Optimal Mobile Computation Offloading With Hard Task Deadlines

OPTIMAL MOBILE COMPUTATION OFFLOADING WITH HARD TASK DEADLINES

BY

ARVIN HEKMATI, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2019) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Optimal Mobile Computation Offloading With Hard Task Deadlines
AUTHOR:	Arvin Hekmati B.Sc., (Electrical Engineering) University of Tehran, Tehran, Iran
SUPERVISOR:	Dr. Terence D. Todd Dr. George Karakostas
NUMBER OF PAGES:	xiii, 91

To my beloved mother

Nooshin

Abstract

This thesis considers mobile computation offloading where task completion times are subject to hard deadline constraints. Hard deadlines are difficult to meet in conventional computation offloading due to the stochastic nature of the wireless channels involved. Rather than using binary offload decisions, we permit concurrent remote and local job execution when it is needed to ensure task completion deadlines. The thesis addresses this problem for homogeneous Markovian wireless channels. Two online energy-optimal computation offloading algorithms, OnOpt and MultiOpt, are proposed. OnOpt uploads the job to the server continuously and MultiOpt uploads the job in separate parts, each of which requires a separate offload initiation decision. The energy optimality of the algorithms is shown by constructing a time-dilated absorbing Markov process and applying dynamic programming. Closed form results are derived for general Markovian channels. The Gilbert-Elliott channel model is used to show how a particular Markov chain structure can be exploited to compute optimal offload initiation times more efficiently. The performance of the proposed algorithms is compared to three others, namely, Immediate Offloading, Channel Threshold, and Local Execution. Performance results show that the proposed algorithms can significantly improve mobile device energy consumption compared to the other approaches while guaranteeing hard task execution deadlines.

Acknowledgements

Foremost, I would like to express my sincere appreciation to my supervisors, Dr. Terence Todd, Dr. Dongmei Zhao, and Dr. George Karakostas, for their patience, encouragement and support through my masters at McMaster University. I am truly grateful for working under their supervision.

I would like to record my warmest gratitude to my parents since their love and endless support have made it possible for me to excel in my studies.

My kind regards to my fellow colleagues in the Wireless Networking Laboratory, who I have had the pleasure of working with. Special thanks to my friend and lab mate, Peyvand Teymoori for her intellectual supports and suggestions.

Last but not the least, I am particularly grateful to my friends in Hamilton for all the fun and unforgettable moments we had together.

Notations

t_r	release time of the job (time slots)
t_D	hard deadline of the job (time slots)
S_{up}	upload data size (bits)
S_{up_1}	upload data size of the first part (bits)
S_{up_2}	upload data size of the second part (bits)
S_{down}	download data size (bits)
T_L	local execution time duration (time slots)
t_L	local execution start time duration (time slot)
T_D	job completion time duration deadline (time slots)
t_D	job completion time deadline (time slot)
T_{up}	upload data time duration (time slots)
T_{up_1}	upload data time duration for the first part (time slots)
T_{up_2}	upload data time duration for the second part (time slots)
T_{exec}	remote execution time (time slots)
T_{down}	download data time duration (time slots)
T_{off}	total offloading time duration (time slots)

t_o	start uploading time (time slot)
t_{o_1}	start uploading time of the first part (time slot)
t_{o_2}	start uploading time of the second part (time slot)
t_f	finish uploading time (time slot)
t_{f_1}	finish uploading time of the first part (time slot)
t_{f_2}	finish uploading time of the second part (time slot)
T_W	elapsed time between two uploads (time slots)
B_{min}	minimum bitrate
B_{max}	maximum bitrate

Abbreviations

3G	Third Generation
CSMC	Channel State Markov Chain
GPRS	General Packet Radio Services
IDC	International Data Corporation
IP	Integer Programming
IT	Information Technology
IaaS	Infrastructure as a Service
LAN	Local Area Network
LTE	Long Term Evolution
MCC	Mobile Cloud Computing
MultiOpt	Mutli-Decision online Optimum)
OnOpt	Online Optimum
PaaS	Platform as a Service
QoS	Quality of Service

SaaS	Software as a Service
TDAMC	Time-Dilated Absorbing Markov Chain
Wi-Fi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

Contents

List of Figures

Chapter 1

Introduction

In the past decade, the number of mobile devices such as phones and tablets has increased significantly. An International Data Corporation (IDC) five-year forecast suggests that the worldwide smartphone market will reach a total of 1.39 billion units shipped in 2019 and will reach 1.54 billion units by 2023 (Scarsella and Stofega, 2019).

In addition to their traditional functionality, mobile devices are continuing to become more pervasive as personal computing platforms. This trend is coinciding with significant increases in mobile application features that benefit from tight interactions with fixed computation infrastructure. According to a recent report, Cisco Inc. predicts that by the year 2021, monthly worldwide mobile data traffic will approach 28 exabytes (Cis, 2017).

Nowadays, mobile phones are often equipped with various sensors which help the device capture environmental inputs and also provide benefits to users such as improved healthcare. These developments are making them an essential part of humans' lives. One of the most essential features of mobile devices is their portability, which allows users do computation tasks anywhere at anytime (Bahwaireth *et al.*, 2015; Wei *et al.*, 2013). This includes computationally intensive tasks such as natural language translation (Balan *et al.*,

2007; Flinn *et al.*, 2002), speech recognition (Balan *et al.*, 2007; Su and Flinn, 2005), optical character recognition, image processing (Kristensen and Bouvin, 2008; Porras *et al.*, 2009), online gaming, and video processing (Chun and Maniatis, 2009; Shiraz *et al.*, 2013). However, mobile devices are inherently resource-constrained due to their small physical size. This limits their ability to satisfy the processing and energy required for resource intensive processing. Because of this, application developers are unable to provide many mobile resource-intensive features such as artificial vision and object tracking. Smartphones suffer from short battery lifetimes, insufficient memory, low processing capability, and constrained storage space. Among these, mobile battery lifetime is by far the most common smartphone complaint (Paczkowski, 2009). This has motivated a wide variety of recent research on mobile energy efficiency (Kumar and Lu, 2010).

In order to overcome the limited battery lifetime issue of mobile devices, reference (Kumar and Lu, 2010) suggested four different approaches to reduce the energy consumption. In the first approach, new semiconductor technologies can be used where transistors are more energy efficient. The problem with this approach is that by decreasing transistor size, a higher number of transistors may be needed to maintain a good level of performance. Therefore, this action may eventually increase energy consumption. In the second approach, we can sleep components of the mobile device that are not in use. In the third approach, we can run the programs slowly by using the CPU at a lower clock speed, which results in a reduction of consumed energy. However, this approach may reduce user experience below a minimum satisfactory level. Finally, we can eliminate or reduce the computations performed locally on the mobile device by offloading execution to a remote cloud server. This approach is referred to as mobile cloud computing in the literature.

In this thesis, we focus on the fourth approach by considering the optimal offloading

times under random channel conditions so that mobile energy use is minimized. This is done subject to satisfying hard task execution deadline constraints.

1.1 Mobile Cloud Computing

Nowadays, cloud computing is pervasive and provides excellent computing capabilities for various applications. However, it does not have a specific meaning, and it has been defined by various industry practitioners, academics, analyst firms, and Information Technology (IT) companies. According to the Berkeley RAD Lab, cloud computing is the aggregation of two different aspects. The first is the services and applications which are provided to users over the internet, and the second is the components of the datacenters such as hardware and systems software which provide the infrastructure for those applications and services. We generally call the data center hardware and software the cloud, and Software as a Service (SaaS) refers to the services provided to the end users. If the cloud is available to the public as a utility, it is defined as a Public Cloud. On the other hand, a Private Cloud refers to the internal utilities of a company which is not available to the public. Private Clouds are not considered as a part of Cloud Computing (Armbrust *et al.*, 2009).

As can be seen in Figure 1.1, the structure of Cloud Computing has three architectural layers. The first layer is called Infrastructure as a Service (IaaS), which is essentially the processing or storage services that are provided to the users. Examples of this are Amazon Web Services with its Elastic Compute Cloud (EC2) for processing and Simple Storage Service (S3) for storage. The second layer is called Platform as a Service (PaaS), which provides platforms for the developers to write and run their code. In this layer, the developers are not concerned with the hardware infrastructure of the lower layer (Iaas). An example of PaaS is the Google App Engine, which allows applications to run on Googles

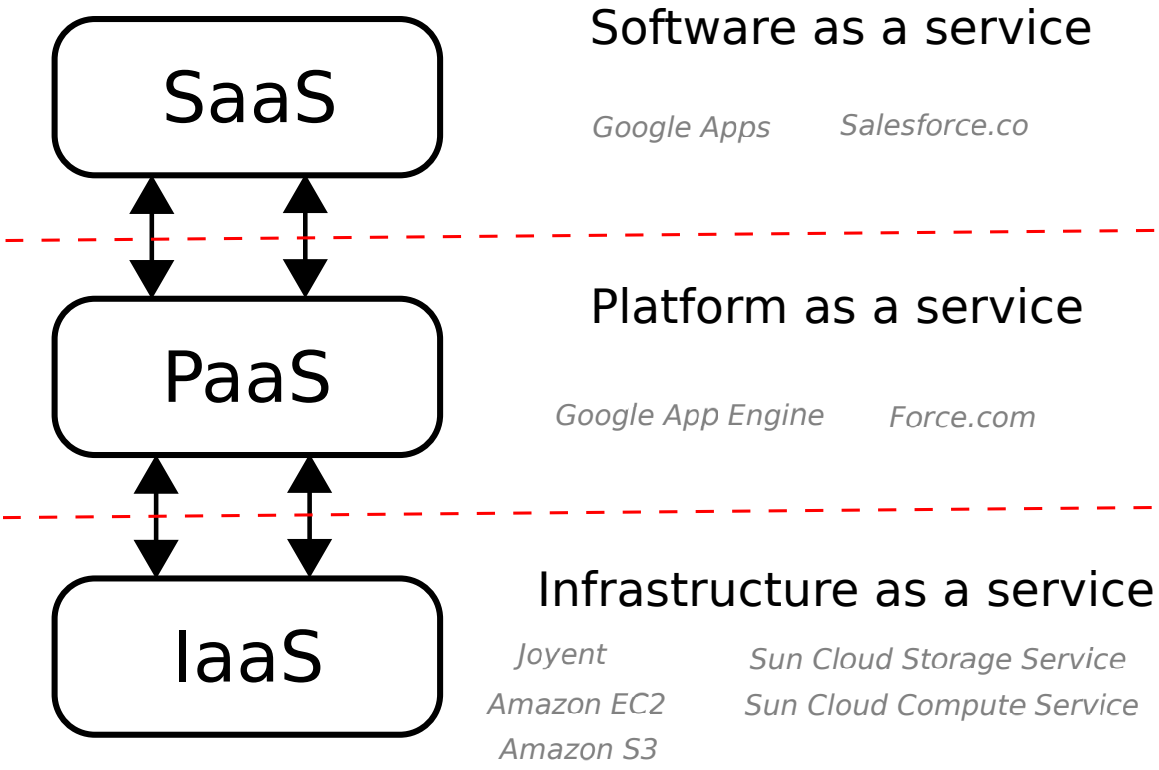

Figure 1.1: The Three Layers of Cloud Computing

platform. The third layer is called Software as a Service (SaaS) which provides on-demand applications over the internet. This layer is the most visible part of cloud computing to the users. Examples of SaaS offerings are Salesforce.com and Google Apps such as Google Mail and Google Docs and Spreadsheets. The consumers of this layer are not concerned with the underlying layers (Patidar *et al.*, 2012).

Nowadays, mobile devices have the benefits of compact design, high-quality graphics, customized user application support and multimodal connectivity features. Mobile phones employ various wireless network technologies for accessing the internet; such as 3G connectivity, Wireless Fidelity (Wi-Fi), Worldwide Interoperability for Microwave Access (Wi-Max), and Long Term Evolution (LTE). The latest development in smartphones has made people more intent to run heavy computational tasks on their devices, as is the case with powerful stationary computers. Although lots of improvements have been made in the software and hardware, mobile devices are still poor computing devices due to their constrained resources such as limited CPU potential, memory capacity, and battery lifetime. In addition, they have other limitations because of their light weight, small size, and restrictions due to the wireless medium and mobility (Shiraz *et al.*, 2013).

Mobile Cloud Computing (MCC) has been introduced to help alleviate some of these shortcomings, and to support the ever increasing computation and storage demands for mobile devices (You *et al.*, 2016; Chiang and Zhang, 2016). Cloud computing provides new powerful resources for Information Technology (IT) services. Cloud-based services and applications are on-demand, scalable, device-independent and reliable. Therefore, the MCC purpose is to use cloud computing techniques and resources for storage and processing of data on mobile devices (Guan *et al.*, 2011). Vast resources are available in the cloud servers to help smartphones with computational functions. Therefore, cloud resources can be used

to deliver elastic computing power and also storage to support resource-constrained end-user devices. Its goal is to centralize the computation, storage, and network management (Mao *et al.*, 2017; Armbrust *et al.*, 2009; Zhang *et al.*, 2010).

It has been estimated that tens of billions of future cloud-based network edge devices will be deployed to satisfy mobile demands. This will provide significant resources for performing computationally intensive and latency-critical mobile-centric tasks (ETSI, 2014; Mao *et al.*, 2017). Mobile computation offloading has been proposed as a way of decreasing mobile device energy use by dynamically offloading job execution to infrastructure based cloud servers (Chun *et al.*, 2011; Chun and Maniatis, 2009; Satyanarayanan *et al.*, 2009; Huerta-Canepa and Lee, 2010; Ba *et al.*, 2013). It has been demonstrated that task offloading can significantly improve battery lifetime compared to the non-offloading case (Rudenko *et al.*, 1998, 1999).

Some of the important applications which can be run on mobile devices by using computation offloading are natural language processing, object/gesture recognition, and image/video editing. In these heavy computational algorithms, we should consider task precedence requirements to achieve the maximum energy reduction for the mobile devices. Therefore, by partitioning large applications into several tasks and offloading the computationally heavy ones to the cloud, the requirements on the mobile CPU decrease, and as a result, battery lifetime and also quality of service can increase. Therefore, smartphones can run more sophisticated applications and achieve a better user experience (Lin *et al.*, 2014).

1.2 Motivation and Contribution

In this thesis we study mobile computation offloading where job completion times are subjected to *hard* deadline constraints, i.e., deadline constraints that should *never* be violated, as opposed to deadline constraints that are satisfied with high probability, or incur a penalty when violated, etc. This objective will become increasingly important as mobile applications become more sophisticated and interact more closely with cloud job execution (Lagar-Cavilla *et al.*, 2007). Hard deadlines, however, are often difficult to achieve in mobile networks due to the randomness of the wireless channels used for the mobile/cloud data interactions. In harsh wireless conditions, for example, complete channel outage can even occur over extended time periods. In this work, we study the approach of permitting *concurrent* local and cloud offload execution when a job completion deadline must be respected. This is in contrast to the conventional computation offloading model where job execution is either local or remote. As is the case in conventional computation offloading, the objective is to reduce the mobile device energy needed for job execution.

The main contributions of this thesis are summarized, as follows.

1. To the best of our knowledge, this is the first work that uses computation offloading to reduce mobile energy and provides a mechanism for guaranteeing that hard job deadlines are always satisfied, even in the presence of full wireless channel outage conditions.

2. Online offloading decision algorithms, i.e., OnOpt and MultiOpt, are introduced. It is theoretically proven that the algorithms not only satisfy hard deadline constraints of the applications with certainty, but also achieve the minimum mean mobile device energy possible for homogeneous Markovian wireless channels. The OnOpt

algorithm makes the optimal decision for continuous offloading in a single piece. MultiOpt uploads data in two pieces. This can sometimes decrease mobile device energy consumption compared to OnOpt.

3. Two integer programs (IPs) are formulated that compute strict lower bounds on mobile device energy for both single and multi-part offloading. These bounds are used for comparisons in our performance results.

4. Closed form results are derived for obtaining job completion time probabilities for the homogeneous Markovian wireless channel case.

5. Although the proposed OnOpt and MultiOpt algorithms satisfy hard deadlines and are proven to be energy optimal, performance results are also presented that compare them with the computation offloading heuristics: Immediate Offloading, Channel Threshold and Local Execution. These algorithms also ensure that hard job deadlines are preserved.

The rest of this thesis is organized as follows. Chapter 2 introduces the architectures and challenges of the mobile cloud computing. In Chapter 3, we describe the system and present a model for local and remote job execution that satisfies hard job execution deadline constraints. Following this, we derive an offline lower bound on the energy consumption, which is plotted in the results section and compared to various offloading algorithms. We also discuss the Markovian channel model and how it is used to form a time-dilated absorbing Markov chain.This construction permits us to apply dynamic programming and derive the energy optimum online algorithms (OnOpt and MultiOpt), proposed in Chapter 4. Then, in Chapter 5 the thesis focuses on the Gilbert-Elliot channel model, where it is

shown that calculations can be performed efficiently, decreasing the complexity of algorithm running time. In Chapter 6, performance results are presented that compare OnOpt and MultiOpt with various other computation offloading algorithms that ensure that hard job deadlines are preserved. Finally, we present our conclusions in Chapter 7.

Chapter 2

Literature Review

Figure 2.1 shows the basic architecture of mobile cloud computing. It consists of mobile end-user devices, wireless internet communication facilities, and a computational cloud. The mobile device sends data to the cloud for remote processing by using wireless transmission/reception and then receives back the results of the remote computation. This procedure may help the device to save energy by reducing local execution at the mobile device (Shiraz *et al.*, 2013).

In order to fully describe the process of mobile cloud computing, reference (Guan *et al.*, 2011) has proposed a "concept model", which has three layers, as shown in Figure 2.2.

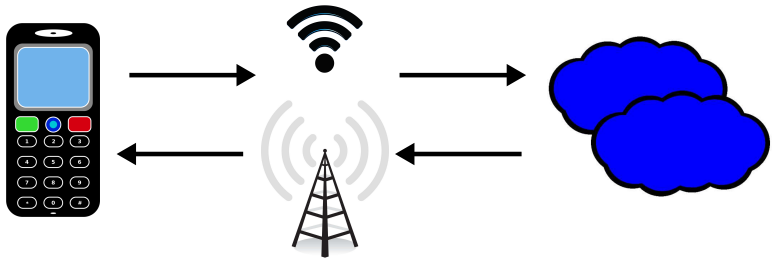

Figure 2.1: Model of Mobile Cloud Computing

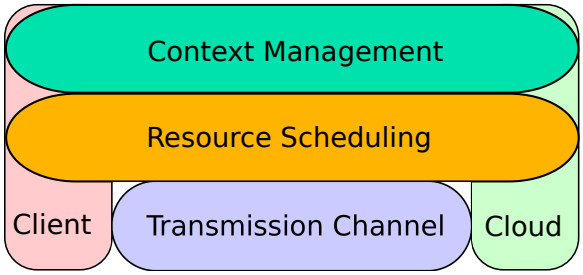

Figure 2.2: Concept Model of Mobile Cloud Computing

On the left, we have the client, and on the right, the cloud. These two components are connected via transmission channels, which can include various wireless communication protocols. The above layers are referred to as Context Management and Resource Scheduling. This model assumes that the cloud delivers elastic, on-demand services and the client is context-aware. In mobile cloud computing, we have computing resources and storage resources, which are scheduled by the Resource Scheduling layer. Virtual machines are commonly used in this layer to handle resource dispatching. The Context Management layer can track context parameters of the mobile device in order to adapt to changes in contextual conditions. One of the examples of the context parameters is the position of the mobile device. Sharing the position of the mobile devices with the cloud helps them to better discover the position of the computational resources available near them. This information helps them to select the best resources for offloading with lowest transmission delay.

Various architectures have been proposed for mobile computation offloading. Reference (Cuervo *et al.*, 2010) originally proposed an architecture known as MAUI, which controls computation offloading for runtime .NET applications by formulating the offloading problem as a linear program. A similar architecture for Android applications has been proposed in (Chun *et al.*, 2011). Other popular architectures are MobiCloud, CloneCloud,

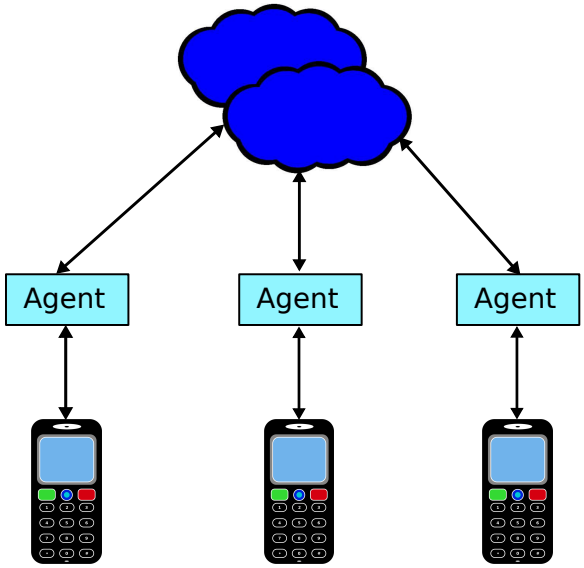

Figure 2.3: Agent-Client Architecture

Cloudlet, and Hyrax, which supports a variety of different types of applications (Wei *et al.*, 2013). These various types of architectures can be divided into two main categories according to the location where mobile users execute their applications and how the devices and clouds are connected.

