EFFICIENT MOBILE COMPUTATION OFFLOADING WITH HARD TASK DEADLINES AND CONCURRENT LOCAL EXECUTION

EFFICIENT MOBILE COMPUTATION OFFLOADING WITH HARD TASK DEADLINES AND CONCURRENT LOCAL EXECUTION

BY

PEYVAND TEYMOORI, M.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ECE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2021) McMaster University

(ECE) Hamilton, Ontario, Canada

TITLE:	Efficient Mobile Computation Offloading with Hard Task Deadlines and Concurrent Local Execution
AUTHOR:	Peyvand Teymoori M.Sc. (Electrical & Computer Engineering), Iran University of Science and Technology, Tehran, Iran
SUPERVISOR:	Prof. George Karakostas Prof. Terence D. Todd Prof. Dongmei Zhao
NUMBER OF PAGES:	xvi, 163

In memory of

my beloved father

Lay Abstract

This work considers the problem of mobile computation offloading over stochastic wireless communication channels. The objective is to minimize the energy consumption of the mobile device while satisfying a hard task execution deadline. To guarantee that hard task execution deadline constraints are met, a model of concurrent local execution (CLE) is introduced, where local task execution may be initiated even if remote offloading is in progress. This problem is addressed for continuous (1-Part), multi-part (K-Part), and preemptive offloading scenarios. The theory of optimal stopping for Markov chains and dynamic programming are used to develop provably optimal online offloading algorithms for each offloading scenario. The computational complexity of these algorithms, especially in the case of preemptive offloading, can be restrictive. Some approximation techniques are proposed to reduce the computational complexity of the online algorithms.

Abstract

Mobile computation offloading (MCO) can alleviate the hardware limitations of mobile devices by migrating heavy computational tasks from mobile devices to more powerful cloud servers. This can lead to better performance and energy savings for the mobile devices. This thesis considers MCO over stochastic wireless channels when task completion times are subject to hard deadline constraints. Hard deadlines, however, are difficult to meet in conventional computation offloading due to the randomness caused by the wireless channels. In the proposed offloading policies, concurrent local execution (CLE) is used to guarantee task execution time constraints. By sometimes allowing simultaneous local and remote execution, CLE ensures that job deadlines are always satisfied in the face of any unexpected wireless channel conditions. The thesis introduces online optimal algorithms that reduce the remote and local execution overlap so that energy wastage is minimized. Markov processes are used to model the communication channels.

MCO is addressed for three different job offloading schemes: continuous, multi-part, and preemptive. In the case of continuous offloading, referred to as *1-Part offloading*, the mobile device will upload the entire job in one piece without interruption, when the scheduler decides to do so. In multi-part computation offloading, the job is partitioned into a known number (K) of parts, and each part is uploaded separately. In this offloading mechanism, which is referred to as *K-Part Offloading*, the upload initiation times of each part

must be determined dynamically during runtime, and there may be waiting time periods between consecutive upload parts. *Preemptive offloading* is a generalization of K-Part Offloading where the number of task upload parts is unknown. In this scheme, a decision to either continue offloading or to temporarily interrupt the offload is made at the start of each time slot.

Compared to the conventional contiguous computation offloading, interrupted offloading mechanisms (i.e., K-Part and preemptive offloading) allow the system to adapt when channel conditions change and therefore may result in lower mobile device energy consumption. This energy reduction will be obtained at the expense of having higher computational complexity.

In this thesis, for each offloading scheme, an online computation offloading algorithm is introduced by constructing a time-dilated absorbing Markov chain (TDAMC) and applying dynamic programming (DP). These algorithms are shown to be energy-optimal while ensuring that the hard task deadline constraints are always satisfied. The optimality of these algorithms is proved using Markovian decision process stopping theory. Since the computational complexity of the proposed online algorithms, especially in the case of preemptive offloading, can be significant, three simpler and computationally efficient approximation methods are introduced: Markovian Compression (MC), Time Compression (TC), and Preemption Using Continuous Offloading (Preemption-CO). MC and TC reduce the state space of the offloading Markovian process by using a novel notion of geometric similarity or by running an optimal online offloading algorithm in periodic time steps. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every time-slot is based on non-preemptive calculations. These methods are used alone or in combination to construct practical offloading algorithms. A variety of results are presented

that show the tradeoffs between complexity and mobile energy-saving performance for the different algorithms.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my supervisors, Prof. Terence D. Todd, Prof. George Karakostas, and Prof. Dongmei Zhao, for their dedicated support, encouragement, and valuable guidance throughout this research. It was a great opportunity and honor to work under their supervision. Their willingness to offer me so much of their precious time and intellect is the main reason for successfully completing this thesis.

