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, "The Characteristics of Cloud Computing," in *Proc. 39th Int Parallel Processing Workshops (ICPPW) Conf*, 2010, pp. 275–279. 28

[54] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-Oriented Cloud Computing Architecture," in *Proc. Seventh Int Information Technology: New Generations (ITNG) Conf*, 2010, pp. 684–689. 28

[55] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013. 31, 33

[56] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using Bandwidth Data to Make Computation Offloading Decisions," in *Proc. IEEE Int. Symp. Parallel and Distributed Processing*, 2008, pp. 1–8. 31, 37, 38

[57] G. Chen, B.-T. Kang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and R. Chandramouli, "Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 795–809, Sep. 2004. 31, 37, 38

[58] B. Gao, L. He, L. Liu, K. Li, and S. Jarvis, "From Mobiles to Clouds: Developing Energy-Aware Offloading Strategies for Workflows," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, Sep. 2012, pp. 139 –146. 31

[59] R. Ferzli and I. Khalife, "Mobile Cloud Computing Educational Tool for Image/video Processing Algorithms," in *Proc. IEEE Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE)*, 2011, pp. 529–533. 32

[60] E. Lagerspetz and S. Tarkoma, "Mobile Search and the Cloud: The Benefits of Offloading," in *Proc. IEEE Int Pervasive Computing and Communications Workshops (PERCOM Workshops) Conf*, 2011, pp. 117–122. 32, 37

[61] J. Kim, "Architectural Patterns for Service-based Mobile Applications," in *Proc. IEEE Int Service-Oriented Computing and Applications*, 2010, pp. 1–4. 33, 38

[62] A. Khan, M. Othman, S. Madani, and S. Khan, "A Survey of Mobile Cloud Computing Application Models," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 393–413, First 2014. 33

[63] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013. 33, 34

[64] L. Guan, X. Ke, M. Song, and J. Song, "A Survey of Research on Mobile Cloud Computing," *Computer and Information Science, ACIS International Conference on*, vol. 0, pp. 387–392, 2011. 33

[65] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile Cloud Computing: A Survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84 – 106, 2013. 33

[66] S.-H. Hung, C.-S. Shih, J.-P. Shieh, C.-P. Lee, and Y.-H. Huang, "An Online Migration Environment for Executing Mobile Applications on the Cloud," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, 2011, pp. 20–27. 34, 35

[67] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *Proceedings of the Sixth Conference on Computer Systems*, 2011, pp. 301–314. 34

[68] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 1, pp. 6–15, 2012. 34, 35

[69] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A Computation Offloading Framework for Smartphones," in *Mobile Computing, Applications, and Services*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, M. Gris and G. Yang, Eds. Springer Berlin Heidelberg, 2012, vol. 76, pp. 59–79. 34, 35

[70] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Unleashing the Power of Mobile Cloud Computing using ThinkAir," *CoRR*, pp. 1–17, 2011. 34, 35

[71] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, 2010, pp. 49–62. 34, 38

[72] V. March, Y. Gu, E. Leonardi, G. Goh, M. Kirchberg, and B. S. Lee, "μCloud: Towards a New Paradigm of Rich Mobile Applications," *Procedia Computer Science*, vol. 5, no. 0, pp. 618 – 624, 2011. 34, 35

[73] K. Yang, S. Ou, and H.-H. Chen, "On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications," *IEEE Communications Magazine*, vol. 46, no. 1, pp. 56–63, 2008. 33, 34, 37, 38

[74] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving Portable Computer Battery Power through Remote Process Execution," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 2, pp. 19–26, Jan. 1998. 34

[75] Z. Li, C. Wang, and R. Xu, "Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme," in *Proc. of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. ACM, 2001, pp. 238–246. 34

[76] C. Wang and Z. Li, "Parametric Analysis for Adaptive Computation Offloading," in *Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation*, ser. PLDI '04, 2004, pp. 119–130. 34

[77] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten, "Access Schemes for Mobile Cloud Computing," in *Proc. Eleventh Int Mobile Data Management*, 2010, pp. 387–392. 36

[78] H. J. La and S. D. Kim, "A Conceptual Framework for Provisioning Context-aware Mobile Cloud Services," in *Proc. IEEE 3rd Int Cloud Computing*, 2010, pp. 466–473. 36