In an *agent-client* architecture, as shown in Figure 2.3, each mobile user is assigned an agent from the cloud server, and the mobile device only communicates with this agent to contact other entities. These agents help the mobile devices overcome their limitations in processing power and storage capacity (Guan *et al.*, 2011).

Reference (Satyanarayanan *et al.*, 2009) presents an agent-client design. In this architecture, each mobile device is assigned resources to a virtual machine that is running on the server and connects to it via a wireless LAN. Servers are decentralized and widely distributed over the Internet infrastructure. The practical implementation of this scheme is to equip Wi-Fi access points with substantial processing, memory, and storage resources.

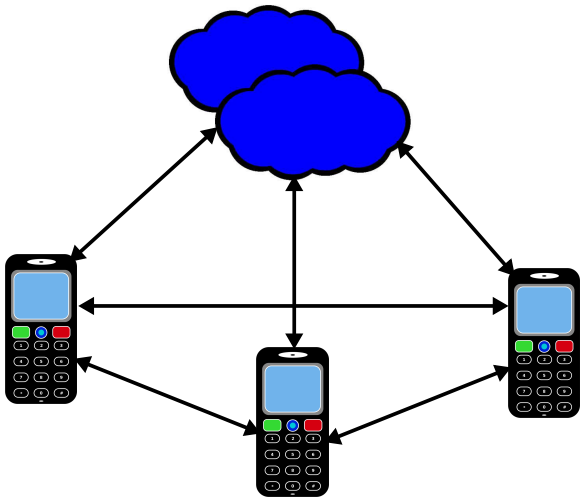

Figure 2.4: Collaborated Architecture

Another example of an agent-client scheme is given in (Zhang *et al.*, 2009). The authors introduce elastic applications which consist of one or more tasks that function independently. Each task can be run on the mobile device or be offloaded to the cloud. An elastic application manager runs on the device, which assigns computation intensive tasks to the cloud platforms and executes the other tasks locally.

As shown in Figure 2.4, in a *collaborated architecture*, each device is considered as a part of the cloud. Therefore, the tasks of each mobile can be offloaded to other mobile devices as well as the cloud servers (Guan *et al.*, 2011).

Hyrax (Marinelli, 2009) is an example of a collaborated architecture which supports cloud computing on Android phones. This platform enables mobile devices to run their applications on the network of smartphones and servers. Reference (Black and Edgar, 2009) discusses the feasibility and benefits of a collaborated scheme by enabling mobile devices within a computing platform grid and implements the client on an Apple iPhones. Offloaded tasks are available at the server and can be downloaded and executed on the iPhones via a virtual machine emulating an x86 processor. After execution, results will be

uploaded to the server.

2.1 Network Latency and Limited Bandwidth

MCC often requires frequent communication between the cloud platform and the mobile device. Unfortunately, transmission channels are often impaired by randomness associated with resource contention and noise. As a result, they often suffer from high latency and limited usable bandwidth. This issue becomes especially important for mobile applications such as online games and augmented reality, which require high processing capacity along with low latency. As a result, applications of this kind will often be executed locally on the mobile device. One must be careful to offload only those applications that are compatible with these effects. Reference (Galinina *et al.*, 2013) ensures the required minimum quality of service by maximizing the energy efficiency of a smartphone transmitting on several wireless channels. In this solution, the optimal transmit power is obtained by solving an optimization problem that is based on Shannon's capacity result.

2.2 Application Models

Computation offloading helps the mobile devices by running heavy computational applications on a cloud platform. It is often important to design elastic mobile applications, i.e., those that can be decomposed into multiple tasks, each of which can be executed locally or remotely. In this case, the mobile device can offload the computationally intensive tasks to the cloud. Furthermore, elastic applications can help the device to decide whether to run each task locally or remotely according to the users' preference and utility.

The first important challenge for elastic application models is how to partition the mobile application. Reference (Giurgiu *et al.*, 2009) introduced a two-step procedure for optimal application partitioning. In the first step, a flow graph is constructed of the application modules using the described behaviour. Then, in the next step, the algorithm finds the optimal partitioning of the job which minimizes energy consumption. Two different types of partitioning are proposed in this paper. In the first algorithm, the best partitioning considers the various mobile devices and all possible network conditions to determine the optimal partitioning. The second algorithm runs on-the-fly when the phone connects to the cloud. In this case the partitioning algorithm only considers the current server and network resources.

The second important issue with Elastic Application Models is how to choose the partitions to be offloaded to the cloud. Reference (Cuervo *et al.*, 2010) presents MAUI, which offloads fine-grained mobile code to the cloud for remote execution.

2.3 Mobile Access Mechanisms

Mobile devices can communicate to a cloud server over various radio access technologies such as GPRS, 3G, LTE, WLAN or WiMax. MCC should choose the best available option that satisfies its performance and minimizes the energy consumption of the mobile device.

Reference (Rahmati and Zhong, 2007) designed an online wireless network selection mechanism. It finds the optimal solution for either offloading using the current Wi-Fi network or to search for a better Wi-Fi connection. In (Gribaudo *et al.*, 2013) the dynamics of user traffic are monitored and radio network resources are allocated to mobile users to minimize user energy consumption.

2.4 Related Work

Many new mobile cloud computing issues and challenges have been addressed in the past few years (Abolfazli *et al.*, 2014; Rehman Khan *et al.*, 2014; Liu *et al.*, 2013; Guan *et al.*, 2011). A significant part of the literature has considered mobile computation offloading issues under stochastic transmission channel and cloud server conditions. Reference (Kumar and Lu, 2010), for example, presents an energy model to analyze offloading, mainly considering mobile computation and communication energy components based on statistical inputs and with fixed wireless channel conditions. This work analyzed the offloading policy assuming that network conditions remain static throughout the offloading/execution process. Network prediction was used as inputs to the decision process. In (Zhang *et al.*, 2015) a method was proposed for energy-optimal mobile cloud computing under stochastic wireless channels. The issue of job deadlines was considered from a statistical viewpoint, rather than enforcing hard job execution deadlines. Dynamic programming was used in (Liu and Lee, 2014) to optimize offloading decisions from an energy viewpoint, but the issue of job execution time constraints was not considered. In Reference (Lagar-Cavilla *et al.*, 2007), job execution time constraints were flagged as a key issue for many interactive applications. The difficulties of achieving this under random channel conditions were highlighted. In (Zhang *et al.*, 2013), a framework was proposed for executing jobs either locally, by CPU frequency scheduling, or remotely, by offloading over a stochastic channel. In the latter case, mobile transmit power control is used to select bit rates to ensure that job deadlines are met. In local execution, a violation parameter is defined that permits the execution to probabilistically exceed the deadline, and, therefore, the latter is not "hard" in our sense. As in this thesis, this work uses the well-known Gilbert-Elliot channel model for its results.

Chapter 3

System Model and Problem Formulation

We consider a statistical wireless communication channel which is modelled by a Markov chain. The computation offloading is subject to a hard deadline, i.e., the execution of each job should be finished by a specific time. In this chapter, we describe the details of our system model, followed by an offline bound of the problem.

3.1 System Model

Continuous Offloading

We consider the execution of computational tasks (jobs) generated by a mobile device, either locally (by the device itself), or by offloading them on a remote cloud server, through a wireless transmission channel. Each job could be a sub-task associated with multiple local/remote job execution components (You *et al.*, 2016; Chiang and Zhang, 2016). We focus on a single task whose characteristics are known at its release time. Note that time is taken to be discrete, i.e., quantized into equal length *time slots* whose duration is normalized to 1. Time values are therefore referred to by their time slot indices. Note that the

time slot duration is defined to accommodate the channel propagation model discussed in Section 3.2 and may contain multiple packet transmission times on the channel. Each job to be executed is characterized by the following:

t_r**:** *Release time* of the job, i.e., the time when the job is ready to start execution, either locally or via offloading. For convenience, we will assume that $t_r = 1$.

t_D**:** *Hard deadline* of the job, i.e., the job execution results *must* be available at the mobile device by time t_D. $T_D = t_D - t_r + 1$ is the maximum number of time slots available for completing the job.

S_{up}**:** Number of bits transmitted through the uplink channel when uploading the job to the cloud.

S_{down}**:** Number of bits transmitted through the downlink channel when downloading job results from the cloud.

Multi-Part Offloading

We also consider a generalization of the previous problem *(multi-part offloading)*, where each job can be split into two parts for offloading, each with a (known) number S_{up_1}, S_{up_2} of bits to be transmitted through the uplink channel, respectively, for a total of $S_{up} = S_{up_1} + S_{up_2}$ bits. S_{down} bits are transmitted through the downlink channel when downloading job results from the cloud.

We now discuss the timing and energy use associated with local and remote offloaded job execution.

3.1.1 Local Execution

It is assumed that the energy cost and time needed to execute a job locally is known at the job release time, t_r, and these are defined by E_L and T_L, respectively. While this may not always be the case, this assumption is often true and has been made in many computational offloading studies (Kumar and Lu, 2010; Chen, 2015; Cao and Cai, 2018).

If the computation offloading algorithm elects to execute the job locally without any remote offloading, we must ensure that the job deadline is always satisfied. Therefore, local execution must start no later than

$$t_L = t_D - T_L + 1, \tag{3.1}$$

unless remote offload/execution results are available at the mobile device before t_L, i.e., local execution must start T_L time slots prior to the job deadline, if remote execution results have not arrived by then.

3.1.2 Remote Execution

Continuous Offloading

In the case of offloading a job, we will assume that, upon its release, the job is assigned an execution time T_{exec} by the cloud server, which is communicated to the mobile device (or is prescribed by, say, the contractual agreement between the user of the device and the cloud server operator). In addition, we assume that the user has been allocated capacity (such as recurring time slots) until the offload has completed. These assumptions are commonly invoked (Kumar and Lu, 2010; Chen, 2015; Cao and Cai, 2018). Therefore, if T_{up} and T_{down} are the time periods needed to, upload the job to the cloud server, and, download its

results to the device, respectively, the total offloading time T_{off} is given by

$$T_{off} = T_{up} + T_{exec} + T_{down}. \tag{3.2}$$

These components are shown in Figure 3.1, where we have defined t_o to be the remote offload initiation time. It is assumed that the channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, T_{up} is a random variable, dependent on the evolution of the uplink channel state as a given upload occurs. In what follows, it is assumed that the channel state can be modelled as a homogeneous discrete-time Markov process; the same holds for T_{down}.

In order to simplify our exposition, we will initially focus on the randomness induced by the Markovian uplink channel. In the following development, we therefore temporarily assume that all offloading deadlines, job sizes (in bits), and energy costs are related only to job uploading, i.e., $T_{off} \equiv T_{up}$ and $S \equiv S_{up}$, so that $T_{exec} = T_{down} = 0$.

Since the job's hard deadline constraint must always be satisfied, we propose its simultaneous cloud server offloading (if possible and beneficial) *and* its local execution. Given the stochastic nature of the transmission channel, deciding whether and when to offload (i.e., t_o in Figure 3.1), depends on the estimation of offloading energy consumption *and* offloading time, in order to both minimize energy costs for the mobile device, *and* satisfy the job deadline constraint.[1] Depending on these estimates, there are three possibilities for offloading at time slot t_o: (i) it certainly finishes before starting the local execution of the job, and, hence, local execution never starts, or, (ii) it finishes after starting the local execution of the job, and, possibly, *before* deadline t_D; then, the fraction of local execution

[1]Note that when offloading occurs, then $t_r \leq t_o \leq t_D$, and when $t_o > t_D$, then there has been no offloading, i.e., there is only local job execution.

energy cost incurred is equal to the fraction of T_L overlapping with the offloading (i.e., local execution is terminated if a remote offload response is received), or (iii) it certainly finishes *after* deadline t_D, so it does not even start, and the total energy cost is equal to the local execution energy cost. Note that in the case of a deterministic channel, one can calculate exactly in which of these three cases the job falls. In this work, we analyze the problem of offloading with hard deadlines over a Markovian stochastic channel.

As in most of the related work references, we assume that the current state of the channel can be determined prior to making the decision to start an offload. This information can be learned in a variety of ways, such as via a short handshake with the basestation at the start of the time slot.

Figure 3.1 represents the case of continuous offloading. The job release time is t_r and its deadline is t_D. The offload begins at t_o and execution is completed $T_{up} + T_{exec} + T_{down}$ time slots later. To enforce the job deadline, local execution must begin at t_L if the mobile is still awaiting a remote response. At time $t_o + T_{up} + T_{exec} + T_{down}$, local execution is terminated provided that a remote offload response arrives before t_D.

Note that starting the local job execution at time slot t_L ensures the hard delay constraint of the task, if a remote offloading response is not received in time. Although this may result in both local and remote executions of the task, it will always satisfy the hard deadline, even if there is channel contention or extended channel outages. However, with the objective of minimizing the mean energy consumption of the mobile device, the proposed algorithm will reduce the possibility of both local and remote executions.

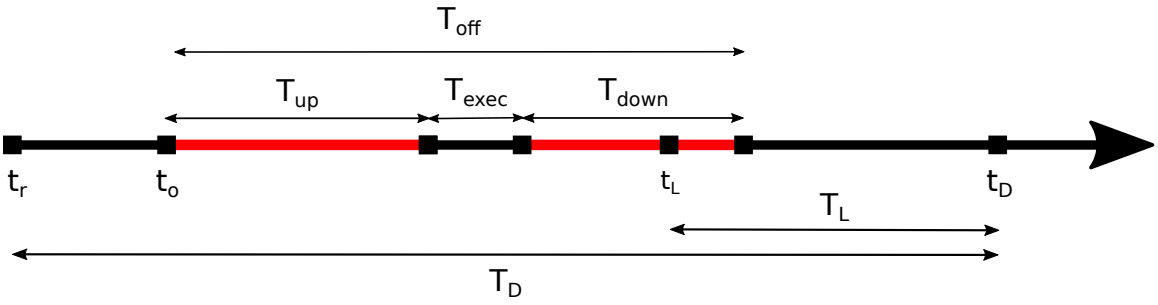

Figure 3.1: Job Computation Timing for Continuous Offloading.

Multi-Part Offloading

In multi-part offloading, the uploading data are split into two parts which are transmitted to the server sequentially. Uploading the first and second part takes T_{up_1} and T_{up_2} time slots respectively; T_W is the elapsed time between the two uploads. Upon its release, the job is assigned a server execution time T_{exec} and a resulting downloading time T_{down} (both deterministic) by the cloud server; both are communicated to the mobile device (or are prescribed by, say, the contractual agreement between the user of the device and the cloud server operator), and their total time is $T_{rest} = T_{exec} + T_{down}$. Hence, the total offloading time is $T_{off} = T_{up_1} + T_W + T_{up_2} + T_{exec} + T_{down}$, as shown in Figure 3.2 (note that t_{o_1}, t_{f_1} and t_{o_2}, t_{f_2} are the starting and finishing times of the uploading of the two job parts, respectively). It is assumed that the mobile device transmits a fixed power and uses bit rate adaptation to accommodate random variations in the uplink channel conditions. As a result, T_{up_1} and T_{up_2} are random variables, dependent on the evolution of the uplink channel state as a given upload occurs. We generalize our problem further, by assuming that T_{down} and T_{exec} are known and non-zero.

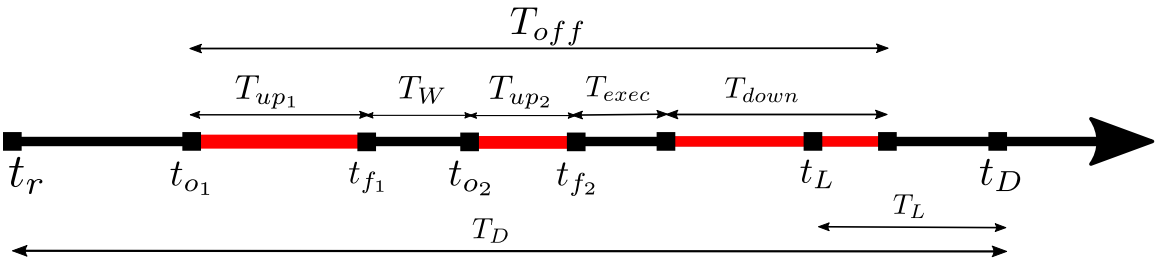

Figure 3.2: Job Computation Timing for Multi-Part Offloading.

3.2 Markovian Channel and the Time-Dilated Absorbing Markov Model

Continuous Offloading

In many studies, homogeneous Markov chains have been used to model random wireless channel conditions and as is often assumed, the Markovian transition probabilities are taken to be known, or have been learned dynamically (Gillbert, 1960; Elliott, 1963; Zhang *et al.*, 2015, 2013; Zafer and Modiano, 2007; Johnston and Krishnamurthy, 2006). Accordingly, we assume that the computation offloading occurs over a finite state Markovian channel. In this case, the OnOpt (Online Optimal) algorithm proposed in Chapter 4 is an online computation offloading algorithm that attains the minimum expected execution energy for continuous offloading. As is commonly assumed, the channel data rate is defined by the Markovian channel state and the receive signal-to-noise ratio (SNR) is such that errors due to random noise are negligible. When this is not the case, then the execution time constraint will still be satisfied by the OnOpt algorithm (Zhang *et al.*, 2015; Johnston and Krishnamurthy, 2006).

In this section we use the conventional channel state Markov chain (CSMC) to form a time-dilated absorbing Markov chain (TDAMC), which models the offloading over the

channel. The resulting Markov process is used by OnOpt in order to compute its energy and offloading time estimates, and by our analysis, in order to show its optimality. As mentioned above, we focus on T_{up}, ignoring T_{exec} and T_{down} (cf. Figure 3.1); hence, T_{off} and S below refer to T_{up} and S_{up}, respectively.

In the CSMC, and starting from the current time slot t_s, the channel conditions will evolve from one time slot to the next according to a homogeneous finite state Markov chain. We denote the set of possible channel states by $\mathcal{M}$, where $M = |\mathcal{M}|$ is the number of states in the CSMC. As discussed previously, the radio transmit power is fixed and bit rate adaptation is used to adjust to varying channel conditions. Therefore, each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded when offloading occurs in that state. In a general Markov chain model, the CSMC transition matrix is defined as $\mathbb{P} = [P_{i,j}]$, where P_{ij} is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. Unfortunately, CSMC is memoryless as far as the state of offloading and channel conditions are concerned; in order to incorporate them into our model, we form a new Markov chain, referred to as a *time-dilated absorbing Markov chain (TDAMC)*. We are again interested in the evolution of the system starting at the current time slot t_s, and running until the computation has completed, either locally or via offloading. The state of the channel in each TDAMC state at time $t \geq t_s$ is represented by X_t where $X_t \in \mathcal{M}$. However, unlike the CSMC, the TDAMC incorporates t and other information into its structure.

The TDAMC models the job offloading progress if the latter is initiated at the current time slot t_s. It is a rooted tree, constructed as follows: The root state is the channel state X_{t_s} at current time slot t_s; since this is the current time slot, X_{t_s} is known. At each subsequent time slot, the Markov chain tree branches forward, according to the transitions possible

from the current state (X_{t_s}, initially) to other CSMC states. At each step along a given tree branch, the number of job bits transmitted is determined by the bit rate associated with the channel state in question. This construction continues along each branching tree path until the number of bits offloaded reaches the job upload size, $S = S_{up}$. At that point, the state reached in the TDAMC is defined as a Markov chain *absorbing* state, i.e., it has a self-transition with probability 1. From this construction it can be seen that the TDAMC includes all possible paths that lead to a successful job offload, and that all of the states are either transient or absorbing. Eventually, all paths terminate in an absorbing state, and the energy cost of that path is proportional to its length, i.e., the number of time slots needed.