I extend my warm thanks to my supervisory committee member, Prof. Douglas Down, for his insightful advice and comments. I would also like to thank the members of the examining committee for reading my thesis and providing thoughtful suggestions.

I would like to thank my warmhearted father for his endless love and sacrifices. This is so sad that he passed away last year, and could not see my graduation. He will be alive in my heart forever. I am extremely grateful to my mother for her efforts, motivations, and for teaching me valuable lessons throughout my life.

I thank my fellow colleagues in the Wireless Networking Laboratory for their support and feedback. Special thanks to my friend and former lab mate, Arvin Hekmati for his contribution and cooperation in this research. I would also like to thank my kind friend, Hong Chen, whom I have had the pleasure of working with.

Abbreviations

1G	First Generation
3G	Third Generation
4G	Fourth Generation
5G	Fifth Generation
BTS	Base Transceiver Station
CC	Cloud Computing
CLE	Concurrent Local Execution
CO	Continuous Offloading
CMDP	Constrained Markov Decision Process
CPU	Central Processing Unit
CSMC	Channel State Markov Chain
DAG	Directed Acyclic Graph
DP	Dynamic Programming
DTMC	Discrete-Time Markov Chain
D2D	Device-to-Device
DVS	Dynamic Voltage Scaling
EC2	Elastic Compute Cloud

EMOP	Energy-efficient Multisite Offloading Policy
ERTP	Energy-Response Time Product
ERTWP	Energy-Response Time Weighted Product
ERTWS	Energy-Response Time Weighted Sum
FHMDP	Finite Horizon Markov Decision Process
GE	Gilbert-Elliot
IP	Integer Programming
ILP	Integer Linear Programming
IaaS	Infrastructure as a Service
LAN	Local Area Network
LARAC	Lagrangian Relaxation Based Aggregated Cost
LODCO	Lyapunov Optimization-based Dynamic Computation Offloading
LTE	Long Term Evolution
MC	Markovian Compression
MCC	Mobile Cloud Computing
MCO	Mobile Computation Offloading
MDP	Markov Decision Process
MultiOpt	Multi-Decision Online Optimum
OnOpt	Online Optimum
OPO	Optimal Preemptive Offloading
PaaS	Platform as a Service
QoS	Quality of Service
RL	Reinforcement Learning

SaaS	Software as a Service
SMD	Smart Mobile Device
SMS	Short Message Service
TC	Time Compression
TDAMC	Time-Dilated Absorbing Markov Chain
TDMA	Time Division Multiple Access
VIA	Value Iteration Algorithm
VM	Virtual Machine
WAP	Wireless Access Point
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