[79] M. Othman and S. Hailes, "Power Conservation Strategy for Mobile Computers Using Load Sharing," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 2, pp. 44–51, Jan. 1998. 37

[80] I. Kelenyi and J. K. Nurminen, "CloudTorrent - Energy-Efficient BitTorrent Content Sharing for Mobile Devices via Cloud Services," in *Proc. 7th IEEE Consumer Communications and Networking Conf. (CCNC)*, 2010, pp. 1–2. 37, 38

[81] X. Zhao, P. Tao, S. Yang, and F. Kong, "Computation Offloading for H.264 Video Encoder on Mobile Devices," in *Proc. IMACS Multiconference Computational Engineering in Systems Applications*, 2006, pp. 1426–1430. 37

[82] A. Olsen, F. Fitzek, and P. Koch, "Optimizing the Number of Cooperating Terminals for Energy Aware Task Computing in Wireless Networks," in *Wireless Personal Multimedia Communications Symposia*, 2005. 38

[83] S. Gitzenis and N. Bambos, "Joint Task Migration and Power Management in Wireless Computing," *IEEE Transactions on Mobile Computing*, vol. 8, no. 9, pp. 1189–1204, 2009. 38

[84] A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan, "Power of Clouds in Your Pocket: An Efficient Approach for Cloud Mobile Hybrid Application Development," in *Proc. IEEE Second Int Cloud Computing Technology and Science (CloudCom) Conf*, 2010, pp. 496–503. 42

[85] W. Itani, A. Chehab, and A. Kayssi, "Energy-Efficient Platform-as-a-Service Security Provisioning in the Cloud," in *Proc. Int Energy Aware Computing (ICEAC) Conf*, 2011, pp. 1–6. 46

[86] S. Ferretti, V. Ghini, F. Panzieri, and E. Turrini, "Seamless Support of Multimedia Distributed Applications Through a Cloud," in *Proc. IEEE 3rd Int Cloud Computing (CLOUD) Conf*, 2010, pp. 548–549. 46

[87] M. Rodriguez-Martinez, J. Seguel, M. Sotomayor, J. P. Aleman, J. Rivera, and M. Greer, "Open911: Experiences with the Mobile Plus Cloud Paradigm," in *Proc. IEEE Int Cloud Computing (CLOUD) Conf*, 2011, pp. 606–613. 46

[88] J. F. M. Bernal, L. Ardito, M. Morisio, and P. Falcarin, "Towards an Efficient Context-Aware System: Problems and Suggestions to Reduce Energy Consumption in Mobile Devices," in *Proc. Ninth Int Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR) Conf*, 2010, pp. 510–514. 53

[89] H. Yuan, C.-C. J. Kuo, and I. Ahmad, "Energy Efficiency in Data Centers and Cloud-Based Multimedia Services: An Overview and Future Directions," in *Proc. Int. Green Computing Conf*, 2010, pp. 375–382. 54

[90] Y. Zhang, N. Ansari, and H. Tsunoda, "Wireless Telemedicine Services over Integrated IEEE 802.11/WLAN and IEEE 802.16/WiMAX Networks," *IEEE Wireless Communications Magazine*, vol. 17, no. 1, pp. 30–36, 2010. 57, 87

[91] W. Jung, C. Kang, C. Yoon, D. Kim, and H. 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*. ACM, 2012, pp. 353–362. 57, 87

[92] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski, "Practical Power Modeling of Data Transmission over 802.11g for Wireless Applications," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, 2010, pp. 75–84. 57, 70, 81

[93] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. ACM, 2011, pp. 321–334. 57, 65

[94] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A Close Examination of Performance and Power Characteristics of 4G LTE Networks," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. ACM, 2012, pp. 225–238. 57, 65

[95] M. Lauridsen, P. Mogensen, and L. Noel, "Empirical LTE Smartphone Power Model with DRX Operation for System Level Simulations," in *IEEE 78th Vehicular Technology Conference*, Sep. 2013, pp. 1–6. 57

[96] Y. Xiao, R. S. Kalyanaraman, and A. Yla-Jaaski, "Energy Consumption of Mobile YouTube: Quantitative Measurement and Analysis," in *Proc. Second Int. Conf. Next Generation Mobile Applications, Services and Technologies NGMAST '08*, 2008, pp. 61–69. 58