An example of a TDAMC is shown in Figure 3.3, for $t_s = 1$. It is constructed from a two-state Gilbert-Elliot channel, which is modelled by a CSMC with $\mathcal{M} = \{G, B\}$ (i.e., with "Good" and "Bad" states, respectively), and transition probabilities matrix

$$\begin{bmatrix} P_{GG} & P_{GB} \\ P_{BG} & P_{BB} \end{bmatrix},$$

i.e., $P_{1,1} = P_{GG}, P_{1,2} = P_{GB}, P_{2,1} = P_{BG}$ and $P_{2,2} = P_{BB}$. In each time slot, the TDAMC transitions to a new state in accordance with these transition probabilities. For clarity, each channel state in the figure is subscripted with its level time and the index of the subtree it belongs to. For example, $G_{3,2}$ indicates that the channel state at level $t = 3$ and subtree 2 is Good. The TDAMC shows that at $t = 3$, the channel can remain in the G state, i.e., $G_{4,2}$ or transition to the B state, i.e., $B_{4,2}$ with the given CSMC transition probabilities. Each state of the TDAMC defines the number of bits that can be offloaded during a time slot while in that state. In the example of Figure 3.3, when the channel state is G, the number of payload bits is defined by the number of bits that can be carried on the channel during a good (i.e.,

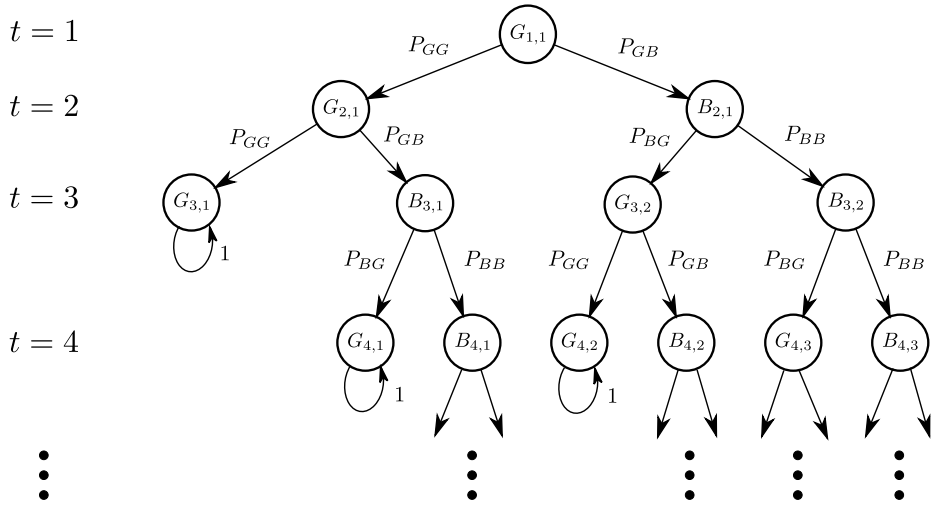

Figure 3.3: Time Dilated Absorbing Markov Chain Example

high bit-rate) channel state. In the general case, when the channel is in state X_t at time t, the number of child states at $t + 1$ is given by the number of non-zero values in the same row of the original CSMC transition matrix. In Figure 3.3, each state continues to branch downwards until the number of offloaded bits for a given branch reaches the total number needed for the offload. At that point, the branch ends in a Markov chain absorbing state discussed previously. In Figure 3.3, states $G_{3,1}$, $G_{4,1}$ and $G_{4,2}$ are absorbing states.

The non-absorbing states in the TDAMC are clearly all transient states. We define $\mathcal{A}$ to be the number of absorbing states and $\mathcal{T}$ to be the number of transient states in the TDAMC. For an absorbing Markov chain, by labeling the transient states first, the resulting transition matrix can be written in the following form (Grinstead and Snell, 2006):

$$\mathbb{P}_{\text{TDAMC}} = \begin{bmatrix} Q & R \\ \mathbf{0} & I_{\mathcal{A}} \end{bmatrix}. \tag{3.3}$$

In $\mathbb{P}_{\text{TDAMC}}$, the $\mathcal{T} \times \mathcal{T}$ sub-matrix Q contains the probabilities of transitioning between

transient states before the job upload is completed. The $\mathcal{T} \times \mathcal{A}$ sub-matrix R contains the probabilities of transitioning from a transient state to an absorbing state, indicating that the job upload is finished. $\mathbf{0}$ is an $\mathcal{A} \times \mathcal{T}$ zero matrix and $I_{\mathcal{A}}$ is an $\mathcal{A} \times \mathcal{A}$ identity (i.e., absorbing) matrix.

Q contains the entries of the original CSMC transition matrix that give the transition probabilities of each state k when it transits to a state in $\{s_k, s_k + 1, \cdots, f_k\}$, and, for our TDAMC, it has the following form:

$$Q = \begin{bmatrix} 0 & P_{1,s_1} & \cdots & P_{1,f_1} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & P_{2,s_2} & \cdots & P_{2,f_2} & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}.$$

It can be seen that Q is upper triangular, as expected, since all states are transient and can be visited at most once. The (possibly) non-zero transition probabilities shown in row one, for example, give the probability of transitioning to all possible $t = 2$ channel states and so on.

With the above construction and using results from the theory of absorbing Markov chains, various statistics can be computed by first forming the fundamental matrix

$$N = (I - Q)^{-1}. \tag{3.4}$$

For example, entry (i, j) of N gives the expected number of times that the TDAMC is in transient state j if the system is started in transient state i.

Due to the structure of our TDAMC, the computation needed in Equation (3.4) can be

greatly simplified. Note that N^{-1} is still an upper triangular matrix with all the diagonal entries equal to one, and can be decomposed as follows:

$$N^{-1} = N_{\mathcal{T}} N_{\mathcal{T}-1} N_{\mathcal{T}-2} \cdots N_1,$$

where

$$N_k = \begin{bmatrix} 1 & 0 & \cdots & 0 & n_{1,k} & \cdots & 0 \\ 0 & 1 & \cdots & 0 & n_{2,k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & n_{k-1,k} & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

N_k is an atomic triangular matrix whose inverse is given by

$$N_k^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & -n_{1,k} & \cdots & 0 \\ 0 & 1 & \cdots & 0 & -n_{2,k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -n_{k-1,k} & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Then

$$N = (N_{\mathcal{T}} N_{\mathcal{T}-1} N_{\mathcal{T}-2} \cdots N_1)^{-1} = N_1^{-1} N_2^{-1} N_3^{-1} \cdots N_{\mathcal{T}}^{-1}$$

Note that each column of the Q matrix has only one nonzero element. Therefore, N^{-1} will have only two nonzero elements in each column. Similarly, in N_k only one of the $n_{1,k}, n_{2,k}, \ldots, n_{k-1,k}$ is non-zero. Therefore, the multiplication can be done efficiently.

The absorption probabilities for all absorbing states can be obtained by

$$W = NR, \tag{3.5}$$

where W is a $\mathcal{T} \times \mathcal{A}$ matrix and $W[i, j]$ gives the probability that a particular absorbing state j will be reached if the system starts in transient state i. Using this procedure, we can thus compute the various probabilities of absorption for each absorbing state, given knowledge of the starting state. Therefore, we can obtain the probability of finishing the offload for every possible offloading time T_{off} by summing all of the absorbing state probabilities that have the same TDAMC path length. We define $P_t(T, x)$ to be the probability of offloading in exactly T time slots, when offloading starts at time t with the channel in state $X_t = x$. Then

$$P_t(T_{off}, x) = \sum_{j \in \mathcal{S}} W[x, j] \tag{3.6}$$

where $\mathcal{S}$ are all of the entries of the matrix where the offloading time is equal to T_{off}. Note that $P_t(T, x) = 0$ when it is impossible to offload in a period of exactly T time slots when offloading at t with the channel in state $X_t = x$, i.e, T is shorter (longer) than the shortest (longest) time needed to offload, under the best (worst) channel conditions. $P_t(T, x)$ is critical for computing the expected cost of offloading used by the algorithm OnOpt (cf.

Chapter 4).

The ability to compute the $P_t(T, x)$ values allows for the computation of the energy costs for both offloading and local execution. If offloading the job (of bit size S) starts at time slot t, its expected transmission energy is calculated as follows, depending on whether

- offloading is certainly completed ($1 \leq t < t_D - \frac{S}{B_{min}} + 1$), in which case the energy spent is proportional to T_{off}.
- offloading may or may not be completed within the deadline ($t_D - \frac{S}{B_{min}} + 1 \leq t \leq t_D$), in which case the energy cost is T_{off} or $t_D - t + 1$, respectively (clearly the deadline t_D is the last time slot where offloading can be done).

Noting that $P_t(T_{off}, x) = 0$ when $T_{off} < \frac{S}{B_{max}}$ or $T_{off} > \frac{S}{B_{min}}$,[2] the expected offloading energy cost when offloading starts at time slot t with the channel in state x, is given by (3.7).

Recall that local execution is postponed until the very last moment, i.e., time slot $t_L = t_D - T_L + 1$, where T_L is the number of time slots needed by the task to execute locally. A central idea of this thesis is that, although local execution is always initiated (if offloading has not completed earlier) at time t_L, in order to guarantee completion within the deadline, offloading will be decided in such a way so that it will (hopefully) terminate *before* t_D, thus saving us the energy cost of the remaining local execution. The overlap time (when such exists) between offloading at time t and local execution is $\min\{t_D + 1, t + T_{off}\} - t_L$. By recalling that E_L is the energy cost of complete local execution of the task, the local execution energy cost will be 0 if there is no overlap, or a fraction $\frac{\min\{t_D+1, t+T_{off}\}-t_L}{T_L}$ of E_L if there is. Hence, we obtain that the expected local execution cost when offloading

[2] We will assume that $\frac{S}{B_{max}}$ and $\frac{S}{B_{min}}$ are integers, to avoid burdening our formulas with ceilings $\left\lceil \frac{S}{B_{max}} \right\rceil$ and $\left\lceil \frac{S}{B_{min}} \right\rceil$.

$$E_{off}(t,x)=\begin{cases} E_{tr}\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)T_{off}, & 1\le t<t_D-\frac{S}{B_{min}}+1 \\ E_{tr}\left(\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{t_D-t} P_t(T_{off},x)T_{off}+\right. & \\ \left.\sum\limits_{T_{off}=t_D-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)(t_D-t+1)\right), & t_D-\frac{S}{B_{min}}+1\le t\le t_D \\ 0 & t>t_D \end{cases} \tag{3.7}$$

$$E_L(t,x)=\begin{cases} \sum\limits_{T_{off}=t_L-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & 1\le t<t_L \\ \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & t_L\le t\le t_D \\ E_L & t>t_D \end{cases} \tag{3.8}$$

starts at time t with the channel in state x, is given by (3.8). In the first case, there will be overlap only for $T_{off}\ge t_L-t+1$, while in the second there is always overlap, since $t-t_L+T_{off}>0$.

Note that the above development was presented by taking into account only the random job uploading process. These results are easily extended to include both the (deterministic) cloud execution, i.e., T_{exec} and a Markovian random downlink channel, i.e., $T_{off}=T_{up}+T_{exec}+T_{down}$ and $S=S_{up}+S_{down}$. This is done as follows. The TDAMC of Figure 3.3, which models the uploading of S_{up} bits, is extended by branching out from each (previously) absorbing state for T_{exec} transition steps. This is followed by branching out according to a process similar to the TDAMC of Figure 3.3, which then models

the downloading of S_{down} bits. The resulting Markov process therefore tracks the channel throughout all three offloading periods, i.e., upload, remote execution, and download, shown in Figure 3.1.

Multi-Part Offloading

As is the case for continuous offloading, we assume that there is a known channel state Markov chain (CSMC), i.e., the channel conditions evolve from one time slot to the next according to a homogeneous finite state Markov chain. We consider the tree-like Markov chain produced by following the evolution of the channel, starting from an initial state at time $t = 1$, and branching out from each state according to the transition probabilities of the CSMC. This new Markov chain is referred to as a *time-dilated absorbing Markov chain (TDAMC)*. We will denote by X_t a state in this Markov chain, reached after running the channel for t time slots. We will consider subtrees of this TDAMC (such at $TDAMC_1$ and $TDAMC_2$ below), endowed with energy costs and absorbing states.

The part of the TDAMC which models the offloading progress if the uploading of S_{up_1} is initiated at a time slot t_s, will be denoted as $TDAMC_1$. An example of $TDAMC_1$ is shown in Figure 3.4: It is constructed from a two-state Gilbert-Elliot channel, which is modelled by a CSMC with two states $\{G, B\}$ (i.e., a "Good" one with the higher bit rate, and a "Bad" one, respectively), and with transition probabilities $P_{GG}, P_{GB}, P_{BG}, P_{BB}$. In each time slot, $TDAMC_1$ transitions to a new state in accordance with these transition probabilities. For clarity, each state s_t^a in the figure is subscripted by its time slot t, and superscripted by a unique identifier a that distinguishes it from the other channel states reachable after t time slots. Hence, the $TDAMC_1$ of Figure 3.4 models the offloading process initiated at time slot t_s, when the channel state that has been reached at that time is

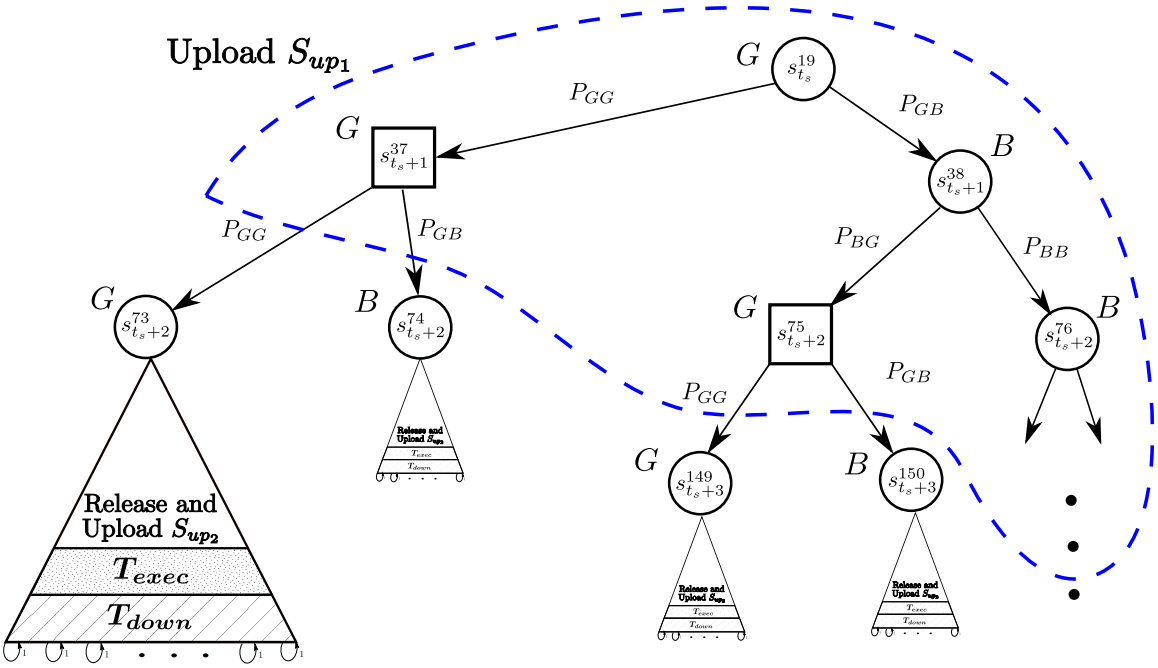

Figure 3.4: $TDAMC_1$ when offloading S_{up_1} starts at time t_s.

$s^{19}_{t_s}$. The bit rate at each state is also indicated.

In general, $TDAMC_1$ is a rooted subtree of the TDAMC, constructed as follows: The root state is the (known) channel state X_{t_s} at current time slot t_s. At each subsequent time slot, the Markov chain tree branches forward, according to the transitions possible from the current state (X_{t_s}, initially) to other TDAMC states. At each state, the number of job bits transmitted is determined by the bit rate associated with that state. The branching continues to create all possible paths of states needed to upload S_{up_1} bits, up to some state $X_{t_{f_1}}$ corresponding to upload finishing time t_{f_1} for each path from the root. (such as $s^{37}_{t_s+1}$, $s^{75}_{t_s+2}$, represented by squares in Figure 3.4). At time $t_{f_1}+1$, the second part S_{up_2} is released. Continuing the branching of the TDAMC, and after a possible waiting period, the uploading of S_{up_2} commences, followed by the job execution in the cloud in time T_{exec}, and the downloading of the results in time T_{down}, ending in an absorbing state (this part of the offloading

is depicted in Figure 3.4 as subtrees hanging from states $s^{73}_{t_s+2}, s^{74}_{t_s+2}, s^{149}_{t_s+3}, s^{150}_{t_s+3}$). The optimal waiting time for each path, i.e., the waiting times which optimize the total (over all paths) expected energy cost for uploading S_{up_2}, is solved in Chapter 4. Then the energy cost of each subtree is the optimal expected (over all paths) cost of completing offloading, when uploading S_{up1} finishes in time slot t_{f_1} and state $X_{t_{f_1}}$. In fact, $TDAMC_1$ does not need to extend all the way into these subtrees, but treats states $X_{t_{f_1}+1}$ as absorbing states, each with cost equal to the energy cost of its subtree.

Similarly to continuous offloading, the probability of uploading S_{up_1} in T_{up_1} time slots, starting at time slot t_{o_1}, and a state $X_{t_{o_1}}$, can be calculated by building $TDAMC_1$, with a set of absorbing states $\mathcal{A}$, and a set of transient states $\mathcal{T}$. Then, the transition matrix can be written (Grinstead and Snell, 2006) as

$$\mathbb{P} = \begin{bmatrix} Q & R \\ \mathbf{0} & I_{\mathcal{A}} \end{bmatrix}, \tag{3.9}$$

where the $|\mathcal{T}| \times |\mathcal{T}|$ sub-matrix Q contains the probabilities of transitioning between transient states, the $|\mathcal{T}| \times |\mathcal{A}|$ sub-matrix R contains the probabilities of transitioning from a transient state to an absorbing state, and $I_{\mathcal{A}}$ is an $|\mathcal{A}| \times |\mathcal{A}|$ identity matrix.

The theory of absorbing Markov chains implies that various statistics can be computed by forming the fundamental matrix $N = (I - Q)^{-1}$, where $N[i, j]$ gives the expected number of times that $TDAMC_1$ is in transient state j if the system is started in transient state i. Given the structure of $TDAMC_1$, N can be easily decomposed and calculated as in continuous offloading, since the particular structure of matrices Q, N, N^{-1} is the same simple one as in continuous offloading. The absorption probabilities matrix W_1 for all

absorbing states is given by

$$W_1 = NR, \tag{3.10}$$

where W_1 is a $|\mathcal{T}| \times |\mathcal{A}|$ matrix, and $W_1[i,j]$ gives the probability that absorbing state j will be reached when starting in transient state i. Therefore, the probability of uploading the first part with size S_{up_1} in T_{up_1} time slots, starting at time t_{o_1} and state $X_{t_{o_1}}$, is

$$P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) = \textstyle\sum_{j \in \mathcal{S}_1} W_1[X_{t_{o_1}}, j], \tag{3.11}$$

where $\mathcal{S}_1$ is the set of absorbing states in $TDAMC_1$ reached by a path of length $T_{up_1}+1$ from the root $X_{t_{o_1}}$.

Similarly to $TDAMC_1$, and in order to calculate the expected cost once the uploading of S_{up_2} commences at time slot t_{o_2}, we construct $TDAMC_2$, which tracks the offloading process from t_{o_2} and state $X_{t_{o_2}}$ until offloading is completed. Just like above, the probability of uploading S_{up_2} in T_{up_2} time slots, starting at time t_{o_2} and state $X_{t_{o_2}}$, is

$$P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) = \textstyle\sum_{j \in \mathcal{S}_2} W_2[X_{t_{o_2}}, j] \tag{3.12}$$

where $\mathcal{S}_2$ is the set of absorbing states in $TDAMC_2$ reached by a path of length $T_{up_2}+1$ from the root $X_{t_{o_2}}$.