Contents

Lay Abstract **iv**

Abstract **v**

Acknowledgements **viii**

Abbreviations **ix**

1 Introduction **1**

1.1 Overview 1

1.2 Contributions 3

2 Background **7**

2.1 Introduction 7

2.2 Mobile Computing 8

2.3 Cloud Computing 10

2.4 Mobile Cloud Computing 12

2.5 Mobile Computation Offloading 15

2.6 Related Work 17

3 Continuous Offloading **33**

3.1 Introduction 33

3.2 System Model 34

3.3 Offline Bound 39

3.4 Markovian Channel and the Time-Dilated Absorbing Markov Model 41

3.5 Optimal Stopping and the OnOpt (Online Optimal) Algorithm 48

3.6 The Gilbert-Elliot Channel Case 51

3.7 Simulation Results 56

4 Multipart Offloading **67**

4.1 Introduction 67

4.2 System Model 69

4.3 Offline Bound 71

4.4 Markovian Channel and the Time-Dilated Absorbing Markov Model 72

4.5 Optimal Algorithm for K-Part Offloading 77

4.6 Simulation Results 81

5 Preemptive Offloading **90**

5.1 Introduction 90

5.2 System Model 93

5.3 Offline Bound 96

5.4 Optimal Algorithm for Preemptive Offloading 97

5.5 Practical Heuristics 104

5.6 Simulation Results 111

6 Approximate Solutions **124**

6.1 Introduction 124

6.2 System Model 126

6.3 Problem Formulation and Optimal Solution 126

6.4 Approximate Solutions 129

6.5 Simulation Results 134

7 Conclusions and Future Work **145**

List of Figures

2.1 Cellular network architecture 9

2.2 Cloud computing evolution 11

2.3 Mobile cloud computing architecture 14

3.1 Concurrent local and remote 1-Part computational job offloading time line . 37

3.2 Time dilated absorbing Markov chain example corresponding to a Gilbert-Elliot channel model. 44

3.3 Average energy consumption vs job size and local execution time duration. 60

3.4 Average energy consumption vs P_{GG} and T_G. 62

3.5 Average energy consumption versus P_{GG} and t_D 64

4.1 Job offloading timing parameters 70

4.2 $TDAMC_1$ when offloading S_{up_1} starts at time t_s. 74

4.3 Average energy consumption versus data size S_{up} and D 84

4.4 Average energy consumption versus P_{GG} and T_D. 86

5.1 Concurrent local and remote preemptive computational job offloading time line 94

5.2 Time dilated absorbing Markov process (TDMP) for a two-state (G, B) channel. 95

5.3 Markov Chain for $\mathbb{P} = \mathbb{P}_1$ and $\mathbb{P}_2$ 113

5.4 Markov Chain for $\mathbb{P} = \mathbb{P}_3$ and $\mathbb{P}_4$ 114
5.5 9-states uniform distribution: $\mathbb{P} = \mathbb{P}_1$ 115
5.6 9-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_2$ 119
5.7 5-states uniform distribution: $\mathbb{P} = \mathbb{P}_3$ 120
5.8 5-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_4$ 122
6.1 Markovian Compression of the original TDAMC (left) to a smaller TDAMC^{approx} (right). 128
6.2 Approximation of 1-Part offloading using 1-Part-MC for different values of α 136
6.3 Approximation of 1-Part offloading using 1-Part-TC for different values of β 137
6.4 Approximation of 1-Part offloading using 1-Part-MC-TC for different combinations of α and β 139
6.5 Approximation of Preemption offloading using Preemption-CO-MC for different values of α 141
6.6 Approximation of Preemption offloading using Preemption-CO-TC for different combinations of β 142
6.7 Approximation of Preemption offloading using Preemption-CO-MC-TC for different combinations of α and β 144

Chapter 1

Introduction

1.1 Overview

Due to advances in wireless communications and computer architecture, mobile devices are being transformed into ubiquitous computing platforms (De (2016)). According to the Ericsson mobility report 2020 (Ericsson (2020)), smartphone use by 2025 will grow to about 8.9 billion, from the current 8.1 billion subscription level. As this number increases, there is an increasing demand from mobile users for running computational heavy and energy-hungry applications which require a significant amount of central processing unit (CPU) and battery power.

Compared to their wired counterparts, mobile terminals are generally more resource-constrained; in particular, the limited computational power, wireless communication capacity, and battery life of these devices are their inherent limitations. Therefore, running complex applications on such devices is challenging. Cloud computing, which is instantly accessible for mobile terminals with built-in wireless interfaces, is a natural solution to augment mobile platforms' capabilities at low cost. The term mobile cloud computing

(MCC) was introduced not long after the emergence of cloud computing around 2007 (Dinh *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); Rudenko *et al.* (1999)). With the widespread penetration of third-/fourth-generation (3G/4G) wireless networks, MCC offers a promising and viable solution that narrows the gap between the ever-increasing demands of new applications and the limited mobile device resources. MCC extends the capabilities of mobile devices by performing mobile computation offloading (MCO), i.e., offloading computation tasks from mobile devices to a capable cloud server via wireless networks. The potential benefits obtained from offloading include reducing the job response time as well as decreasing the amount of energy needed to process a job.