[97] A. Abogharaf and K. Naik, "Client-Centric Data Streaming on Smartphones: An Energy Perspective," in *2013 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, 2013, pp. 36–41. 58, 84

[98] A. Albasir, K. Naik, and T. Abdunabi, "Smart Mobile Web Browsing," in *Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA), 2013 International Joint Conference on*. IEEE, 2013, pp. 671–679. 58, 84

[99] *IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11a-1999, 1999. 59, 60

[100] G. Bianchi, "Performance Analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, 2000. 59, 60, 61

[101] I. Grigorik, *High Performance Browser Networking: What Every Web Developer Should Know about Networking and Web Performance.* " O'Reilly Media, Inc.", 2013. 64

[102] J. D. Gibson, *Mobile Communications Handbook*, 3rd, Ed. CRC press, 2012. 66

[103] M. Assaad and D. Zeghlache, *TCP performance over UMTS-HSDPA systems.* CRC Press, 2006. 67

[104] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing Application Performance Differences on Smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services.* ACM, 2010, pp. 165–178. 67

[105] K. Ullah and J. Nurminen, "Applicability of Different Models of Burstiness to Energy Consumption Estimation," in *Communication Systems, Networks Digital Signal Processing (CSNDSP), 2012 8th International Symposium on*, Jul. 2012, pp. 1–6. 71

[106] A. Kansal and F. Zhao, "Fine-Grained Energy Profiling for Power-Aware Application Design," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 26–31, Aug. 2008. 84

[107] S. Hao, D. Li, W. Halfond, and R. Govindan, "Estimating Mobile Application Energy Consumption using Program Analysis," in *Software Engineering (ICSE), 2013 35th International Conference on*, May 2013, pp. 92–101. 84, 88

[108] A. Carroll and G. Heiser, "An Analysis of Power Consumption in a Smartphone," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference.* USENIX Association, 2010, pp. 21–21. 84, 85

[109] R. Mittal, A. Kansal, and R. Chandra, "Empowering Developers to Estimate App Energy Consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking.* ACM, 2012, pp. 317–328. 84

[110] K. Naik, Y. Ali, V. Mahinthan, A. Singh, and A. Abogharaf, "Categorizing Configuration Parameters of Smartphones for Energy Performance Testing," in *Proceedings of the 9th International Workshop on Automation of Software Test.* ACM, 2014, pp. 15–21. 86

[111] A. Noureddine, R. Rouvoy, and L. Seinturier, "A Review of Energy Measurement Approaches," *SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, Nov. 2013. 86

[112] Y. Li, H. Chen, and W. Shi, "Bugu: an Application Level Power Profiler and Analyzer for Mobile Devices," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 17, no. 3, pp. 27–28, 2013. 87

[113] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications.* IEEE Computer Society, 1999, pp. 2–9. 87

[114] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12).* Boston, MA: USENIX, 2012, pp. 387–400. 88

[115] T. Do, S. Rawshdeh, and W. Shi, "pTop: A Process-level Power Profiling Tool," in *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, 2009. 88

[116] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems.* ACM, 2012, pp. 29–42. 88

[117] S. Hao, D. Li, W. Halfond, and R. Govindan, "Estimating Android Applications' CPU Energy Usage via Bytecode Profiling," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, 2012, pp. 1–7. 88, 117, 118

[118] K. Choi, W. Lee, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling Under a Precise Energy Model Considering Variable and Fixed Components of the System Power dissipation," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, Nov. 2004, pp. 29–34. 88

[119] S. Kaxiras and M. Martonosi, *Computer architecture techniques for power-efficiency.* Morgan and Claypool Publishers, 2008. 89, 90

[120] R. Gonzalez, B. Gordon, and M. Horowitz, "Supply and threshold voltage scaling for low power CMOS," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, Aug. 1997. 90

[121] L. Barroso and U. Holzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007. 91

[122] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2012. 93, 100

[123] W. Zhang, Y. Wen, and H.-H. Chen, "Toward Transcoding as a Service: Energy-Efficient Offloading Policy for Green Mobile Cloud," *Network, IEEE*, vol. 28, no. 6, pp. 67–73, Nov. 2014. 108