If the uploading of S_{up_1} starts at time slot t_{o_1}, and after noting that $P_{t_{o_1}}(S_{up_1}, T_{up_1}, x) = 0$ when $T_{up_1} < \frac{S_{up_1}}{B_{max}}$ or $T_{up_1} > \frac{S_{up_1}}{B_{min}}$, the expected offloading energy cost when offloading starts at time slot t_{o_1} in state $X_{t_{o_1}}$, is given by equation (3.13), and the expected local execution cost is given by (3.14), where E_{tr} is the transmission energy of the mobile device during one time slot.

$$E_{off_1}(S_{up_1}, X_{t_{o_1}}) =$$
$$\begin{cases} E_{tr} \sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) T_{up_1}, & 1 \leq t_{o_1} < t_D - \frac{S_{up_1}}{B_{min}} + 1 \\ E_{tr} \left(\sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{t_D - t_{o_1}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) T_{up_1} + \right. & \\ \left. \sum_{T_{up_1}=t_D-t_{o_1}+1}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}})(t_D - t_{o_1} + 1) \right), & t_D - \frac{S_{up_1}}{B_{min}} + 1 \leq t_{o_1} \leq t_D \end{cases} \tag{3.13}$$

$$E_{L_1}(S_{up_1}, X_{t_{o_1}}) =$$
$$\begin{cases} \sum_{T_{up_1}=t_L-t_{o_1}+1}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) \left(\frac{\min\{t_D+1, t_{o_1}+T_{up_1}\}-t_L}{T_L} E_L \right), & 1 \leq t_{o_1} < t_L \\ \sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) \left(\frac{\min\{t_D+1, t_{o_1}+T_{up_1}\}-t_L}{T_L} E_L \right), & t_L \leq t_{o_1} \leq t_D \end{cases} \tag{3.14}$$

The expected energy cost of uploading S_{up_2} in exactly T_{up_2} time slots, and downloading the results in exactly T_{down} time slots, starting at time t_{o_2} with the channel TDAMC in state $X_{t_{o_2}}$, is given by equation (3.15), where E_{rc} is the energy consumption of the mobile device during one time slot when receiving from the server.

$$\hat{E}_{off_2}(S_{up_2}, T_{up_2}, t_{o_2}) =$$

$$\begin{cases} E_{tr}T_{up_2} + E_{rc}T_{down}, & t_{f_1} < t_{o_2} \leq t_D - T_{up_2} - T_{rest} \\ E_{tr}T_{up_2} + E_{rc}\{t_D - (t_{o_2} + T_{up_2} + T_{exec}) + 1\}, & t_D - T_{rest} < t_{o_2} + T_{up_2} \leq t_D - T_{exec} \\ E_{tr}T_{up_2}, & t_D - T_{exec} < t_{o_2} + T_{up_2} \leq t_D \\ E_{tr}(t_D - t_{o_2} + 1), & t_D < t_{o_2} + T_{up_2} \leq t_D + T_{up_2} \end{cases} \tag{3.15}$$

Then the expected offloading energy cost is

$$E_{off_2}(S_{up_2}, X_{t_{o_2}}) = \sum_{T_{up_2}=\frac{S_{up_2}}{B_{max}}}^{\frac{S_{up_2}}{B_{min}}} P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) \hat{E}_{off_2}(S_{up_2}, T_{up_2}, t_{o_2}). \tag{3.16}$$

Given the finishing time t_{f_1} of uploading S_{up_1}, the local execution energy cost corresponding to the offloading portion, starting with the uploading of S_{up_2} at time t_{o_2} and state $X_{t_{o_2}}$, taking exactly T_{up_2} time slots, and finishing with the downloading of the results, is given in (3.17).

$$\hat{E}_{L_2}(T_{up_2}, t_{f_1}, t_{o_2}) =$$

$$\begin{cases} 0, & t_{f_1} < t_{o_2} < t_L - T_{up_2} - T_{rest} \\ \frac{t_{o_2}+T_{up_2}+T_{rest}-t_L}{T_L} E_L, & t_{f_1} < t_L \wedge t_L - T_{up_2} - T_{rest} \leq t_{o_2} \leq t_D - T_{up_2} - T_{rest} \\ E_L, & t_{f_1} < t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \leq t_D \\ \frac{t_{o_2}+T_{up_2}+T_{rest}-t_{f_1}}{T_L} E_L, & t_{f_1} \geq t_L \wedge t_{f_1} < t_{o_2} \leq t_D - T_{up_2} - T_{rest} \\ \frac{t_D - t_{f_1}}{T_L} E_L, & t_{f_1} \geq t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \leq t_D \end{cases} \tag{3.17}$$

Then the expected local execution energy cost is

$$E_{L_2}(S_{up_2}, t_{f_1}, X_{t_{o_2}}) = \sum_{T_{up_2}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) \hat{E}_{L_2}(T_{up_2}, t_{f_1}, t_{o_2}) \tag{3.18}$$

Note that $E_{off_1} = 0, E_{L_1} = E_L$ for $t_{o_1} \geq t_D + 1$, and $E_{off_2} = 0, E_{L_2} = E_L$ for $t_{o_2} \geq t_D + 1$, i.e., when the first or second part isn't uploaded, respectively.

3.3 Offline Bound

In this section, an offline lower bound on mobile device energy is derived. This bound is used in Chapter 6 for performance comparisons with various online computation offloading algorithms. Since the bound is offline, we assume that the wireless channel states are known

for all future time slots. When a job is released, the bound then chooses the job offload time so that its deadline is met and the energy needed is minimized.

Continuous Offloading

Let t_o be the time to start offloading, given that we know the bit rate B_t (in bits per time slot) at all times $1 \leq t \leq t_D$ (recall that t_r is taken to be 1). Let $t_f(t_o)$ be defined as the offload finishing time when offloading starts at t_o. Then t_o can be found by solving the following IP.

$$\min_{t_o} \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \tag{3.19}$$

$$s.t. \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \leq E_L \tag{3.20}$$

$$1 \leq t_o \leq t_D. \tag{3.21}$$

Objective (3.19) consists of two terms. The first is the local execution energy cost incurred before offloading starts. If $t_o < t_L$, this term is zero, which means that there has been no local execution to that point; otherwise, $\frac{t_o - t_L}{T_L} E_L$ is the energy that has been expended by local execution energy before t_o. The second term in (3.19) is the total energy consumption after offloading starts where e_t is the energy expended in time slot t. When $t_o < t < t_L$, each e_t includes only the offloading energy; and when $t \geq t_L$, both offloading and local execution are performed at time slot t. Therefore, e_t is given as

$$e_t = \begin{cases} E_{tr}, & t < t_L \\ E_{tr} + \frac{E_L}{T_L}, & t \geq t_L \end{cases} \tag{3.22}$$

where E_{tr} is the energy cost per time slot for transmitting on the channel. Constraint (3.20) ensures that the energy used in offloading does not exceed that of executing the job locally. Note that if the IP is infeasible, then there is no feasible offloading start time t_o, i.e., it is best to execute locally without offloading.

Multi-Part Offloading

In this case case, we assumed that we split the uploading data into two parts. For this scenario, we also assumed that the downloading time of the results $T_{down} > 0$. Let t_{o_1} be the time to start offloading, given that we know the bit rate B_t (in bits per time slot) at all times $1 \leq t \leq t_D$ (recall that t_r is taken to be 1). Let $t_{f_1}(t_{o_1})$ and $t_{f_2}(t_{o_2})$ be defined as the offload finishing time when offloading starts at t_{o_1} for the first part and t_{o_2} for the second part, respectively. E_{tr} is the energy cost per time slot for transmitting on the channel. t_{o_1} and t_{o_2} can be found by solving the following IP.

$$\min_{t_{o_1}, t_{o_2}} \quad \frac{\max(t_{f_2}(t_{o_2}) + T_{rest}, t_L) - t_L}{T_L} E_L + (t_{f_1}(t_{o_1}) - t_{o_1}) E_{tr} + (t_{f_2}(t_{o_2}) - t_{o_2}) E_{tr} + T_{down} E_{rc} \tag{3.23}$$

$$s.t. \quad \frac{\max(t_{f_2}(t_{o_2}), t_L) + T_{rest} - t_L}{T_L} E_L + (t_{f_1}(t_{o_1}) - t_{o_1}) E_{tr} + (t_{f_2}(t_{o_2}) - t_{o_2}) E_{tr} + T_{down} E_{rc} \leq E_L \tag{3.24}$$

$$1 \leq t_{o_1} \leq t_D \tag{3.25}$$

$$t_{f_1}(t_{o_1}) < t_{o_2} \leq t_D. \tag{3.26}$$

Objective (3.23) consists of four terms. The first term is the local execution energy cost incurred for the whole offloading of two parts. If $t_{f_2}(t_{o_2}) + T_{rest} < t_L$, first term is zero,

which means that there has been no local execution by the end of offloading the second part; otherwise, $\frac{t_{f_2}(t_{o_2})+T_{rest}-t_L}{T_L}E_L$ is the energy that has been expended by local execution energy. The second and third term in (3.23) are the energy cost of the transmission for uploading the first and second part, respectively. The fourth term is the energy incurred for downloading the results. Constraint (3.24) ensures that the energy used in offloading does not exceed that of executing the job locally. Note that if the IP is infeasible, then there are no feasible offloading start times t_{o_1} and t_{o_2}, i.e., it is best to execute locally without offloading.

Chapter 4

Optimal Offloading Starting Time

In this Chapter, we use the time-dilated absorbing Markov model construction of Section 3.2 and the theory of optimal stopping for Markov decision processes (Peskir and Shiryaev, 2006) to define the OnOpt and MultiOpt algorithms, and show that they achieve the optimal expected energy consumption for offloading.

4.1 OnOpt (Online Optimal) Algorithm

A high-level description of the algorithm is as follows: At each time slot t (starting from the job release time slot), the algorithm considers the TDAMC model for starting offloading at current time $t_s = t$. It computes (based on the TDAMC) the optimal offloading starting time $\tau^*_{t_D} \geq t$, by formulating the problem as a Markovian optimal stopping problem. If $\tau^*_{t_D} = t$, then offloading is started immediately at time t. Otherwise, the algorithm waits till time slot $t + 1$, to repeat the above process.

Suppose that the current time slot is t_s, and consider the corresponding TDAMC rooted at state X_{t_s}. In order to compute the optimal time slot for starting offloading (if offloading

turns out to be more beneficial, in expectation, than executing the task solely locally), we need to compute the *offloading starting time* $\tau^*_{t_D}$ that satisfies the following optimization problem:

$$\begin{aligned} v_{t_D}(y) &= \min_{t:t_s\le t\le t_D+1} E[g_t(X_t)|X_{t_s}=y] \\ &= \min_{t:t_s\le t\le t_D+1} \sum_{z\in\mathcal{M}} Pr[X_t=z|X_{t_s}=y]g_t(z), \end{aligned} \tag{4.1}$$

where X_{t_s} is the current channel state, and $g_t(x)$ is the expected total energy cost if offloading starts at time slot t with channel state $X_t = x$. The choice of $t = t_D + 1$ in (4.1) corresponds to no offloading, in which case (3.7) and (3.8) imply a total cost of E_L. Then, for $t_s \le t \le t_D$,

$$g_t(x) = E_{off}(t,x) + E_L(t,x), \tag{4.2}$$

where $E_{off}(t,x), E_L(t,x)$ are the expected offloading and local execution costs, respectively, as defined in (3.7) and (3.8), when offloading starts at time t with the channel in state $X_t = x$.

The optimization problem (4.1) is inherently an off-line problem, while the algorithm we would like to use is inherently an on-line one, in the sense that at every time slot it has to decide whether to offload or not, *given the history of channel states it has encountered so far.* Such an algorithm is defined by the following recursion, which can be solved using Dynamic Programming (DP), i.e.,

$$V_t(x) = \begin{cases} E_L, & t \ge t_D \\ \min\{g_t(x), E[V_{t+1}|X_t=x]\}, & t = t_s,\cdots,t_D-1 \end{cases} \tag{4.3}$$

Note that $V_t(x)$ is the minimum between the expected total cost of offloading at the current time slot t, and the expected cost of postponing that decision to time slot $t+1$, given that the channel state at time t is x, and $E[V_{t+1}|X_t = x]$ is the expectation of $V_{t+1}(X_{t+1})$ over all possible X_{t+1}, under the condition that $X_t = x$, i.e.,

$$E[V_{t+1}|X_t = x] = \sum_{y \in \mathcal{M}} Pr[X_{t+1} = y|X_t = x]V_{t+1}(y).$$

Note that (4.3) implies a *policy*, that dictates whether at any time t and state X_t the algorithm should start uploading (if the $\min$ is attained by g_t), or should otherwise wait.

It is well known (e.g., Theorem 1.7 in (Peskir and Shiryaev, 2006)) that policy V_t in (4.3) is *optimal*, i.e., it solves the original problem (4.1), since

$$v_{t_D}(y) = V_{t_s}(y), \ \forall y. \tag{4.4}$$

Therefore, the following lemma holds:

Lemma 1. (Peskir and Shiryaev, 2006) *The optimal offloading starting time for* (4.1) *is* $\tau^*_{t_D} = \arg\min_{t_s \le t \le t_D+1}\{V_t(x) = g_t(x)\}$.

Lemma 3 implies that the on-line algorithm OnOpt, given in Algorithm 2, is optimal. Note that this result is true for any Markovian channel. The algorithm is given the local execution starting time t_L, local execution energy E_L, job deadline t_D, and job size S. It then arranges for the job to be executed either locally or by remote offloading (or both, if needed). Initially, the remote offload is disabled by setting t_o to a value greater than t_D in Line 1. At each time slot t_s with the channel at state $X_{t_s} = x$, we test if $t_s < t_o$, i.e., no offload has been initiated for the job. Then both $g_{t_s}(x)$ and $E[V_{t_s+1}|X_{t_s} = x]$ are computed (using (4.2) and using DP to solve (4.3), respectively). If $g_{t_s}(x) \le E[V_{t_s+1}|X_{t_s} = x]$, then

the offload begins at time t_s, i.e., $t_o = t_s$, since in this case $\tau^*_{t_D} = t_s$ from Lemma 3. If offloading finishes before a local execution finishes, then local execution is terminated (Line 11). At Line 13 we check to see if local execution should start so that the job's deadline can be guaranteed. Similarly, Line 16 tests if the local job has completed. In that case, any remote offload in progress will be terminated.

Algorithm 1 OnOpt (Online Optimal) Algorithm

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job size S.

1: $t_o := \infty$ ▷ Offloading initially disabled (t_o is offload start time)
2: **for all** $t_s \in \{1, \ldots, t_D\}$ **do**
3: **if** $t_s < t_o$ **then**
4: $\overline{c_{t_s}} := g_{t_s}(x)$ ▷ Expected energy cost of offloading at t_s.
5: $\overline{c_{t_s+1}} := E[V_{t_s+1}|X_{t_s} = x]$ ▷ Expected energy cost of waiting until $t_s + 1$.
6: **if** $\overline{c_{t_s}} \leq \overline{c_{t_s+1}}$ **then**
7: $t_o := t_s$ ▷ Start offloading.
8: **end if**
9: **else if** offloading terminates at t_s **then**
10: Abort local execution (if active). ▷ Remote offload response has been received.
11: **return**
12: **end if**
13: **if** $t_s = t_D - T_L + 1$ **then**
14: Start local execution. ▷ Ensure that the job deadline is satisfied.
15: **end if**
16: **if** $t_s = t_D$ **then**
17: Abort remote offload (if active). ▷ Local execution has completed.
18: **return**
19: **end if**
20: **end for**

4.2 Optimality Proof of OnOpt

Here we are going to proof the correctness of the equation (4.4)

$$v_{t_D}(x) = V_{t_s}(x)$$

where,

$v_{t_D}(x)$ is the value function with the time horizon t_D which is the deadline of the job (Wang, 2006).

Consider the process:

$$Z_t = \sum_{j=t_s}^{t-1} c_j(X_j) + V_t(X_t)\ ; \qquad t = t_s, t_s+1, \ldots, t_D \tag{4.5}$$

Z_t is the accumulated total expected cost we incur if we postpone our offloading decision to time t. Note that $c_j(X_j)$ is cost of waiting which in our case is zero.

Here we proof that, the process $\{Z_t : t = t_s, t_s+1, \ldots, t_D\}$ is a sub-martingle. Indeed,

1. $E[Z_{t+1}|X_t, X_{t-1}, \ldots, X_{t_s}] = \sum_{j=t_s}^{t} c_j(X_j) + E[V_{t+1}(X_{t+1})|X_t = x] =$
   $\sum_{j=t_s}^{t-1} c_j(X_j) + c_t(X_t) + E[V_{t+1}(X_{t+1})|X_t = x]$
2. $V_t(X_t) = min\{g_t(X_t), c_t(x) + E[V_{t+1}(X_{t+1})|X_t = x]\}$

From 1 and 2, we have: $E[Z_{t+1}|X_t, X_{t-1}, \ldots, X_{t_s}] \geq \sum_{j=t_s}^{t-1} c_j(X_j) + V_t(X_t) = Z_t$

Now, If we get expectation of the inequality of the result we will have:

$$E[Z_{t+1}] \geq E[Z_t] \geq E[Z_{t-1}] \geq \cdots \geq E[Z_{t_s}]$$
$$\Longrightarrow E[Z_\tau] \geq E[Z_{t_s}] = V_{t_s}(x)$$

For any stopping times (τ) taking values in $\{t_s, t_s + 1, \ldots, t_D\}$.

But By definition, we know that $V_j(X_j) \leq g_j(X_j)$ for any j, thus:

$$V_{t_s}(x) \leq E[Z_\tau] \leq E[\sum_{j=t_s}^{\tau-1} c_j(X_j) + g_\tau(X_\tau)]$$

We define:

$$Z_{\tau^*_{t_D} \wedge t} = 1_{\{\tau^*_{t_D} \leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D} \geq (t+1)\}} Z_t$$

The definition of $1_{\{condition\}}$ is:

$$1_{\{condition\}} = \begin{cases} 1, & \text{if condition is true} \\ 0, & \text{if condition is false} \end{cases}$$

Actually, $Z_{\tau^*_{t_D} \wedge t}$ is the total cost we incur at time equals to t, if we start offloading at time equals to $\tau^*_{t_D}$.

For every $t = t_s, t_s + 1, \ldots, t_D - 1$, we have:

$$Z_{\tau^*_{t_D}\wedge(t+1)} = 1_{\{\tau^*_{t_D}\leq(t+1)\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+2)\}} Z_{t+1} =$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}=(t+1)\}} Z_{t+1} + 1_{\{\tau^*_{t_D}\geq(t+2)\}} Z_{t+1}$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} Z_{t+1}$$

If we get expectation of $Z_{\tau^*_{t_D}\wedge(t+1)}$, we will have:

$$E[Z_{\tau^*_{t_D}\wedge(t+1)}|X_t, X_{t-1}, \ldots, X_{t_s}] =$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} E[Z_{t+1}|X_t, X_{t-1}, \ldots, X_{t_s}]$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} \left[\sum_{j=t_s}^{t} c_j(X_j) + E[V_{t+1}(X_{t+1})|X_t = x] \right]$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} \left[\sum_{j=t_s}^{t-1} c_j(X_j) + c_t(X_t) + E[V_{t+1}(X_{t+1})|X_t = x] \right]$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} \left[\sum_{j=t_s}^{t-1} c_j(X_j) + V_t(X_t) \right]$$

$$= 1_{\{\tau^*_{t_D}\leq t\}} Z_{\tau^*_{t_D}} + 1_{\{\tau^*_{t_D}\geq(t+1)\}} Z_t = Z_{\tau^*_{t_D}\wedge t}$$

Therefore, we can get to following result by simply getting expectation of expectation,

$$E[Z_{\tau^*_{t_D}\wedge t_s}] = E[Z_{\tau^*_{t_D}\wedge t_s+1}] = E[Z_{\tau^*_{t_D}\wedge t_s+2}] = \cdots = E[Z_{\tau^*_{t_D}\wedge \tau^*_{t_D}}] = E[Z_{\tau^*_{t_D}\wedge t_D}]$$

Moreover, we know:

$$E[Z_{\tau^*_{t_D}\wedge t_s}] = E[Z_{t_s}]$$
$$E[Z_{\tau^*_{t_D}\wedge \tau^*_{t_D}}] = E[Z_{\tau^*_{t_D}}]$$

Therefore,

$$E[Z_{t_s}] = E[Z_{\tau^*_{t_D}}] = E\left[\sum_{j=t_s}^{\tau^*_{t_D}-1} c_j(X_j) + V_{\tau^*_{t_D}}(X_{\tau^*_{t_D}})\right] = E\left[\sum_{j=t_s}^{\tau^*_{t_D}-1} c_j(X_j) + g_{\tau^*_{t_D}}(X_{\tau^*_{t_D}})\right]$$

$$= v_{t_D}(x)$$

Moreover, we know:

$$V_{t_s}(x) = E[Z_{t_s}]$$

Therefore,

$$V_{t_s}(x) = v_{t_D}(x).$$

Here the proof ends.

4.3 MultiOpt (Multi-Decision online Optimal) Algorithm

A high-level description of the algorithm is as follows: Starting from time slot $t = 1$ (the release time of the job), at each time slot t the algorithm considers $TDAMC_1$ in order to determine the expected cost of the whole offloading process if uploading S_{up_1} commences at the current time t. If that cost is less than the expected offloading cost when the algorithm waits one more time slot, then $t^*_{o_1} = t$ (offloading S_{up_1} commences), otherwise the algorithm postpones its decision to time slot $t + 1$. Once the uploading of S_{up_1} finishes, the algorithm repeats the same decision process at every time slot (using $TDAMC_2$ to compute expected costs), to determine the time $t^*_{o_2}$ of starting uploading S_{up_2}.

MultiOpt will be optimal only if its first decision $t^*_{o_1} \geq t$, i.e., its starting time of uploading S_{up_1}, coincides with the solution of the following minimization problem (where the choice $t_{o_1} = t_D + 1$ corresponds to no uploading):

$$v_1(X_t) = \min_{t \leq t_{o_1} \leq t_D+1} \left\{ \sum_{X_{t_{o_1}} \in \mathcal{S}_1} Pr[X_{t_{o_1}} | X_t] \Bigg(E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}}) + \sum_{X_{t_{f_1}+1} \in \mathcal{S}_2} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}] v_2(t_{f_1}, X_{t_{f_1}+1}) \Bigg) \right\} \tag{4.6}$$

where $\mathcal{S}_1$ is the set of states reachable after running the channel for t_{o_1} time slots, $\mathcal{S}_2$ is the set of absorbing states of $TDAMC_1$ rooted at $X_{t_{o_1}}$, and $v_2(t_{f_1}, X_{t_{f_1}+1})$ is the optimal expected energy cost for the rest of the offloading, when S_{up_1} finished uploading at time t_{f_1}, i.e., the cost of the absorbing state $X_{t_{f_1}+1}$ of $TDAMC_1$ (or, equivalently, the corresponding subtree of Figure 3.4). This optimal cost is the solution of the following optimization

problem for $t > t_{f_1}$, when we set $X_t := X_{f_1+1}$:

$$\begin{aligned} v_2(t_{f_1}, X_t) &= \min_{t \le t_{o_2} \le t_D+1} E[g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})|X_t] \\ &= \min_{t \le t_{o_2} \le t_D+1} \sum_{X_{t_{o_2}} \in \mathcal{T}_1} Pr[X_{t_{o_2}}|X_t] g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}), \end{aligned} \tag{4.7}$$

where $\mathcal{T}_1$ is the set of states reachable after running the channel for t_{o_1} time slots, and $g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})$ is the expected energy cost of uploading S_{up_2} and downloading the results, if uploading of S_{up_1} finishes at t_{f_1} and uploading S_{up_2} starts at time slot t_{o_2} and state $X_{t_{o_2}}$, i.e.,

$$g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}) = E_{off_2}(S_{up_2}, X_{t_{o_2}}) + E_{L_2}(S_{up_2}, t_{f_1}, X_{t_{o_2}}). \tag{4.8}$$

For $t > t_D$, $v_2(t_{f_1}, X_t) = 0$ (no uploading of the second part).