The advent of LTE and 5G wireless communication networks has further facilitated the use of mobile cloud computing. During the year 2020, the spread of COVID-19 has led to social distancing, which has kept millions at home, placing significant demands on infrastructure and telecommunications. Partly for this reason, 5G technology deployment has been accelerated compared to previous years. LTE will remain the dominant mobile access technology. It is projected to peak in 2022 at 5.1 billion subscriptions and decline to around 4.4 billion by the end of 2025 as more subscribers migrate to 5G. It has been estimated that tens of billions of cloud-based network edge devices will be deployed in the future to satisfy mobile demands. This will provide significant resources for performing computationally intensive and latency-critical mobile-centric tasks (Patel *et al.* (2014); Mao *et al.* (2017)).

There is a large literature that has studied various issues dealing with mobile computation offloading (MCO) (Ba *et al.* (2013); Huerta-Canepa and Lee (2010); Chun *et al.* (2011); Chun and Maniatis (2009); Satyanarayanan *et al.* (2009)). Since the offloading

itself will incur both wireless communication energy and latency costs, the decision to offload a task is not always straightforward. Making optimal offloading decisions is even more complicated when the mobile device interacts with the cloud over stochastic transmission channels. This issue was studied in (Kumar and Lu (2010)), where an energy model was proposed that considers both computation and energy components, using the statistics of the wireless channel. The channel was assumed to remain constant throughout the computation offload at the state encountered by the mobile device at the start of the offload. In more general situations, however, the wireless channel condition may randomly evolve during the computation offload. Varying channel conditions further complicate the decision to offload or to execute a given task locally, especially when the task execution time is subject to a hard deadline constraint, i.e., deadline constraints that should never be violated. Hard task execution deadlines, as opposed to deadline constraints that are satisfied with a high probability, or incur a penalty when violated (Zhang *et al.* (2014); Zhang *et al.* (2013a)), 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, a complete channel outage can even occur over extended time periods. This is the environment that is considered in this thesis.

1.2 Contributions

This work considers mobile computation offloading under stochastic wireless channel conditions when task completion times are subjected to hard deadline constraints. This objective will become increasingly important as mobile applications become more sophisticated and interact more closely with cloud server job execution (Lagar-Cavilla *et al.* (2007)).

The wireless transmission channel is modeled with a discrete-time homogenous

Markov chain. It is assumed that the Markov channel states and the state transition probabilities are known by the mobile terminal. Estimating the quality of the wireless channel to define the corresponding Markov model has been widely studied in the literature in the field of mobile communication networks (Oyerinde and Mneney (2012) Bildea *et al.* (2015) Lioumpas *et al.* (2007) Pu *et al.* (2010) Sadeghi *et al.* (2008)).

To guarantee a hard job completion deadline constraint, in the face of unforeseen channel conditions, Concurrent Local Execution (CLE) is used, where local task execution may be initiated even if remote offloading is in progress. This mechanism can ensure that hard task deadlines are satisfied regardless of any randomness induced by the wireless channel, network, or cloud servers. This is in contrast to the conventional computation offloading model where job execution is either local or remote (Chun *et al.* (2011); Cuervo *et al.* (2010)). As is the case in conventional computation offloading, the objective is to reduce the mobile device energy needed for job execution. This problem was solved for different job uploading scenarios and online algorithms were developed that employ CLE to satisfy hard task deadline constraints. The thesis shows that these online algorithms are energy optimal, in the sense that no other CLE online computation offloading algorithm can achieve a lower mean mobile device energy consumption. First, the problem is formulated as contiguous computation offloading, referred to as *1-Part offloading*, which means the entire job will be uploaded continuously in one piece without interruption. Then by constructing a time-dilated absorbing Markov chain (TDAMC), which incorporates the time progress and other offloading information into its structure, and using the theory of optimal stopping for Markov chains, an online optimal algorithm (OnOpt) is developed. This algorithm employs dynamic programming (DP) to find the optimal offload initiation time. Second, multi-part computation offloading is considered, where the job upload is segmented into multiple