[124] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda, "Characterizing and Modeling User Activity on Smartphones: Summary," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 375–376, Jun. 2010. 115, 118

[125] Android ConnectivityManager Class. Android Developer. Accessed Dec. 2014. [Online]. Available: https://developer.android.com/reference/android/net/ConnectivityManager.html 115, 118

[126] P. Papakos, L. Capra, and D. S. Rosenblum, "VOLARE: Context-aware Adaptive Cloud Service Discovery for Mobile Systems," in *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*. ACM, 2010, pp. 32–38. 115, 118

[127] T. Han and K. M. Sim, "An Ontology-Enhanced Cloud Service Discovery System," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2010, pp. 17–19. 115, 118

[128] A. Diaz, P. Merino, and F. J. Rivas, "Mobile Application Profiling for Connected Mobile Devices," *IEEE Pervasive Computing*, vol. 9, no. 1, pp. 54–61, 2010. 115, 118

[129] C. Thompson, J. White, B. Dougherty, and D. Schmidt, "Optimizing Mobile Application Performance with ModelDriven Engineering," in *Software Technologies for Embedded and Ubiquitous Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5860, pp. 36–46. 115, 118

[130] Apple iOS Get Battery Status. Apple Developer. Accessed Dec. 2011. [Online]. Available: https://developer.apple.com/library/ios/samplecode/BatteryStatus/ 117, 118

[131] Android BatteryManager Class. Android Developer. Accessed Nov. 2014. [Online]. Available: http://developer.android.com/reference/android/os/BatteryManager.html 117, 118

[132] R. Palit, A. Singh, and K. Naik, "Modeling the Energy Cost of Applications on Portable Wireless Devices," in *Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems*, 2008, pp. 346–353. 117, 118

[133] P. Bellasi, W. Fornaciari, and D. Siorpaes, "Predictive Models for Multimedia Applications Power Consumption based on Use-Case and OS Level Analysis," in *Design, Automation and Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 1446–1451. 117, 118

Author's Publications

[1] M. Altamimi, K. Naik, and X. Shen, "Parallel Link Rendezvous in Ad Hoc Cognitive Radio Networks," in *Proc. IEEE Global Telecommunications Conf. GLOBECOM 2010*, 2010, pp. 1–6.

[2] M. Altamimi and K. Naik, "The Concept of a Mobile Cloud Computing to Reduce Energy Cost of Smartphones and ICT Systems," in *Proceedings of the First international conference on Information and Communication on Technology for the Fight against Global Warming (ICT-GLOW'11)*. Berlin, Heidelberg: Springer-Verlag, August 2011, pp. 79–86.

[3] R. Palit, M. Altamimi, K. Naik, and A. Singh, "Challenges and Opportunities in Designing Next Generation Energy-efficient Smartphones," in *invited paper in the proceedings of the 1st International Conference on Advanced Computing (ICoAC 2011)*, Chennai, India, December 14-16 2011.

[4] M. Altamimi, R. Palit, K. Naik, and A. Nayak, "Energy-as-a-Service (EaaS): On the Efficacy of Multimedia Cloud Computing to Save Smartphone Energy," in *IEEE 5th International Conference on Cloud Computing (CLOUD)*, June 2012, pp. 764 –771.

[5] M. Altamimi and K. Naik, "A Practical Task Offloading Decision Engine for Mobile Devices to Use Energy-as-a-Service (EaaS)," in *2014 IEEE World Congress on Services (SERVICES)*, June 2014, pp. 452–453.

[6] M. Altamimi, A. Abdrabou, K. Naik, and A. Nayak, "Energy Cost Models of Smartphones for Task Offloading to the Cloud," to appear in *IEEE Transactions on Emerging Topics in Computing (TETC)*.

[7] M. Altamimi, K. Naik, P. Srivastava, and B. Plourde, "A Hardware Profiling Procedure for Smartphone App Developers to Estimate Energy Cost," submitted to *IEEE 81st Vehicular Technology Conference (VTC2015-Spring)*.

[8] M. Altamimi and K. Naik, "A Framework to Offload Tasks from Mobile Devices to Cloud," submitted to ACM Transactions on Modeling and Performance Evaluation of Computing Systems.