Given the first decision $t^*_{o_1}$ of MultiOpt, we show that its second decision $t^*_{o_2}$ solves the optimization problem (4.7). For every time slot $t_{o_2} > t_{f_1}$ and state $X_{t_{o_2}}$, we define the expected cost $V_2(t_{f_1}, X_{t_{o_2}})$ recursively as follows:

$$V_2(t_{f_1}, X_{t_{o_2}}) = \begin{cases} 0, & t_{o_2} > t_{f_1} \ge t_D \\ E_L - \frac{\max\{t_{f_1}, t_L\} - t_L}{T_L} E_L, & t_{o_2} \ge t_D > t_{f_1} \\ \min\{g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}), E[V_2(t_{f_1}, X_{t_{o_2}+1})|X_{t_{o_2}}]\}, & t_D > t_{o_2}. \end{cases} \tag{4.9}$$

$V_2(t_{f_1}, X_{t_{o_2}})$ can be computed using Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_2} at time slot t_{o_2} and state $X_{t_{o_2}}$,

and the expected cost of postponing that decision to time slot $t_{o_2}+1$

$$E[V_2(t_{f_1}, X_{t_{o_2}+1})|X_{t_{o_2}}] = \sum_{X_{t_{o_2}+1}\in\mathcal{T}_2} Pr[X_{t_{o_2}+1}|X_{t_{o_2}}]V_2(t_{f_1}, X_{t_{o_2}+1}), \tag{4.10}$$

where $\mathcal{T}_2$ is the set of states reachable after running the channel for $t_{o_2}+1$ time slots. Note that (4.9) implies a *policy*, that dictates whether at any time t_{o_2} and state $X_{t_{0_2}}$ the algorithm should start uploading (if the $\min$ is attained by g_2), or should otherwise wait. It is well known (e.g., Theorem 1.7 in (Peskir and Shiryaev, 2006)) that policy V_2 is *optimal*, i.e., solves the original problem (4.7), since

$$v_2(t_{f_1}, X_t) = V_2(t_{f_1}, X_t),\ \forall t > t_{f_1}, X_t. \tag{4.11}$$

Hence the following holds:

Lemma 2. (Peskir and Shiryaev, 2006) *The optimal time for starting uploading S_{up_2} is* $t^*_{o_2} = \arg\min_{t_{f_1}<t_{o_2}\leq t_D}\{V_2(t_{f_1}, X_{t_{o_2}}) = g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})\}$.

It remains to prove that the first decision $t^*_{o_1}$ of MultiOpt is also optimal. For any possible choice t_{o_1} for the first decision of MultiOpt, (4.11) can be applied, and the optimization problem (4.6) becomes

$$v_1(X_t) = \min_{t\leq t_{o_1}\leq t_D+1}\left\{\sum_{X_{t_{o_1}}\in\mathcal{S}_1} Pr[X_{t_{o_1}}|X_t]\Bigg(E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}})+ \sum_{X_{t_{f_1}+1}\in\mathcal{S}_1} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}]V_2(t_{f_1}, X_{t_{f_1}+1})\Bigg)\right\} \tag{4.12}$$

The expected energy cost of offloading when starting uploading S_{up_1} at time t_{o_1} and state $X_{t_{o_1}}$ is

$$g_1(S_{up_1}, X_{t_{o_1}}) = E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}}) + \sum_{X_{t_{f_1}+1} \in \mathcal{S}_1} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}] V_2(t_{f_1}, X_{t_{f_1}+1}). \tag{4.13}$$

For every time slot t_{o_1} and state $X_{t_{o_1}}$, we define the expected cost $V_1(X_{t_{o_1}})$ recursively as follows:

$$V_1(X_{t_{o_1}}) = \begin{cases} E_L, & t_{o_1} \geq t_D \\ \min\left\{ g_1(S_{up_1}, X_{t_{o_1}}), E[V_1(X_{t_{o_1}+1})|X_{t_{o_1}}] \right\}, & t_{o_1} = 1, \ldots, t_D - 1 \end{cases} \tag{4.14}$$

$V_1(X_{t_{o_1}})$ can be computed using Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_1} at time slot t_{o_1} and state $X_{t_{0_1}}$, and the expected cost of postponing that decision to time slot $t_{o_1} + 1$

$$E[V_1(X_{t_{o_1}+1})|X_{t_{o_1}}] = \sum_{X_{t_{o_1}+1} \in \mathcal{S}_3} Pr[X_{t_{o_1}+1}|X_{t_{o_1}}] V_1(X_{t_{o_1}+1}),$$

where $\mathcal{S}_3$ is the set of states reachable after running the channel for $t_{o_1} + 1$ time slots.

In exactly the same way as Lemma 2, one can show that policy V_1 is also *optimal*, i.e., solves the original problem (4.6), since $v_1(X_t) = V_1(X_t),\ \forall t, X_t$. Hence the following holds:

Lemma 3. (Peskir and Shiryaev, 2006) *The optimal time for starting uploading S_{up_1} is* $t^*_{o_1} = \arg\min_{1 \leq t_{o_1} \leq t_D} \{V_1(X_{t_{o_1}}) = g_1(S_{up_1}, X_{t_{o_1}})\}$.

Lemmata 2 and 3 imply that the on-line algorithm MultiOpt, given in Algorithm 2, is optimal. Note that this result is true for any Markovian channel.

Algorithm 2 MultiOpt (Multi-decision online Optimal)

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job sizes S_{up_1}, S_{up_2} .

1: **for all** $t = 1, \ldots, t_D$ **do**
2: **Case 1**: **If** uploading part 2 is finished **then Break**
3: **Case 2**: **If** still uploading at t **then Continue**
4: **Case 3**: **If** uploading part 1 not started **then** perform check (4.14); **If** $\min$ is g_1 **then** start uploading part 1.
5: **Case 4**: **If** part 1 has been uploaded but part 2 has not started uploading **then** perform check (4.9); **If** $\min$ is g_2 **then** start uploading part 2.
6: **end for**

Chapter 5

A Case Study: The Gilbert-Elliot Channel

In this chapter, we consider the well-known Gilbert-Elliot channel model (Gillbert, 1960; Elliott, 1963), which has been used in many studies to model stochastic communication channels, e.g., (Zhang *et al.*, 2015, 2013; Zafer and Modiano, 2007; Johnston and Krishnamurthy, 2006; Zed *et al.*, 1995), and will be used in the results section of this thesis. This channel model is typically used to characterize the effects of burst noise in wireless channels, i.e., where the channel can abruptly transition from good to bad conditions (and vice versa). This type of channels is a difficult one for computation offloading algorithms to deal with, compared to one where there is much more correlation in the channel quality as the offloading progresses. In this chapter, closed form results are derived for this channel model that will be used to generate numerical results in Chapter 6.

5.1 Deriving Formulas

With the two state channel model, we have $B_{max} = B_g$ and $B_{min} = B_b$, where B_g and B_b are the bit rates of the good and bad channel states, respectively (in bits per time slot). In order to run Algorithm 2 with the specific energy costs of (3.7) and (3.8), we need to calculate the probabilities $P_t(T_{off}, X_t)$, which is the probability of an offload finishing in T_{off} time slots, if it starts at time slot t with channel state X_t.

Let b be the number of bad state time slots during the T_{off} offloading time slots. Given the data size S to be offloaded, b and T_{off} must satisfy $S \leq bB_b + (T_{off} - b)B_g < S + B_g$. The upper bound is due to the fact that we transmit at most $S + B_g$ bits (we assume that even when the transmission of the useful S bits has been completed, paying the transmission cost continues until the end of the last time slot). This implies that

$$\frac{(T_{off} - 1)B_g - S}{B_g - B_b} < b \leq \frac{T_{off}B_g - S}{B_g - B_b} \tag{5.1}$$

Define $\mathcal{B}$ as a set of integers b satisfying (5.1). For any $b \in \mathcal{B}$, the actual transmitted number of bits, $\hat{S}$, is given by

$$\hat{S} = bB_b + (T_{off} - b)B_g. \tag{5.2}$$

Define $\hat{P}_t(T_{off}, b, X_t)$ as the probability of an offloading, that starts at time slot t with state X_t and takes T_{off} time slots (among which b time slots are in the bad states). We have that

$$P_t(T_{off}, X_t) = \sum_{b \in \mathcal{B}} \hat{P}_t(T_{off}, b, X_t). \tag{5.3}$$

Thus, $P_t(T_{off}, X_t)$ can be obtained by summing over all of possible b's in $\hat{P}_t(T_{off}, b, X_t)$. As a special case, we set $\hat{P}_t(T_{off}, b, X_t) = 0$ for all T_{off} and X_t when $b < 0$. In order to

derive $\hat{P}_t(T_{off}, b, X_t)$, we need the following lemma.

Lemma 4. *If $\hat{S} - S \geq B_b$, then the final transmission state must be G.*

Proof. Assume, for contradiction, that the final state is B. Then, the number of bits transmitted in $T_{off} - 1$ time slots is $\hat{S}_{T_{off}-1} \geq \hat{S}_{T_{off}} - B_b$. Given the condition of the lemma, this implies that $\hat{S}_{T_{off}-1} - S \geq 0$, i.e., offloading finished within $T_{off} - 1$ time slots, a contradiction. □

Based on Lemma 4 and X_t, four different cases are considered when calculating $\hat{P}_t(T_{off}, b, X_t)$, and are obtained from elementary counting:

- $X_t = G$ and $\hat{S} - S \geq B_b$: See (5.4).
- $X_t = G$ and $\hat{S} - S < B_b$: See (5.5).
- $X_t = B$ and $\hat{S} - S \geq B_b$: See (5.6).
- $X_t = B$ and $\hat{S} - S < B_b$: See (5.7).

$$\hat{P}_t(T_{off}, b, X_t) = \begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k} \binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{5.4}$$

$$\hat{P}_t(T_{off}, b, X_t) =$$
$$\begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k} \binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} & \\ + \sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k} \binom{T_{off}-b-1}{k} P_{GB}^{k+1} P_{BG}^{k} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{5.5}$$

$$\hat{P}_t(T_{off}, b, X_t) = \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k} \binom{T_{off}-b-1}{k} P_{GB}^{k} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} \tag{5.6}$$

$$\begin{aligned} \hat{P}_t(T_{off}, b, X_t) = & \sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k} \binom{T_{off}-b-1}{k} P_{GB}^{k} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} \\ & + \sum_{k=1}^{\min(b-1, T_{off}-b)} \binom{b-1}{k} \binom{T_{off}-b-1}{k-1} P_{GB}^{k} P_{BG}^{k} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k} \end{aligned} \tag{5.7}$$

Although equations (5.4)-(5.7) can be used to calculate $P_t(T_{off}, X_t)$ directly, we now show how they can be computed recursively, which leads to a significant reduction in computation time (albeit with the use of more memory). We show this for the case $\hat{S} - S \geq B_b$ and X_t is Good (the other cases are handled similarly). In that case, (5.4) applies. We

assume $b > 0$ (case $b = 0$ is trivial). Then, (5.4) for $b > 0$ implies

$$\hat{P}_t(T_{off}, b, Good) = \sum_{k=0}^{\min\{b-1, T_{off}-b-2\}} Z(k, T_{off}) \tag{5.8}$$

where

$$Z(k, T_{off}) = \binom{b-1}{k} \binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} \; P_{BG}^{k+1} \; P_{BB}^{b-k-1} \; P_{GG}^{T_{off}-b-k-2} \tag{5.9}$$

and

$$Z(0, T_{off}) = (T_{off} - b - 1) P_{GB} \; P_{BG} \; P_{BB}^{b-1} \; P_{GG}^{T_{off}-b-2} \tag{5.10}$$

Then, it is easy to see that

$$Z(k+1, T_{off}) = \frac{(b-k-1)(T_{off}-b-k-2)}{(k+1)(k+2)} \frac{P_{GB}P_{BG}}{P_{BB}P_{GG}} Z(k, T_{off}) \tag{5.11}$$

for all $0 \leq k \leq \frac{S}{B_b}$. By treating B_b and B_g as constant, precomputing $Z(k, T_{off})$ for all $0 \leq k \leq \frac{S}{B_b}$ and $\frac{S}{B_g} \leq T_{off} \leq \frac{S}{B_b}$ takes $O(S^2)$ operations when (5.10) and (5.11) are used. Then, for any value of T_{off}, each $\hat{P}_t(T_{off}, b, G)$ can be computed with $O(S)$ operations from (5.8); eventually, $O(1)$ $\hat{P}_t$ values are combined to compute each $P_t(T_{off}, G)$ from (5.3) (note that $|\mathcal{B}| = O(1)$, and that $\hat{P}_t, P_t$ do not depend on t, except for defining X_t in their arguments). Hence, we can precompute (and store) all possible $P_t(T_{off}, X_t)$ using $O(S^2)$ operations (and memory) overall. After that, (3.7) and (3.8) imply that we can calculate $E_{off}(t, x)$ and $E_L(t, x)$ for each $1 \leq t \leq t_D$ with $O(S)$ arithmetic operations. This implies that we can use (4.2) to precompute (and store) all $g_t(x)$'s using $O(ST_D)$ operations (and memory) overall, and, therefore, all $V_t(x)$'s using $O(ST_D)$ operations (and memory), using the recursive definition (4.3). After this $O(S^2 + ST_D)$ preprocessing,

Algorithm 2 can run in $O(1)$ time per time slot. Although $T_D = \Omega(S)$ in order for the deadline to make sense, if $T_D >> \frac{S}{B_b}$ then offloading immediately would be the practical option. Therefore, we can assume that $T_D = \Theta(S)$, and the time and memory complexity of the algorithm is $O(S^2)$ in practice.

In this chapter we were able to derive better time and memory complexity than the one implied in Section 3.2, by taking advantage of the specific Markovian process structure of Gilbert-Elliot channels. In order to achieve similar gains for other Markovian channels, one will need to tailor the Dynamic Programming approach above to the specific structure of the channel Markov chain, if at all possible.

Chapter 6

Performance Comparison of Optimal and Heuristic Algorithms

In this chapter, computer simulation is used to study the performance of the proposed OnOpt Algorithm. As discussed in Chapter 5, a Gilbert-Elliot channel is assumed when offloading. It should be emphasized that based on the described system model, the optimality of the proposed OnOpt algorithm has been theoretically proved in terms of minimizing the mean energy consumption. The Gilbert-Elliott channel model is commonly used to model the effects of harsh channel conditions where burst noise can abruptly affect the data rate. The simulation results based on this channel model are used to illustrate that, even with its harsh channel conditions, there is significant gain in using the OnOpt algorithm over other heuristics. We also assume that transmit power control is used on the downlink, and therefore, T_{down} (and T_{exec}) are deterministic. Their effects can therefore be accounted for by modifying the remote offload end-times used in the analysis.

6.1 Continuous Offloading

There are three sets of simulations for Continuous Offloading, which span a wide range of parameter values. This was done to assess the relative performance of the offloading algorithms in widely varying situations. For comparison, we also plot the offline bound given in Section 3.3, *Local Execution* and two other fast algorithms, referred to as *Immediate Offloading* and *Channel Threshold*. The Local Execution algorithm executes the entire job locally without doing any offloading. For the Immediate Offloading algorithm, offloading starts at the job release time unless $S/B_g > t_D$, i.e., if offloading cannot be completed before the job deadline even with contiguous best wireless channel states, then the job is only executed locally. For the Channel Threshold algorithm, offloading starts at the first time slot when the channel condition is above a given threshold unless the remaining time before the job completion deadline is less than S/B_g. For the Gilbert-Elliot channel used in our results, any threshold between the good and bad states can be used, i.e., offloading starts at the first good channel time slot provided that the remaining time before the job completion deadline is no less than S/B_g. In both the Immediate Offloading and Channel Threshold algorithms, local execution starts at time slot t_L if offloading is not completed at time slot $t_L - 1$, i.e., they ensure that the job deadline is satisfied. The default parameters used in the simulations are given as follows. Each time slot is taken to be 1 msec. The data transmission rates are $B_b = 1$Mbps and $B_g = 10$Mbps, or $B_b = 1$kb per time slot and $B_g = 10$kb per time slot. The transmit power is 1 W, which means that the transmission energy for each time slot is $E_{tr} = 1$mJ. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$mJ and the local computation power $f_l = 1$M CPU cycles per time slot (Nir *et al.*, 2014; Huang *et al.*, 2012). We consider a job with $S = 60$Kb, $D = 10$M CPU cycles, and $t_D = 60$ time slots, where D is the number of local CPU cycles needed in order

to execute the job. Therefore, the local execution time is $T_L = D/f_l = 10$ time slots, and the local energy consumption $E_L = v_l D = 20$mJ. Based on B_g and B_g, a minimum of 6 time slots and a maximum of 60 time slots are needed in order to complete job offloading. In all of the graphs, each value of average energy consumption is obtained after repeating the simulation for 10,000 runs.

Scenario 1

Here we set $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. In this case, $P_{GB} = P_{BB}$, $P_{BG} = P_{GG}$, and the equilibrium channel state probabilities are given by $P_g = P_{GG}$ and $P_b = P_{BB}$. P_{GG} can therefore be used as a measure of the average channel quality. In this set we present graphs by varying parameters such as T_D, S, and good/bad state residency times.

Figure 6.1 shows the average energy consumption versus T_G, the asymptotic channel residence time in the good state, where $T_G = \frac{1}{P_{GB}}$. The energy used by Local Execution is obviously constant for all residence times. When the good state residence time is low, the OnOpt algorithm does not offload because there is not enough time to complete the offload, or, the expected energy is higher than E_L. As the residence time increases, the energy consumption for OnOpt decreases. The energy consumption for Channel Threshold and Immediate Offloading decreases as the residence time in the good state increases. The energy for these algorithms is above E_L when the residence time is low.