parts. During task offloading, mobile device energy may sometimes be reduced by segmenting the task upload into a known number (K) of parts rather than doing a conventional contiguous task upload. We refer to this method as *K-Part offloading*. The advantage of this offloading mechanism is that the mobile device can dynamically adapt to channel conditions during the offload which may lead to lower overall energy consumption. The upload initiation time of each part is determined during the running of the algorithm using DP. The proposed online optimal algorithm to solve this problem is called *MultiOpt* which is shown to be energy-optimal using Markovian decision process stopping theory. OnOpt is a special case of MultiOpt where $K = 1$. Next, we study *preemptive offloading*, which means a decision is made to either continue offloading or to temporarily interrupt the offload at the start of each time slot. This offloading mechanism is an extension of K-Part offloading, and gives the system more capability to adapt to varying channel conditions. An online computation offloading algorithm, referred to as Optimal Preemptive Offloading (*OPO*), is formulated for preemptive offloading, and is shown to be energy-optimal. The computational complexity of OPO is prohibitive, even for simple Markovian channels, and, therefore, three computationally efficient techniques: Water-Filling, Water-Filling with Scheduling, and Generalized Water-Filling are introduced to estimate the solution of OPO. For each technique, two variations were considered. The first (Equ) uses the equilibrium channel state probabilities to determine its offloading decisions, and the second (Exp) uses Markovian transition matrix exponentiation. The six resulting algorithms have a wide variety of energy performance and computational complexity.

The proposed online optimal algorithms, especially OPO, often have a high computational complexity, which prevents their application in online mobile implementations. Three algorithms, namely, Markovian Compression (MC), Time Compression (TC), and

Preemption Using Continuous Offloading (Preemption-CO), are proposed to reduce this complexity. MC and TC reduce the state space of the offloading Markovian process by using a novel notion of geometric similarity, or by running an optimal online offloading algorithm in periodic time steps. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every time slot is based on non-preemptive calculations.

The rest of the thesis is organized as follows. Chapter 2 gives a detailed background and reviews the literature that is most related to this work. In Chapter 3, the online continuous energy-optimal computation offloading algorithm, OnOpt, is proposed. Then, Chapter 4 formulates the multi-part mobile computation offloading problem and introduces the online MultiOpt algorithm to solve this problem. In Chapter 5, preemptive mobile computation offloading is addressed, and an online optimal preemption offloading (OPO) algorithm is developed. Some heuristics are proposed to tackle the high computational complexity of OPO. Chapter 6 introduces some approximation methods that reduce the computational complexity and running time of the algorithms.

Chapter 2

Background

2.1 Introduction

In the future, mobile devices will continue to expand in their role as portable computing platforms. Unfortunately, for many applications, they are still restricted by their limited processing capabilities, storage capacity, and battery life, compared to their desktop and server counterparts. Cloud computing has emerged as a solution to this problem and is revolutionizing the world of computation. The cloud can be viewed as a distributed collection of interconnected, dynamically scalable, and virtualized computers, which are available as computing resources offered on an on-demand and pay-per-use basis. Mobile Cloud Computing (MCC) is a mixture of cloud computing and mobile computing in which mobile devices use computation offloading to exploit the power of the cloud for accelerating application execution and saving on energy consumption. This chapter provides a brief background on MC, CC, and MCC fundamentals.

2.2 Mobile Computing

The development of portable wireless computing devices, along with the evolution of fast, reliable networks, has made mobile computing a reality. Mobile Computing is the process of executing computational tasks on a mobile device and transmission of data via a wireless medium without having to be tethered to a physical communication link. Mobile computing requires mobile communication, hardware, and software, and allows people to access data and information from anywhere at any time. The types of these devices include smartphones, tablets, Laptops, eReaders (e.g., Kindle) and wearable devices (e.g., Apple Watch). Mobile computing can also be defined as the process of distributed computation on diversified mobile devices and hybrid networks interconnected by mobile communication links (De (2016)).

2.2.1 Mobile Network Architecture

A mobile network is a communication network where the link to the end mobile devices is wireless. These types of networks are often cellular as shown in Figure 2.1, where each geographic coverage area is split into wireless coverage cells, each served by at least one fixed-location base transceiver station (BTS). These base stations provide the cell with network coverage, which can be used to transmit voice, data, and other types of content. The base stations are assigned communication resources owned by the system, and typically a frequency plan ensures that interference between nearby cells is controlled (De (2016); Miao *et al.* (2016); Kamal (2008)). Mobile communications has evolved considerably in the past few decades. In cellular networks for example, five generations, i.e., 1G (first generation) to 5G (fifth generation) have been deployed. Each generation has brought with it significant milestones in the performance of these systems.