Figure 6.2 shows the average energy consumption versus T_B, the asymptotic mean channel residence time in the bad state, where $T_B = \frac{1}{P_{BG}}$. Figure 6.2 shows that as the bad state residence time increases, the energy consumption for all of the algorithms initially increases. When T_B is above about 10 time slots, both the offline bound and the

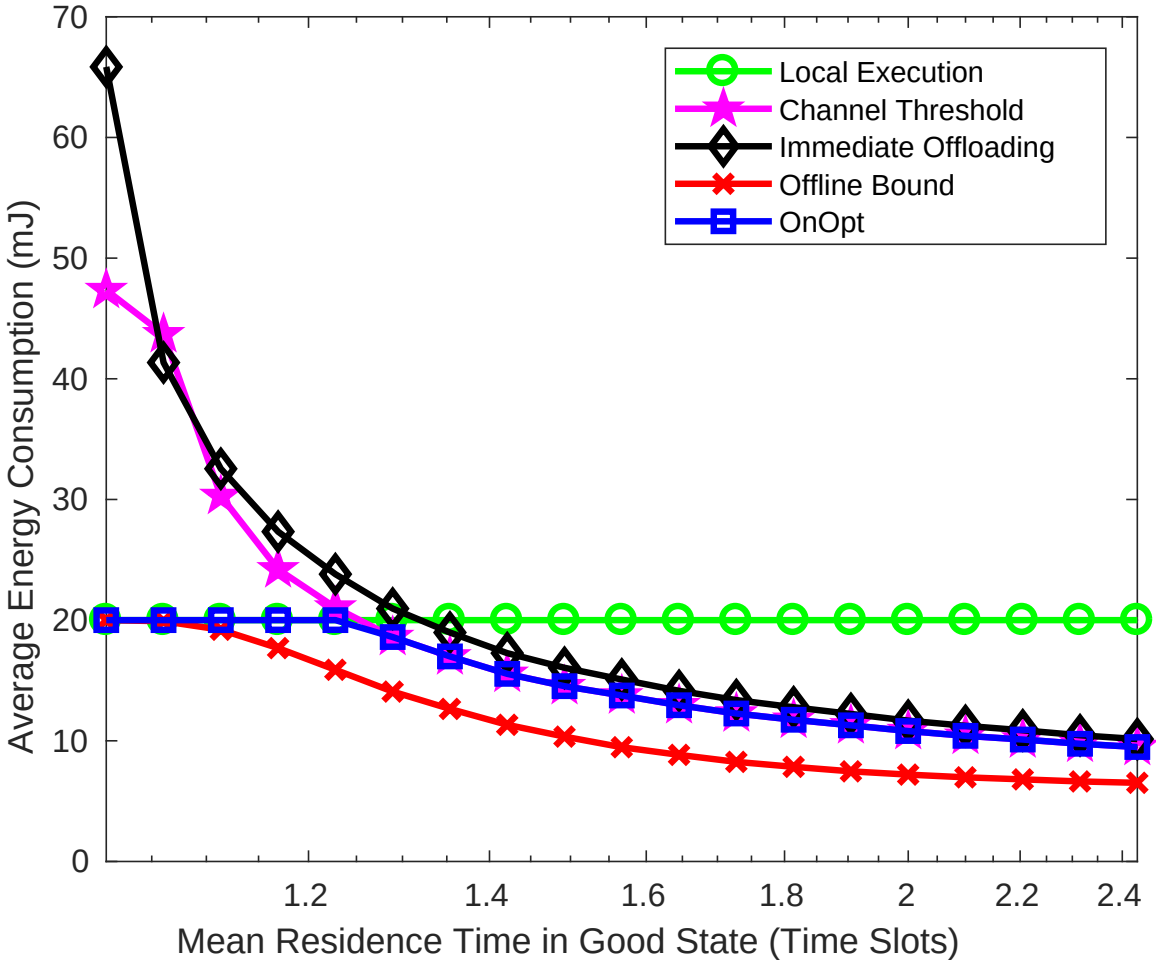

Figure 6.1: Average Energy Consumption Versus Channel Residence Time in Good State

OnOpt algorithm do not offload due to the long time needed, eventually resulting in the same energy consumption as Local Execution. For the Channel Threshold algorithm, as T_B increases, offloading may still be possible either because the channel is in the good state at the release time or the first good channel state appears not long afterwards. However, the probability that the offload can be completed before t_D decreases as T_B increases. Therefore, the energy consumption increases with T_B. As T_B further increases, offloading

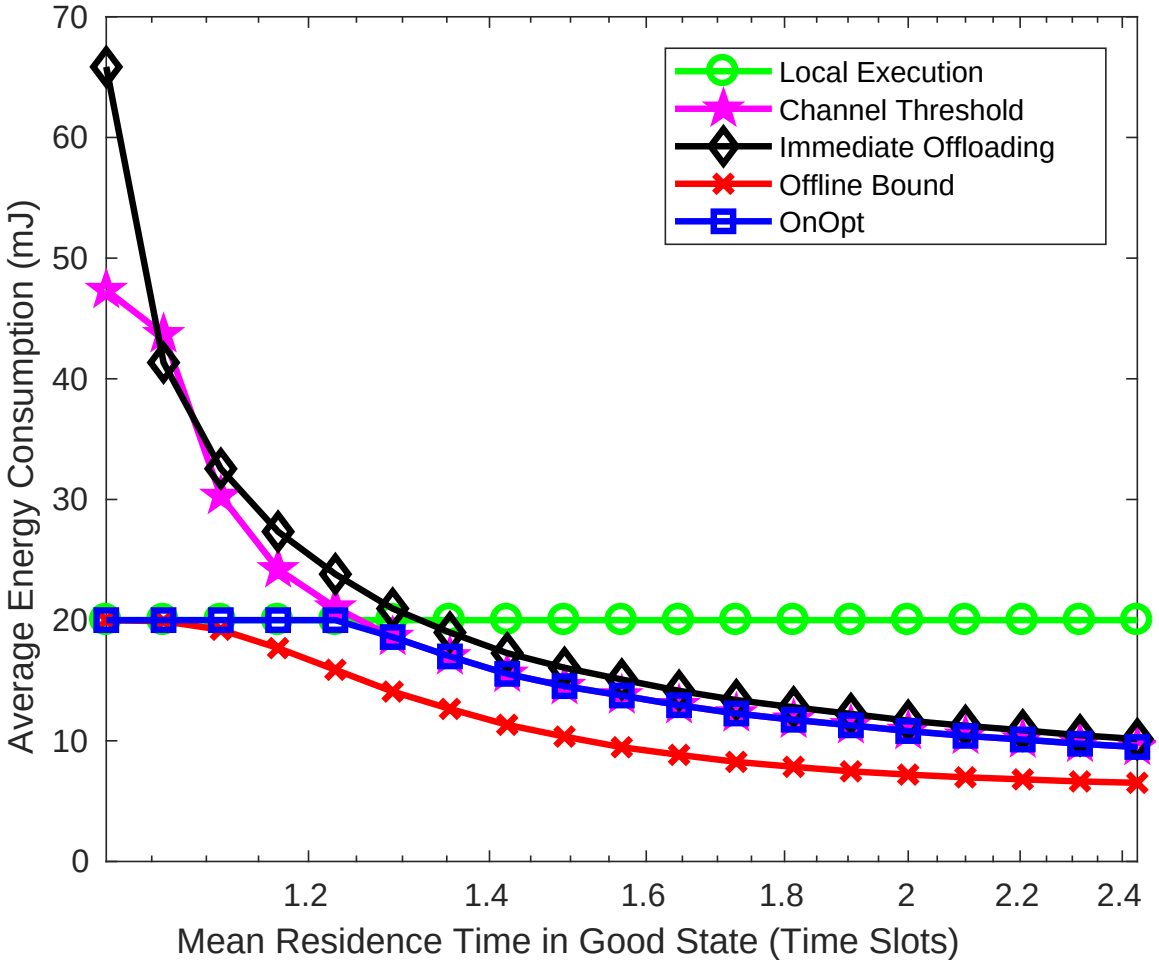

Figure 6.2: Average Energy Consumption Versus Channel Residence Time in Bad State

is possible only if the channel is in the good state at the release time (the probability decreases as T_B increases), and therefore, the energy consumption decreases. In Immediate Offloading, the energy consumption increases with T_B until T_B is so large that the channel is practically always in the bad state. The energy consumption, in this case, converges to $E_L + E_{tr}S/B_b = 80$mJ.

Figure 6.10 shows the average energy consumption of the mobile device as the data size S increases. When S is small, offloading can most likely meet the delay constraint without

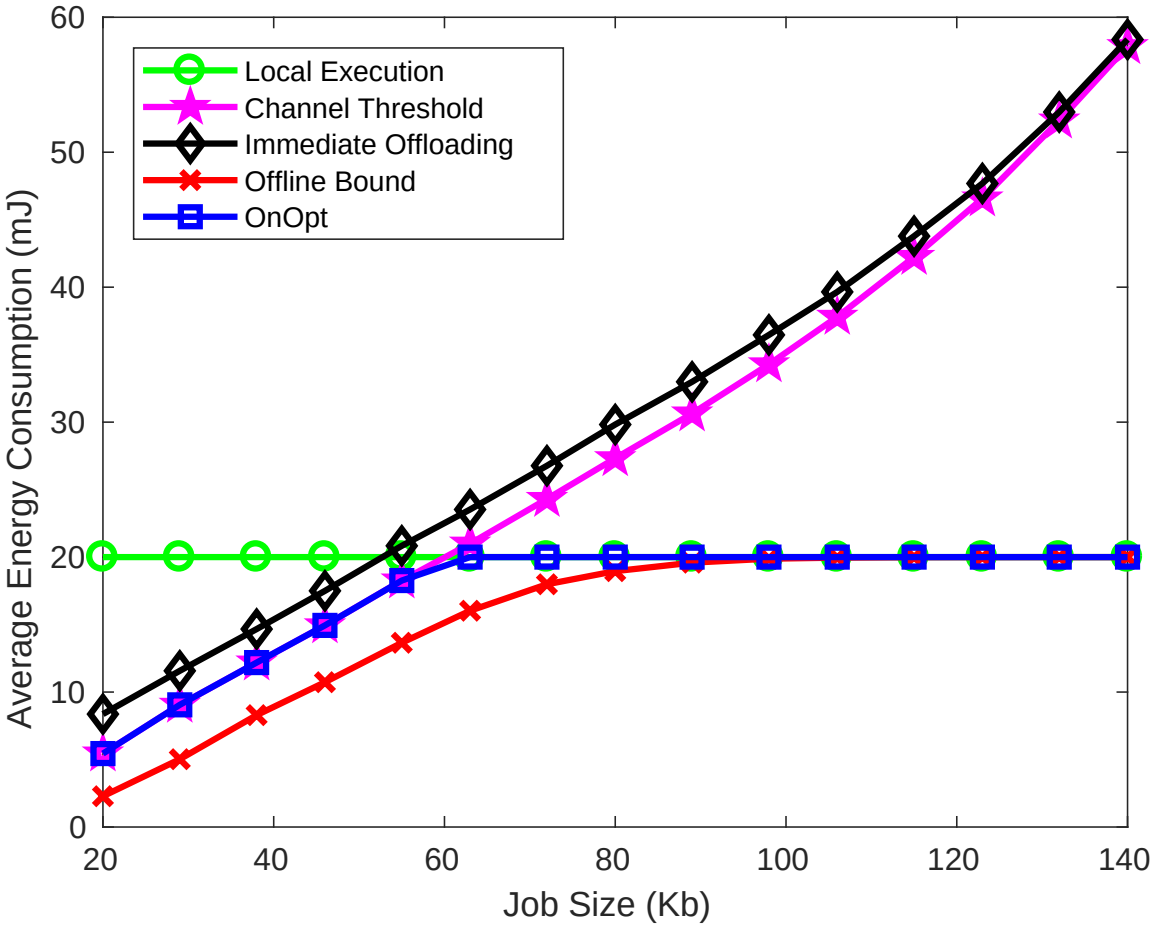

Figure 6.3: Average Energy Consumption Versus Data Size S: $P_{GG} = 0.2$

local execution. The average energy consumption of the Channel Threshold and OnOpt algorithms is the same, since the two algorithms offload at the same time slot, while the Immediate Offloading algorithm consumes higher energy for the same reason as explained previously. As S increases, a longer time is needed for wireless transmission, and both the offline and OnOpt algorithms may decide not to offload, resulting in the same energy consumption as Local Execution, while the Immediate Offloading and Channel Threshold algorithms waste energy by offloading unnecessarily, which results in much higher energy consumption.

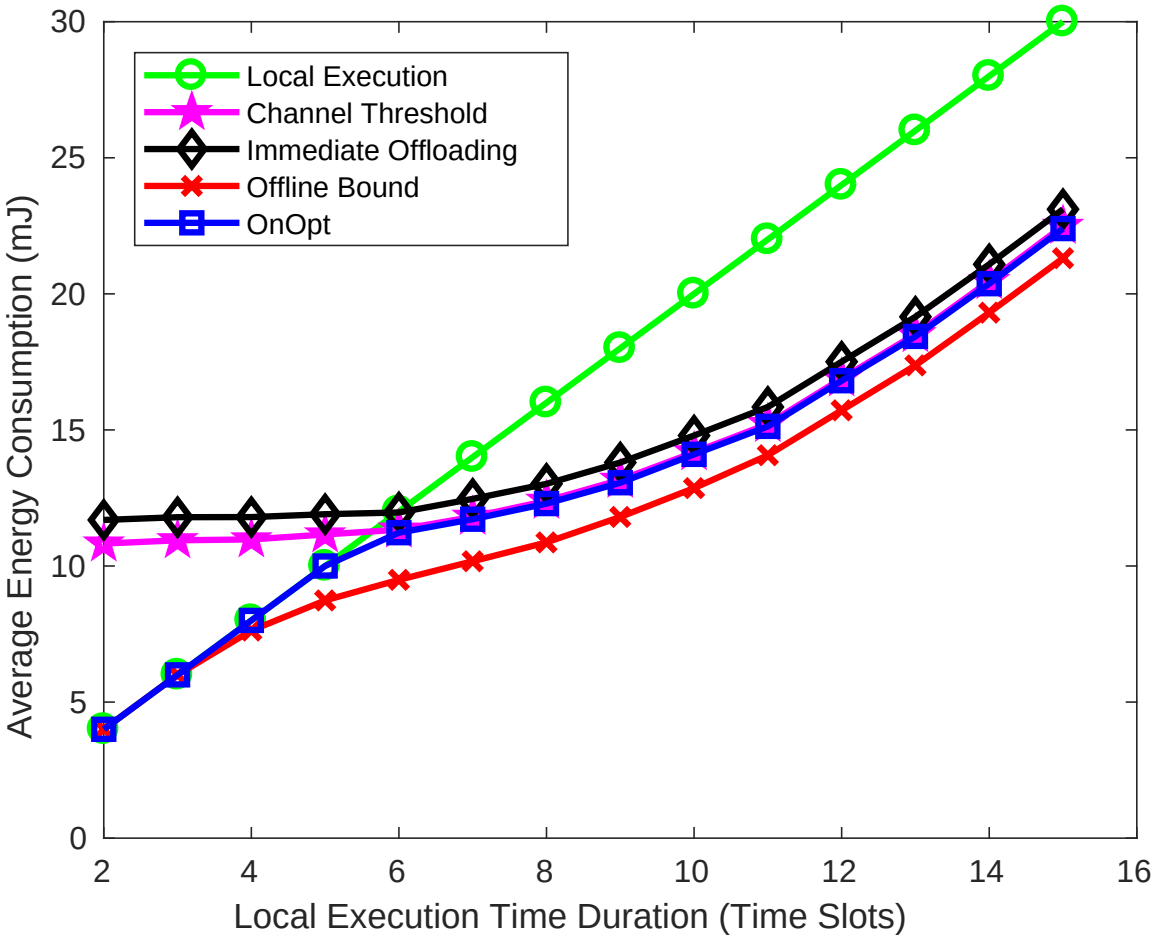

Figure 6.4: Average Energy Consumption Versus Local Execution Time T_L: $P_{GG} = 0.5$

Figure 6.4 shows the average energy consumption versus the local execution duration time. Here, we change D from 2 to 15 CPU Mega-cycles. Deadline T_D is set to 20 time slots in order to make the deadline tighter and observe the effects of increasing T_L. When E_L is small, the OnOpt algorithm does not offload because the expected cost is higher than E_L. When E_L becomes large enough, the OnOpt algorithm starts offloading, thus reducing its energy use. Increasing T_L increases the chance that overlap occurs between local execution and offloading. Therefore, the energy consumption for OnOpt starts to increase. A similar situation happens for the other algorithms.

Scenario 2

In the second set, we set $P_{BB} = P_{GG}$, so that the equilibrium channel state probabilities are equal, but by varying these parameters, we can observe the effects of mean channel state residency time. The channel state transition probabilities are assumed to satisfy $P_{BB} = P_{GG}$. In this case, the equilibrium channel state probabilities are equal, and therefore, a larger P_{GG} does not indicate better channel quality on average. Instead, it represents how dynamically the channel state changes. When P_{GG} (and P_{BB}) is large for example, once the channel enters a particular state, it is more likely to persist in that state, i.e., more consecutive time slots in the same state are likely. The opposite is true when P_{GG} (and P_{BB}) are made smaller. By varying P_{GG}, the average energy consumption of all four algorithms are given in Figure 6.11 for $t_D = 40$ time slots and Figure 6.6 for $t_D = 20$ time slots.

The offline solution can foresee future channel states, and a larger P_{GG} makes it more likely to choose consecutive time slots with good channel states. Therefore, the average energy consumption of the offline bound decreases as P_{GG} increases. When P_{GG} is very close to zero, the channel state is likely to toggle in the next time slot. In this case, the Immediate Offloading algorithm consumes about 0.5mJ extra energy, compared to the Channel Threshold algorithm, i.e., 0.5mJ is 50% (which is the probability that the channel state at the job release time is bad) times 1mJ (which is the transmission energy in the first time slot). As P_{GG} increases, it is increasingly likely to have consecutive time slots with the same channel conditions. If the channel is in the good state when a job is released, the Immediate Offloading and Channel Threshold algorithms are the same. However, if the channel is in the bad state when a job is released, it is likely that the bad channel state persists for a relatively long time, during which Immediate Offloading may waste energy. Therefore, with higher P_{GG}, the difference between Immediate Offloading and the Channel

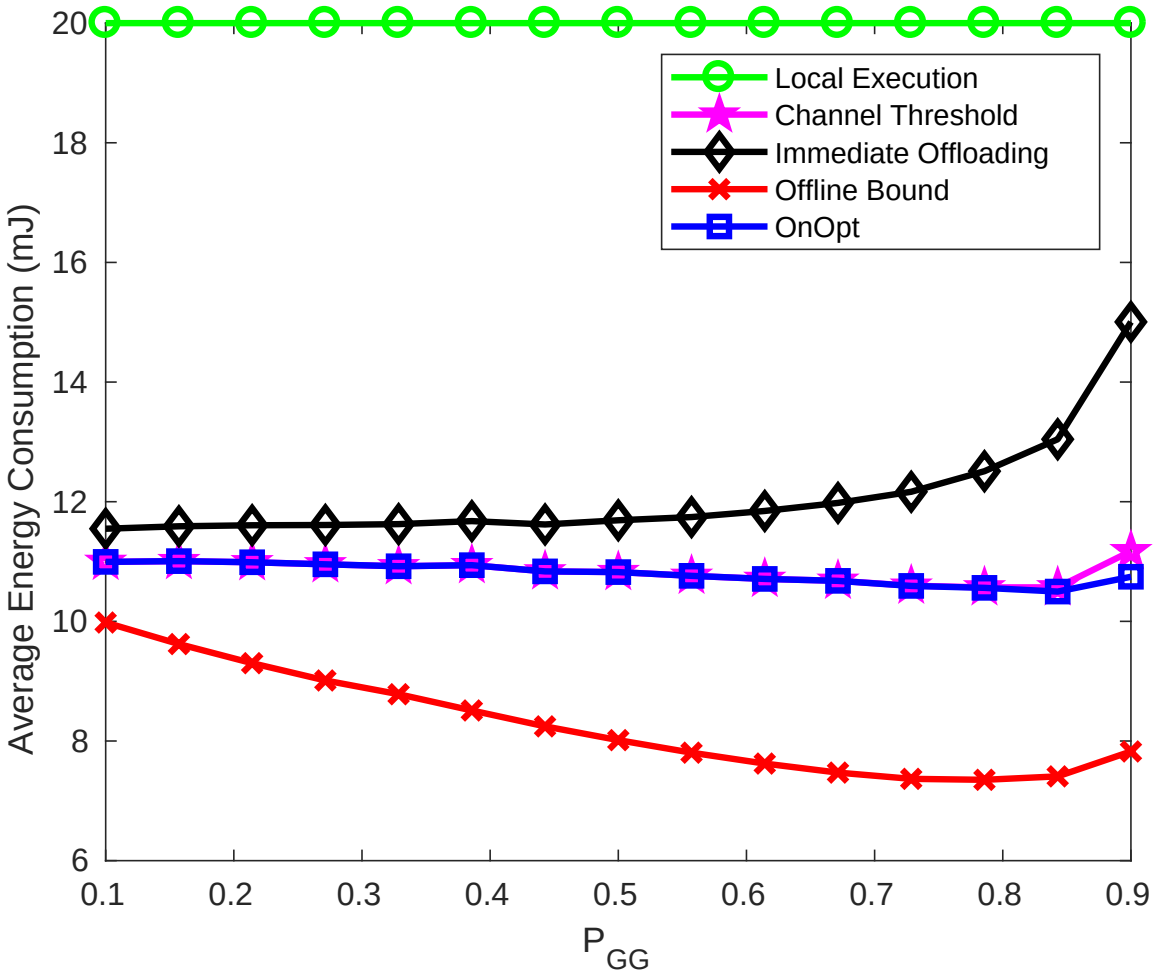

Figure 6.5: Average Energy Consumption Versus P_{GG}: $t_D = 40$ time slots

Threshold algorithm increases. The OnOpt and the Channel Threshold algorithms are very close when $t_D = 80$ time slots since the time constraint is loose enough for the OnOpt algorithm to offload at the first time slot with good channel conditions. When $t_D = 35$ time slots, the difference between the two algorithms starts increasing as P_{GG} becomes large. This is because OnOpt has the flexibility to offload at a bad time slot while the Channel Threshold algorithm does not. As a result, the OnOpt may finish offloading much sooner than the Channel Threshold algorithm.