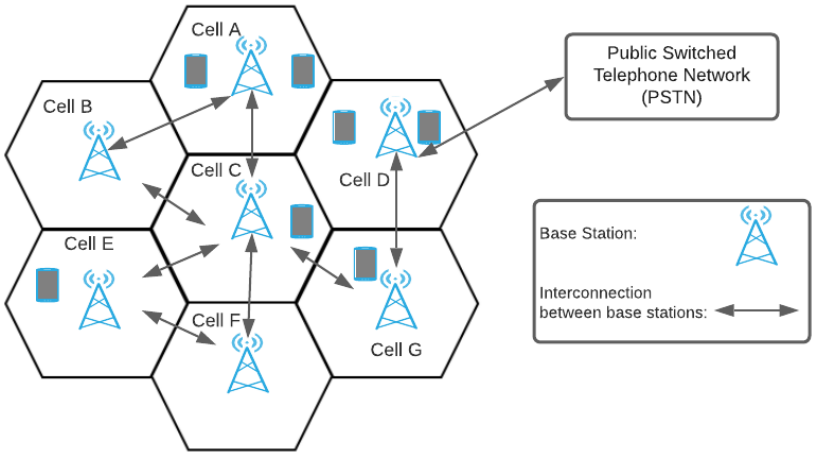


Figure 2.1: Cellular network architecture

Mobile communication is gaining considerable importance with the increasing use of portable computers and other mobile devices. Although mobile devices have attained enormous commercial success, they still have some limitations (De (2016)).

Processing power: As smartphone applications increase in complexity, so do their demand for computing resources. Despite all the advances that mobile devices have gone through in terms of hardware and computational power, they are still too slow to satisfy the computational requirements of these applications. Computation offloading is a mechanism that can be used to migrate large computations and complex processing from resource-constrained mobile devices to resource-rich cloud servers (Bangui *et al.* (2015)). This avoids lengthy application execution on mobile devices that consume a large amount of energy.

Battery Life: In the absence of power outlet connections, mobile devices rely on their batteries as the primary source of power (Forman and Zahorjan (1994)). Because mobile devices are limited in size, this places a practical restriction on the amount of energy that can be carried along with the device. Minimizing energy consumption can improve this portability by reducing battery weight and lengthening the battery lifetime.

Data Storage: The limited memory and storage capabilities of mobile devices obviously places restrictions on the size and scale of applications that can be processed locally. Solutions proposed for handling this shortcoming includes offloading the storage to some other device as facilitated by cloud computing (Mukherjee *et al.* (2013)).

Wireless Communication: Mobile computers require wireless network access for internet connectivity. Wireless communication however, is fraught with problems due to the random nature of wireless propagation. This ranges from distance dependent large scale fading and shadowing to small scale fading caused by multipath radio propagation (Shakkottai and Rappaport (2002)). This can lead to frequent link disconnection, low bandwidth accessibility, and unforeseen changes in network quality (Al-Ali *et al.* (2005)). Wireless communication is also more vulnerable than wired communication from a security viewpoint (Forman and Zahorjan (1994)).

2.3 Cloud Computing

Computer users increasingly access Internet services through lightweight portable devices, as desktop machines have become less popular (De (2016);). Cloud computing has emerged as a solution to this problem and has revolutionized the world of computation. The cloud