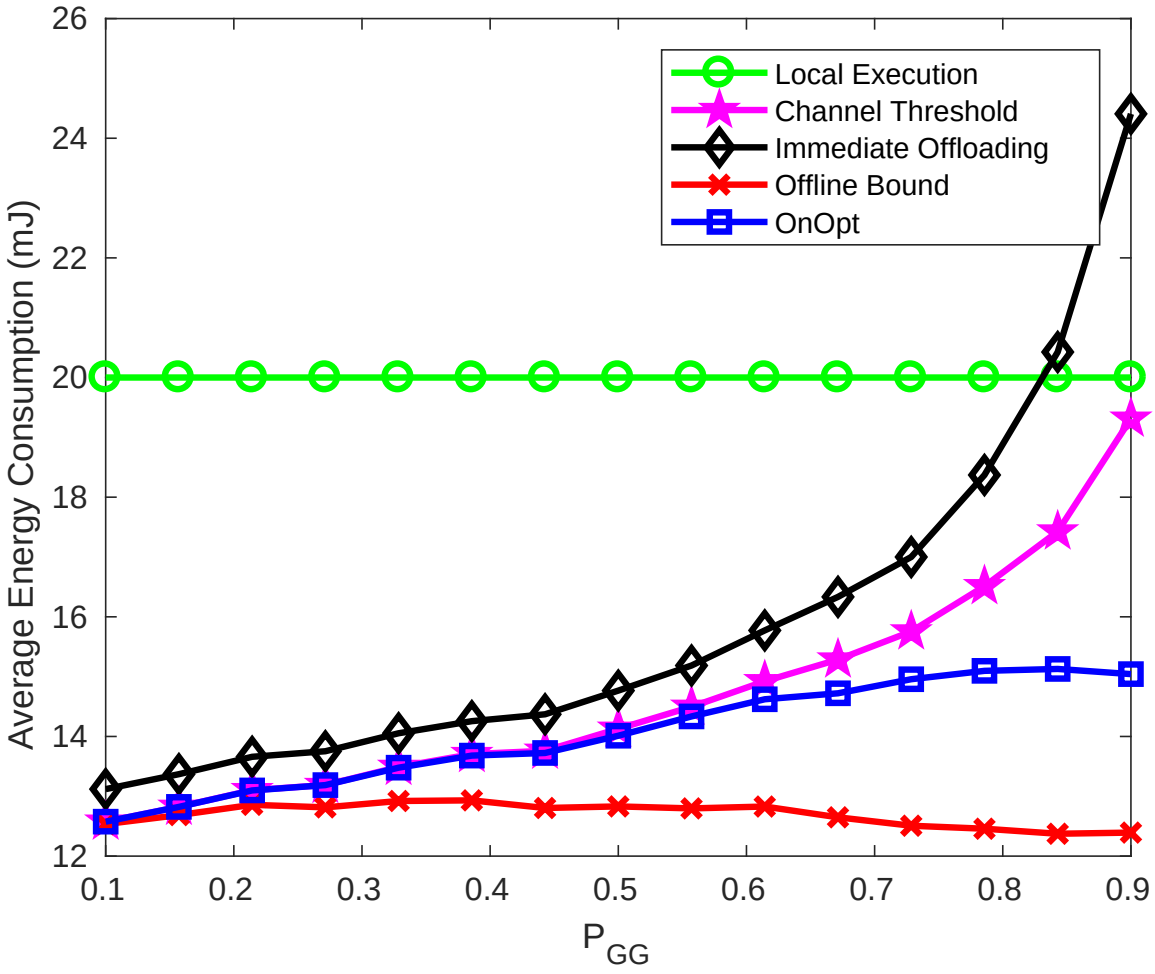

Figure 6.6: Average Energy Consumption Versus P_{GG}: $t_D = 20$ time slots

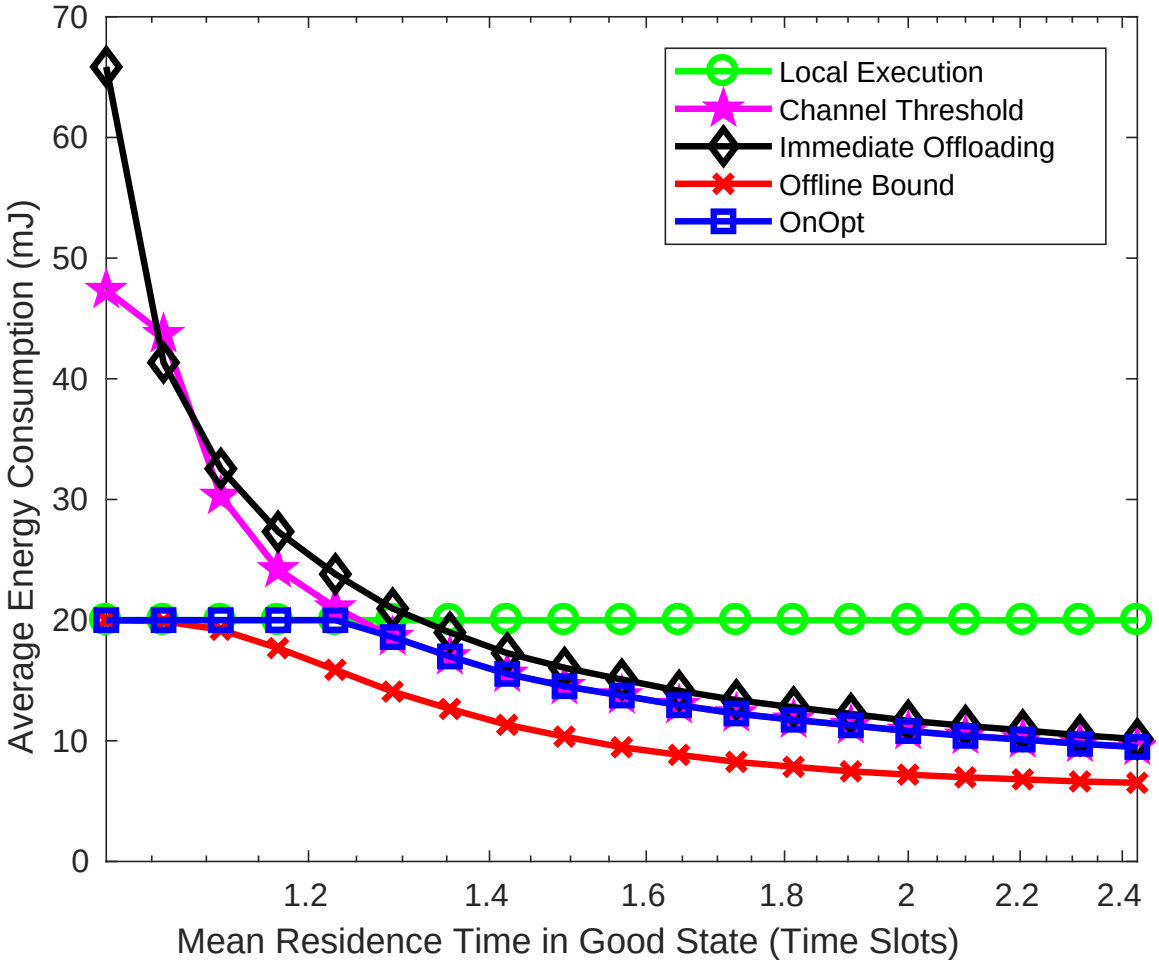

Figure 6.7: Average Energy Consumption Versus Residency Time in Good State: $t_D = 40$ time slots

Figure 6.7 shows the average energy consumption versus T_G, which is the asymptotic average channel residence time in the good state, where $T_G = 1/P_{GB} = 1/(1-P_{GG})$. Note that in this set of results, $T_B = T_G$ since $P_{GG} = P_{BB}$. When T_G is below about 10 time slots (i.e., P_{GG} is between 0.1 and 0.9), the observations are the same as seen in Figure 6.11. Therefore, the discussion below is only for $T_G > 10$ time slots. The Immediate Offloading algorithm can consume much higher energy than the others, since it may have to transmit for a long time if the channel is in the bad state at the job release time. The OpOpt and

Channel Threshold algorithms are essentially the same, since both decide to offload at the first time slot with the good channel state.

Scenario 3

In this set of results, we use the application parameters for x264 (H.264) encoding from (Miettinen and Nurminen, 2010), and consider a job with $S = 80$Kb, $D = 18$M CPU cycles, and $t_D = 80$ time slots. The local execution energy per CPU cycle is $v_l = 1.5 \times 10^{-6}$mJ and the local computation power is $f_l = 600$ M CPU cycles per second or $f_l = 0.6$ M CPU cycles per time slot. Therefore, the local execution time is $T_L = D/f_l = 30$ time slots, and the local energy consumption $E_L = v_l D = 27$mJ. Based on B_g and B_g, a minimum of 8 time slots and a maximum of 80 time slots are needed in order to complete job offloading. In addition to the results presented below, we have also simulated the algorithms based on parameters given in (Sumi *et al.*, 2014). This reference does experiments of computation offloading for face recognition on mobile devices. Since the qualitative observations and conclusions are the same as those presented below, these results have not been included.

In this case we set $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. As discussed previously, P_{GG} is a measure of the average channel quality. Figure 6.8 shows the average energy consumption of different algorithms as P_{GG} is varied. The offline bound is the same as the energy consumption of Local Execution only when P_{GG} is close to 0 and it decreases as P_{GG} increases. When P_{GG} is very low, the offline optimal solution chooses to process the job locally without offloading because of the long data transmission time (and possibly a long overlap time between offloading and local execution). As a result, there is a high probability that offloading cannot meet the delay constraint and/or consumes

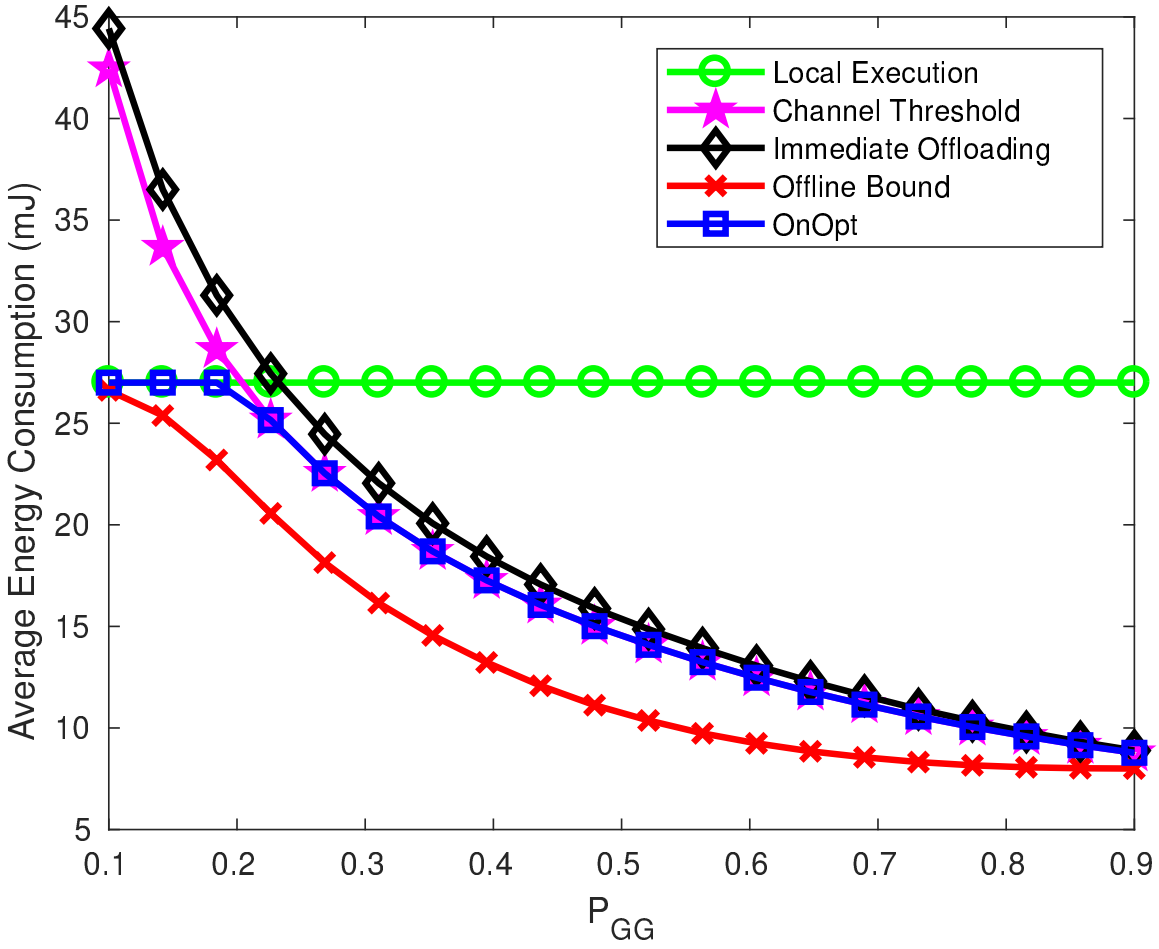

Figure 6.8: Average Energy Consumption Versus P_{GG}

higher energy than E_L. As P_{GG} becomes larger, the good channel state frequency increases, and a shorter time is needed to complete the offloading. In this case, it is more likely that offloading can meet the delay constraint and consume less energy. The Immediate Offloading algorithm results in much higher energy consumption when P_{GG} is small. Since the channel is in the bad state in most time slots, it is less likely that offloading can meet the deadline, and the deadline of the job is most likely met by performing local execution. Therefore, energy is unnecessarily wasted by performing offloading. As P_{GG} increases, the possibility that offloading can meet the deadline increases, so that less local execution is

performed, and the total energy consumption decreases. The Channel Threshold algorithm consumes slightly lower energy than Immediate Offloading. By delaying the offloading until the channel is in the good state, unnecessary transmissions are avoided. The difference is more obvious when P_{GG} is smaller, since the probability is higher that the channel is found in the bad state. For the proposed OnOpt algorithm, it chooses to not offload when P_{GG} is low, and therefore, results in the same energy consumption as Local Execution. When P_{GG} is larger, channel conditions become better and a shorter time is needed to offload. Given that the offloading decision is made using only the current channel state and statistical channel information, if the decision is to offload at a time slot, it is most likely the first time slot with a good channel state. Therefore, the OnOpt and Channel Threshold algorithms consume almost the same energy when P_{GG} is relatively large. The gap between the OnOpt algorithm and the offline bound is due to the fact that the online algorithm can only use statistical channel information, while the offline bound has knowledge of future channel conditions.

Figure 6.9 shows the average energy consumption of the algorithms as the job deadline t_D changes. For the offline bound, a loose latency constraint helps it find a better offloading time so that fewer time slots are needed to complete the required transmissions. Ideally, the minimum energy consumption is achieved if eight consecutive time slots with good channel states appear before t_L. The probability of this decreases as the deadline is tightened. However, a larger t_D increases the possibility of finding a shorter time interval to complete the offloading, thus reducing the energy consumption. When t_D is sufficiently large, it is almost always possible to find consecutive time slots with good channel states, and therefore, increasing t_D further cannot significantly decrease the average energy consumption.

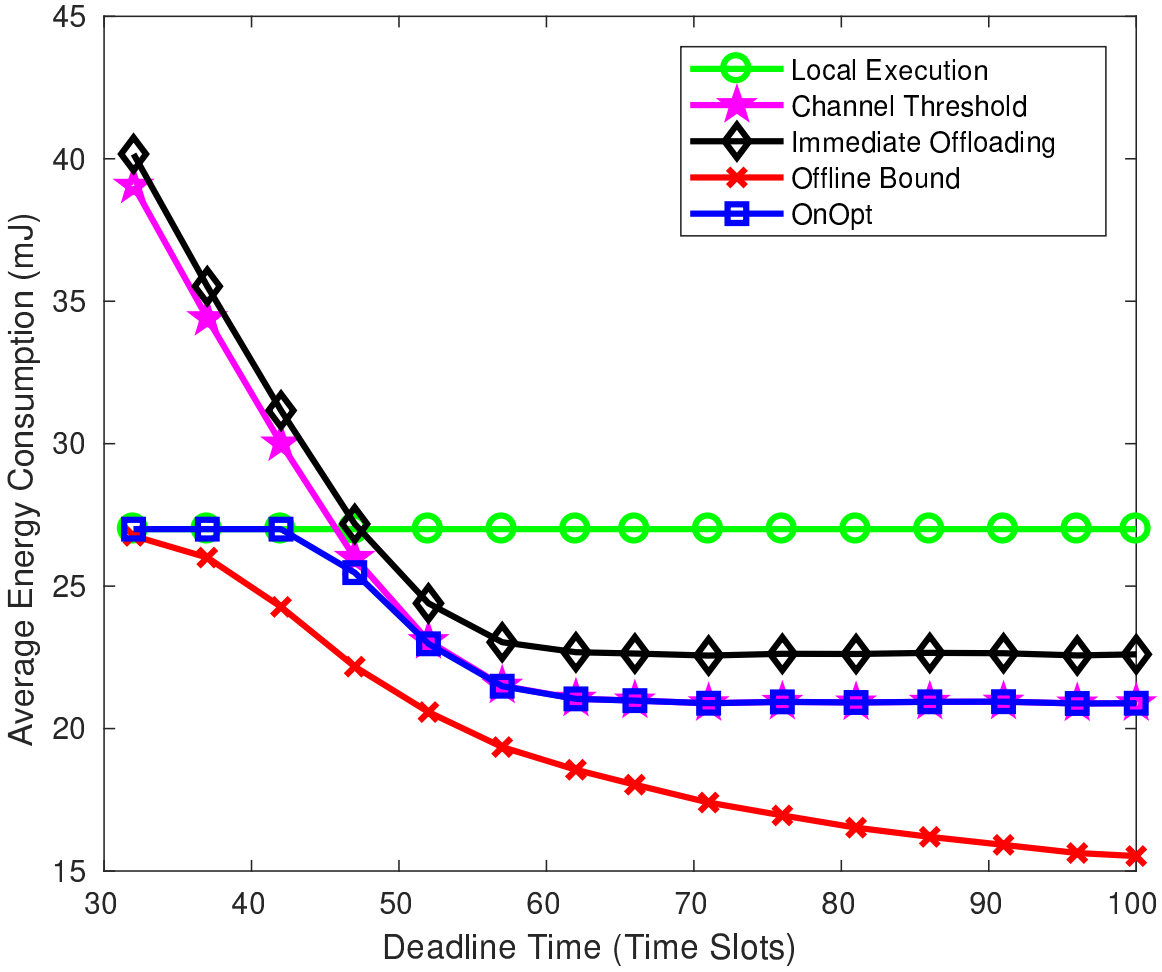

Figure 6.9: Average Energy Consumption Versus t_D: $P_{GG} = 0.3$

The Channel Threshold and OnOpt algorithms result in similar average energy consumption, which is slightly lower than using Immediate Offloading and much lower than using Local Execution, provided that t_D is not too small. As t_D increases, more time is available to offload before triggering local execution, resulting in lower energy consumption. When t_D is sufficiently large, Channel Threshold and OnOpt all start offloading at the first time slot with a good channel state, while Immediate Offloading may have to transmit over an

initial bad channel, resulting in slightly higher energy consumption than the other two algorithms. When t_D is sufficiently large so that offloading can always be completed before t_L regardless of the initial channel state, further increasing t_D does not help in reducing the energy consumption. This is true for all three online algorithms.

6.2 Multi-Part Offloading

In this section, we set $P_{BB} := 1 - P_{GG}$, in order to use P_{GG} as a measure of the average channel quality, i.e., larger P_{GG} indicates better channel conditions on average.

We compare the energy consumption of MultiOpt to an *offline bound*, the *Local Execution* of the job, and three other algorithms, namely *OnOpt*, *Immediate Offloading*, and *Channel Threshold*. The Local Execution of the entire job is done locally at the mobile device, without doing any offloading. The Immediate Offloading algorithm offloads the job immediately at its release time, unless $S_{up}/B_g + T_{rest} > T_D$, i.e., unless offloading cannot be completed before the job deadline even under the best channel conditions, in which case the job is executed locally without offloading. The Channel Threshold algorithm starts the uploading of the first part at the first time slot when the channel condition becomes Good, unless the remaining time before t_D is less than $S_{up}/B_g + T_{rest}$; when uploading the first part is completed (if the decision is to offload), uploading the second part starts as soon as the channel state becomes Good, unless the remaining time before t_D is less than $S_{up_2}/B_g + T_{rest}$. For both the Immediate Offloading and the Channel Threshold algorithms, local execution starts at time slot t_L if offloading is not completed at time slot $t_L - 1$.

We will assume that the total amount of data to be offloaded is split into two equal parts, i.e., $S_{up_1} = S_{up_2} = S_{up}/2$. The parameter settings used in the simulations are as follows: Each time slot lasts for 1 ms. The data transmission rates are $B_b = 1$Mbps and

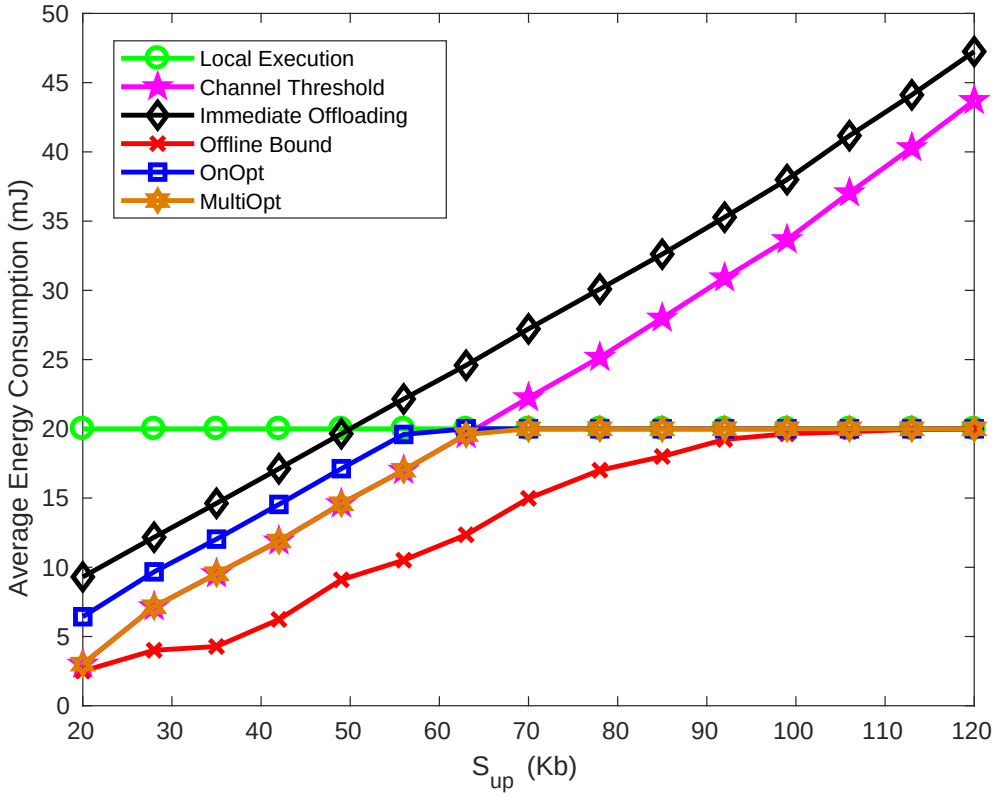

Figure 6.10: Average Energy Consumption Versus S_{up}: $P_{GG} = 0.2$

$B_g = 10$Mbps, or $B_b = 1$kb per time slot and $B_g = 10$kb per time slot. The transmission and reception power of the mobile device is 1 W and 0.5 W, respectively, which means that the transmission and reception energy per time slot is $E_{tr} = 1$mJ and $E_{rc} = 0.5$mJ, respectively. The download time T_{down} is 1 time slot. In the results below, the average energy consumption is obtained after repeating the simulation for 10,000 runs.

We first consider a job with $D = 10$M CPU cycles and $T_D = 60$ time slots. The local execution energy per CPU cycle is $\upsilon_l = 2 \times 10^{-6}$mJ and the local computation power is $f_l = 1$M CPU cycles per time slot (Nir *et al.*, 2014; Huang *et al.*, 2012). Therefore,

the local execution time is $T_L = D/f_l = 10$ time slots, and the local energy consumption $E_L = v_l D = 20$mJ. The remote execution time is $T_{exec} = 1$ time slot. Figure 6.10 shows the average energy consumption of the mobile device as the data size S_{up} increases. The energy used by Local Execution is constant for all S_{up}. When S_{up} is smaller, it is more likely for offloading to meet the delay constraint due to shorter channel uploading time. Therefore, the energy consumption of all offloading algorithms is smaller than that of Local Execution. As S_{up} increases, the average energy consumption of the Immediate Offloading and Channel Threshold algorithms keeps increasing, and can be much larger than that of Local Execution, while the average energy consumption of the offline bound, MultiOpt, and OnOpt algorithms increases first and then keeps the same as that of Local Execution as S_{up} becomes large. The Immediate Offloading algorithm has the highest energy consumption among all the algorithms because it always offloads. By delaying the offloading until the first Good channel state, the Channel Threshold algorithm consumes slightly lower average energy than Immediate Offloading, but its average energy consumption still keeps increasing with S_{up}. This is due to the fact that the offloading decision of the Channel Threshold algorithm is most beneficial in case of a continuous Good channel bit rate, which is not true during the actual uploading process, since it encounters more Bad channel states as S_{up} increases. Note that the average energy consumption of the offline bound is always the lowest, due to the future channel state information available to it. Compared to OnOpt, the energy consumption of MultiOpt is lower, as expected. By splitting the total amount of data into two parts, the MultiOpt algorithm has more flexibility that helps the mobile device to avoid uploading over long periods of bad channel states and save energy. This extra degree of freedom is not available to OnOpt, which has to continue uploading even over a bad channel once it has committed to offloading.

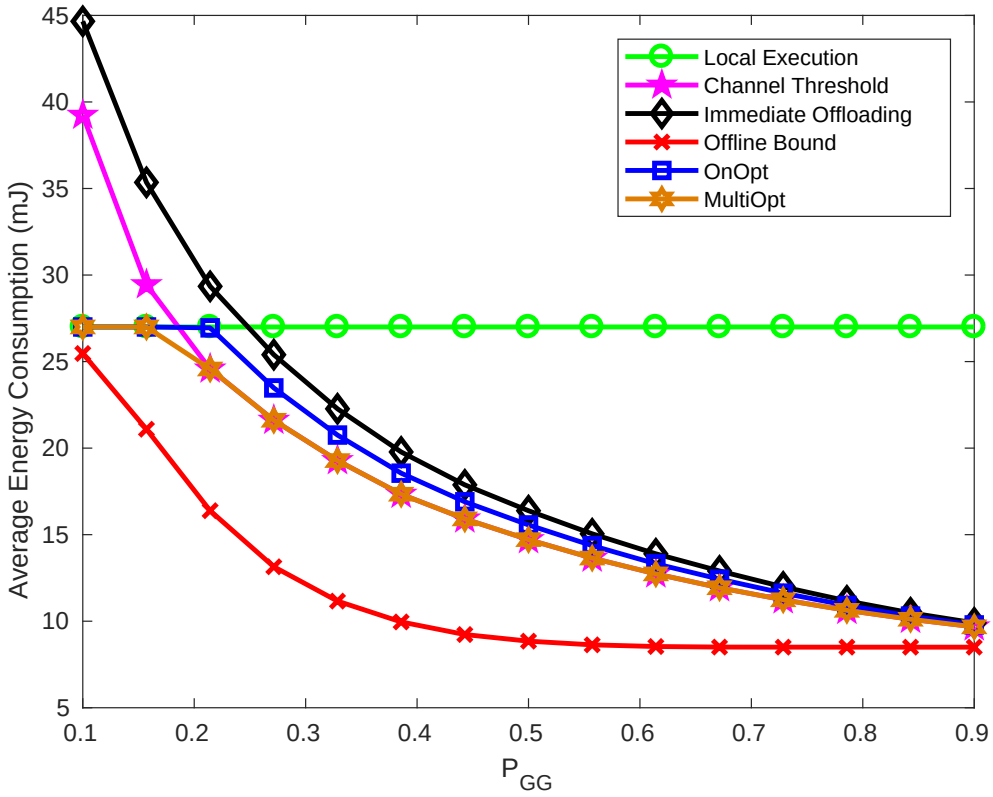

Figure 6.11: Average Energy Consumption Versus P_{GG}

Next, we use the application parameters for x264 (H.264) encoding from (Miettinen and Nurminen, 2010), and consider a job with $S_{up} = 80$Kb, $D = 18$M CPU cycles, and $T_D = 80$ time slots. The local execution energy per CPU cycle is $v_l = 1.5 \times 10^{-6}$mJ and the local computation power is $f_l = 600$ M CPU cycles per second or $f_l = 0.6$ M CPU cycles per time slot. Therefore, the local execution time is $T_L = D/f_l = 30$ time slots, and the local energy consumption $E_L = v_l D = 27$mJ. The remote execution time T_{exec} is 3 time slots. The results are shown in Figures 6.11 and 6.12.

Figure 6.11 shows the average energy consumption of different algorithms as P_{GG} varies. When P_{GG} is small, channel condition is poor, the offline bound, MultiOpt, and

OnOpt algorithms are more likely to decide to not offload, resulting in energy consumption very close to that of Local Execution, while the Immediate Offloading and Channel Threshold algorithms may consume much more energy than the latter by offloading. As P_{GG} increases, the average energy consumption of all the offloading algorithms decreases, since a shorter time is needed to complete the uploading due to better channel conditions. The energy consumption of the offline bound decreases with P_{GG} much faster than that of the other offloading algorithms due to available future information, and then becomes almost constant when P_{GG} is sufficiently large (e.g., exceeds 0.5 in Figure 6.11). Immediate Offloading results in the highest energy consumption among all the algorithms. As expected, the average energy consumption of the proposed MultiOpt algorithm is lower than that of the OnOpt for all P_{GG} values, and for the same reasons as above. When P_{GG} is close to 1, all offloading algorithms have about the same average energy consumption, since the channel conditions are almost always Good, and all the algorithms make the same offloading decisions.

Figure 6.12 shows the average energy consumption of the algorithms as the job deadline T_D changes. In general, as T_D increases, the average energy consumption of all the offloading algorithms decreases, while that of the Local Execution is not affected. When T_D is small, offloading is less likely to meet the deadline requirement; therefore, the offline bound, MultiOpt and OnOpt algorithms are more likely to decide not to offload, and, as a result, the average energy consumption of these algorithms is the same as or close to the energy consumption of Local Execution. When T_D is sufficiently large, the MultiOpt algorithm is almost the same as the Channel Threshold algorithm in terms of average energy consumption, since both algorithms end up deciding to offload at the earliest Good state for each part of the data.

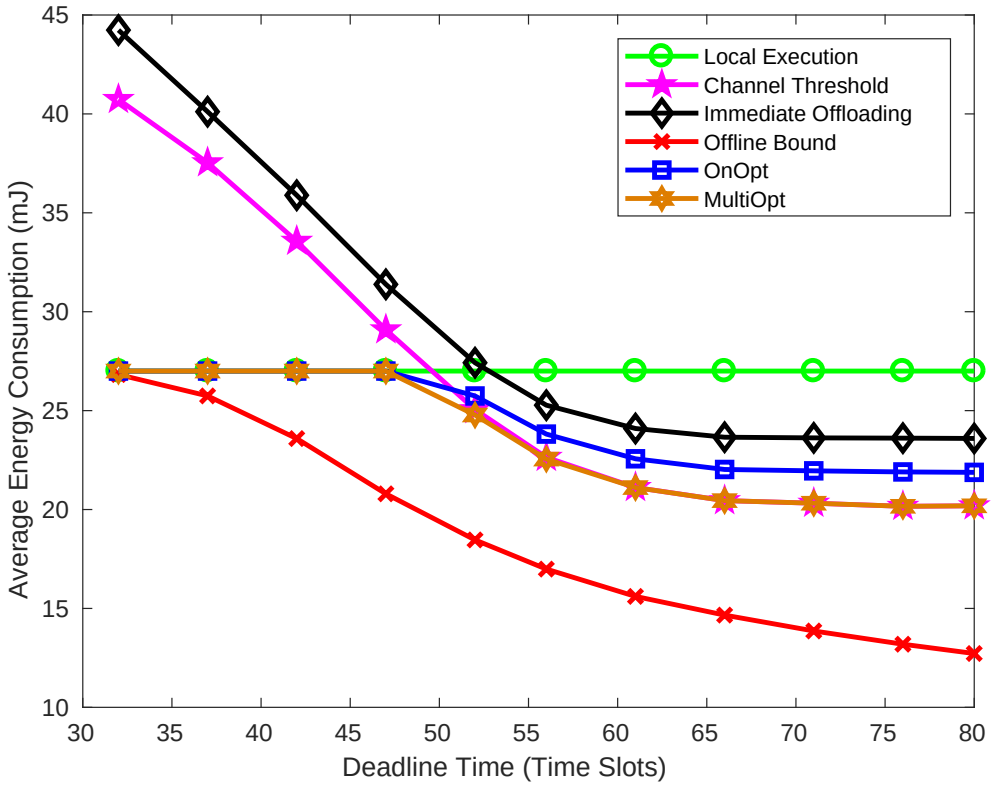

Figure 6.12: Average Energy Consumption Versus T_D: $P_{GG} = 0.3$

Chapter 7

Conclusions and Future Work

This thesis has considered mobile computation offloading where job completion times are subject to hard deadline constraints. Instead of using conventional offload/no-offload decisions, the thesis allows simultaneous remote offloading and local job execution, which is used to ensure that job completion deadlines are met in the face of random channel conditions. The thesis considered this problem when the wireless channels are modelled as homogeneous Markovian processes. The OnOpt (Online Optimum) and MultiOpt (Multi-Decision online Optimum) algorithms were proposed, and were shown to achieve the minimum mean energy consumption possible. This was done by first constructing a time-dilated absorbing Markov chain (TDAMC) from the underlying Markovian channel description. Dynamic programming results were then used with the TDAMC to prove the optimality of the algorithms.

The thesis then used the Gilbert-Elliott channel model and derived closed-form results that are used to find optimal offload initiation times. The job completion time probabilities were computed recursively, which leads to a large reduction in the computational complexity. The performance of the proposed algorithms were compared to three others that also

ensure that job deadlines are satisfied, i.e., Immediate Offloading, Channel Threshold, and Local Execution. An offline lower bound on energy consumption was computed and used in these comparisons. Performance results show that the proposed algorithms can significantly improve mobile device energy consumption compared to the other approaches while guaranteeing hard task execution deadlines.

When channel conditions are poor or the job execution time constraint is tight, the proposed algorithms may decide to not offload, saving energy by not transmitting unnecessarily. When the channel conditions are good on average, the proposed algorithms can choose the earliest transmission time, saving as much energy for local execution as possible or by not triggering local execution at all. It has been shown that the MultiOpt algorithm can reduce energy consumption compared to the OnOpt algorithm.

Partitioning the offloading data into any number of pieces will be the next step of this thesis. The final goal in this direction is to derive the optimum offloading algorithm for preemption which is the case that we minimize the energy consumption of the mobile device by offloading in any time slot.

Bibliography

(2017). Cisco Visual Networking Index: Forecast and Methodology, 2016-2021. *Cisco, San Jose, CA, USA*.

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A., and Buyya, R. (2014). Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges. *IEEE Communications Surveys and Tutorials*, **16**(1), 337–368.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical report.

Ba, H., Heinzelman, W., Janssen, C.-A., and Shi, J. (2013). Mobile computing - A green computing resource. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4451–4456.

Bahwaireth, K. S., Tawalbeh, L., Basalamah, A., Jararweh, Y., and Tawalbeh, M. (2015). Efficient Techniques for Energy Optimization in Mobile Cloud Computing. In *IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8.

Balan, R. K., Gergle, D., Satyanarayanan, M., and Herbsleb, J. (2007). Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285. ACM.

Black, M. and Edgar, W. (2009). Exploring mobile devices as grid resources: Using an x86 virtual machine to run boinc on an iphone. In *10th IEEE/ACM International Conference on Grid Computing*, pages 9 – 16.

Cao, H. and Cai, J. (2018). Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Machine Learning Approach. *IEEE Transactions on Vehicular Technology*, **68**(1), 752–764.

Chen, X. (2015). Decentralized Computation Offloading Game for Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, **26**(4), 974–983.

Chiang, M. and Zhang, T. (2016). Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, **3**(6), 854–864.

Chun, B.-G. and Maniatis, P. (2009). Augmented Smartphone Applications Through Clone Cloud Execution. In *Proceedings of 12th Conference Hot Topics Operating Systems*, page 8.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA. ACM.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and

Bahl, P. (2010). MAUI: making smartphones last longer with code offload. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 49–62.

Elliott, E. O. (1963). Estimates of error rates for codes on burst-noise channels. *Bell Syst. Tech. J.*, **42**, 1977–1997.

ETSI (2014). Mobile-edge computing introductory technical white paper. In *[Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge computing - introductory technical white paper v1*.

Flinn, J., Park, S. Y., and Satyanarayanan, M. (2002). Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 217–226. IEEE.

Galinina, O., Trushanin, A., Shumilov, V., Maslennikov, R., Saffer, Z., Andreev, S., and Koucheryavy, Y. (2013). Energy-efficient operation of a mobile user in a multi-tier cellular network. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 198–213. Springer.

Gillbert, E. N. (1960). Capacity of a burst noise channel. *Bell Syst. Tech. J.*, **39**, 1253–1266.

Giurgiu, I., Riva, O., Juric, D., Krivulev, I., and Alonso, G. (2009). Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*, pages 83–102. Springer.

Gribaudo, M., Manini, D., and Chiasserini, C. (2013). Studying mobile internet technologies with agent based mean-field models. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 112–126. Springer.

Grinstead, C. M. and Snell, J. L. (2006). Chapter 11 - markov chains. In *Grinstead and Snells Introduction to Probability*, pages 405–470.

Guan, L., Ke, X., Song, M., and Song, J. (2011). A Survey of Research on Mobile Cloud Computing. In *IEEE/ACIS International Conference on Computer and Information Science*, pages 387–392.

Huang, D., Wang, P., and Niyato, D. (2012). A dynamic offloading algorithm for mobile computing. *Wireless Communications, IEEE Transactions on*, **11**(6), 1991–1995.

Huerta-Canepa, G. and Lee, D. (2010). A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, page 6.

Johnston, L. A. and Krishnamurthy, V. (2006). Opportunistic file transfer over a fading channel: A POMDP search theory formulation with optimal threshold policies. *IEEE Transactions on Wireless Communications*, **5**(2), 394–405.

Kristensen, M. and Bouvin, N. (2008). Developing cyber foraging applications for portable devices. In *Portable Information Devices, 2008 and the 2008 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics. PORTABLE-POLYTRONIC 2008. 2nd IEEE International Interdisciplinary Conference on*, pages 1–6.

Kumar, K. and Lu, Y.-H. (2010). Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *IEEE Computer*, **43**(4), 51–56.

Lagar-Cavilla, H., Tolia, N., Lara, E. D., Satyanarayanan, M., and OHallaron, D. (2007). Interactive resource-intensive applications made easy. In *ACM/IFIP/USENIX International Conf. Middleware*, pages 143–163.

Lin, X., Wang, Y., Xie, Q., and Pedram, M. (2014). Energy and performance-aware task scheduling in a mobile cloud computing environment. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 192–199. IEEE.

Liu, F., Shu, P., Jin, H., Ding, L., Yu, J., Niu, D., and Li, B. (2013). Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Communications*, **20**(3), 14–22.

Liu, Y. and Lee, M. J. (2014). An effective dynamic programming offloading algorithm in mobile cloud computing system. In *Proceedings of IEEE International Conference on Wireless Communications and Networking (WCNC)*, pages 1868–1873.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys and Tutorials*, **19**(4), 2322–2358.

Marinelli, E. E. (2009). Hyrax: cloud computing on mobile devices using mapreduce. Technical report, DTIC Document.

Miettinen, A. P. and Nurminen, J. K. (2010). Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on hot topics in cloud computing, Berkeley, CA, USA*.

Nir, M., Matrawy, A., and St-Hilaire, M. (2014). An energy optimizing scheduler for mobile cloud computing environments. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 404–409. IEEE.

Paczkowski, J. (2009). iphone owners would like to replace battery, at&t. Technical report.

Patidar, S., Rane, D., and Jain, P. (2012). A survey paper on cloud computing. In *2012 Second International Conference on Advanced Computing and Communication Technologies*, pages 394 – 398.

Peskir, G. and Shiryaev, A. (2006). *Optimal stopping and free-boundary problems*. Lectures in Mathematics ETH Zurich. Springer, Dordrecht.

Porras, J., Riva, O., and Kristensen, M. D. (2009). Dynamic resource management and cyber foraging. In *Middleware for Network Eccentric and Mobile Applications*, pages 349–368. Springer.

Rahmati, A. and Zhong, L. (2007). Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 165–178. ACM.

Rehman Khan, A. u., Othman, M., Madani, S. A., and Ullah Khan, S. (2014). A Survey of Mobile Cloud Computing Application Models. *IEEE Communications Surveys and Tutorials*, **16**(1), 393–413.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1998). Saving Portable Computer Battery Power through Remote Process Execution. *ACM SIGMOBILE Mobile Computing and Communications Review*, **2**(1), 19–26.

Rudenko, A., Reiher, P., Popek, G. J., and Kuenning, G. H. (1999). The remote processing framework for portable computer power saving. In *Proceedings of the 1999 ACM symposium on Applied computing*, pages 365–372.

Satyanarayanan, M., Bahl, P., and Cceres, R. (2009). The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, **8**(4), 14–23.

Scarsella, A. and Stofega, W. (2019). Worldwide smartphone forecast, 20192023. Technical report.

Shiraz, M., Gani, A., Khokhar, R. H., and Buyya, R. (2013). A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, **15**(3), 1294–1313.

Su, Y.-Y. and Flinn, J. (2005). Slingshot: Deploying stateful services in wireless hotspots. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, pages 79–92, New York, NY, USA. ACM.

Sumi, N., Baba, A., and Moshnyaga, V. G. (2014). Effect of computation offload on performance and energy consumption of mobile face recognition. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–7.

Wang, H. (2006). Optimal stopping of markov chains. In *Introduction to Stochastic Control Theory (I) Notes*.

Wei, X., Fan, J., Lu, Z., and Ding, K. (2013). Application scheduling in mobile cloud computing with load balancing. *Journal of Applied Mathematics*, **2013**.

You, C., Huang, K., and Chae, H. (2016). Energy Efficient Mobile Cloud Computing Powered by Wireless Energy Transfer. *IEEE Journal on Selected Areas in Communications*, **34**(5), 1757–1771.

Zafer, M. and Modiano, E. (2007). Minimum energy transmission over a wireless fading channel with packet deadlines. In *Proceedings of IEEE International Conference on Decision and Control*, pages 1148–1155.

Zed, M., Rao, R. R., and Milstein, L. B. (1995). On the accuracy of a first-order Markov model for data transmission on fading channels. In *IEEE International Conference on Universal Personal Communications*, pages 211–215.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. In *J. Internet Services Appl.*, volume 1, page 718.

Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O. (2013). Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel. *IEEE Transactions on Wireless Communications*, **12**(9), 4569–4581.

Zhang, W., Wen, Y., and Wu, D. O. (2015). Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel. *IEEE Transactions on Wireless Communications*, **14**(1), 81–93.

Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., and Jeong, S. (2009). Securing elastic applications on mobile devices for cloud computing. In *ACM workshop on Cloud computing security*, pages 127–134.