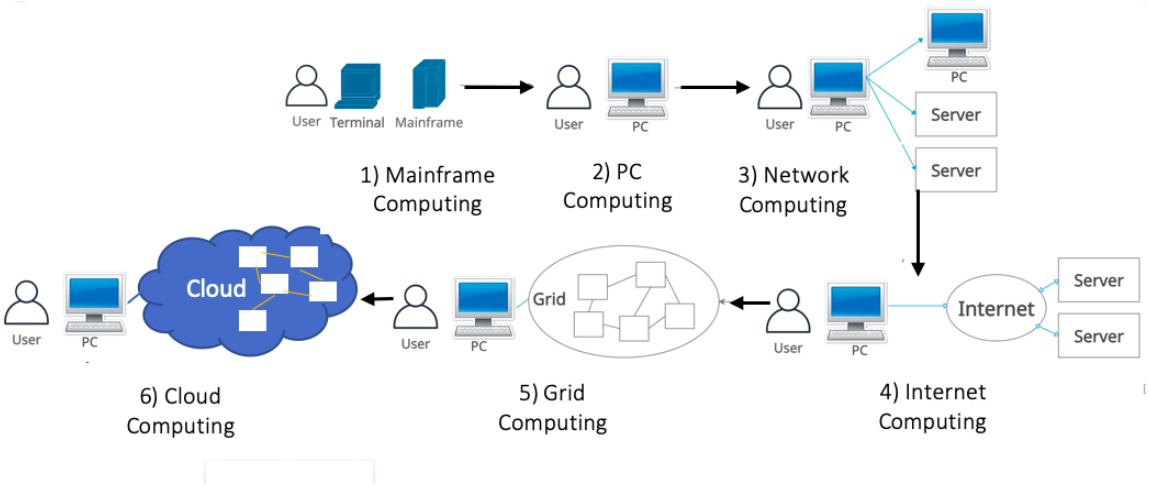


Figure 2.2: Cloud computing evolution

is a distributed computing paradigm consisting of a collection of internet interconnected, dynamically scalable, and virtualized computers, which are provisioned as computing resources offered on an on-demand and pay-per-use basis. The advantages of cloud computing technology include cost savings, high availability, and easy scalability.

The origin of the concept of cloud computing dates back to 1950 when people used terminals to connect to powerful mainframes simultaneously shared by many users (Voas and Zhang (2009)). Figure 2.2 shows six distinct computing paradigms, from dumb terminals/mainframes to grid and cloud computing (Buyya *et al.* (2013)). In phase 1, many users shared powerful mainframes using dumb terminals. In phase 2, stand-alone PCs became powerful enough to meet the majority of users' needs. In phase 3, PCs, laptops, and servers were connected through local area networks to share resources. In phase 4, local networks were interconnected, forming a global network such as the Internet to utilize remote applications and resources. In phase 5, grid computing provided shared computing power and storage through distributed computing systems. In phase 6, cloud computing further provides shared resources on the Internet in a scalable and straightforward way.

One of the main goals of cloud computing is to allow IT departments to focus on their businesses instead of expending resources on data centres and their maintenance (Barga *et al.* (2010); Huth and Cebula (2011); Kumar and Lu (2010)). A cloud computing system mainly consists of clients, data centers, and distributed servers. In cloud computing architecture, clients are the devices with which the end-users interact to manage information on the cloud. The collection of servers where the applications to which the customers subscribe is termed a data center. Typically these servers are not located at the local site but in geographically different locations, which are transparent to the end-users. This architecture also solves reliability issues such as site failure. Amazon, for example, has its servers located worldwide and if any of these sites fail, the entire system will continue functioning as usual.

The most significant characteristics of cloud computing include broad network access, on-demand self-service, resource pooling, and rapid elasticity. Broad network access allows the users to access cloud services that are available using the internet. On-demand self-service is a very attractive and valuable feature of the cloud, which permits users to access resources they need quickly and easily. Resource pooling means that the service provider's computing resources are pooled to serve multiple users, and different resources are reassigned according to user demand. Elasticity is the ability to add or remove infrastructure resources dynamically as needed to adapt to workload changes.

2.4 Mobile Cloud Computing

Despite the evolution of mobile devices, they are still limited from a computational viewpoint, as discussed in Section 2.2. Mobile cloud computing (MCC) has been introduced as a way to overcome obstacles related to mobile device performance (e.g., battery life,

storage, and computational power), environment (e.g., heterogeneity, scalability, and availability), and security (e.g., reliability and privacy) (Akherfi *et al.* (2018)). Mobile cloud computing is defined as a computing technology that controls integrated elastic resources of different clouds and network technologies in order to serve a large number of mobile devices anywhere and at any time (Sanaei *et al.* (2013)). MCC can be seen as a bridge that fills the gap between the limited computing resources of smart mobile devices (SMDs) and the processing requirements of intensive applications (Akherfi *et al.* (2018)). The Mobile Cloud Computing Forum defines MCC as follows: "Mobile Cloud Computing at its simplest form refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the resource-rich cloud servers, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers" (Dinh *et al.* (2013)).

Figure 2.3 shows the general structure of MCC, which consists of three components: Mobile network, Internet service, and Cloud service. Mobile devices are connected to the network operator via base stations which establish and control the wireless connection. Internet service plays the role of a bridge between the mobile network and the cloud where subscriber requests are delivered via wired or wireless connections. After receiving requests from the users, cloud controllers process them and cloud services will be delivered via Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Database as a Service (DaaS) functionality.

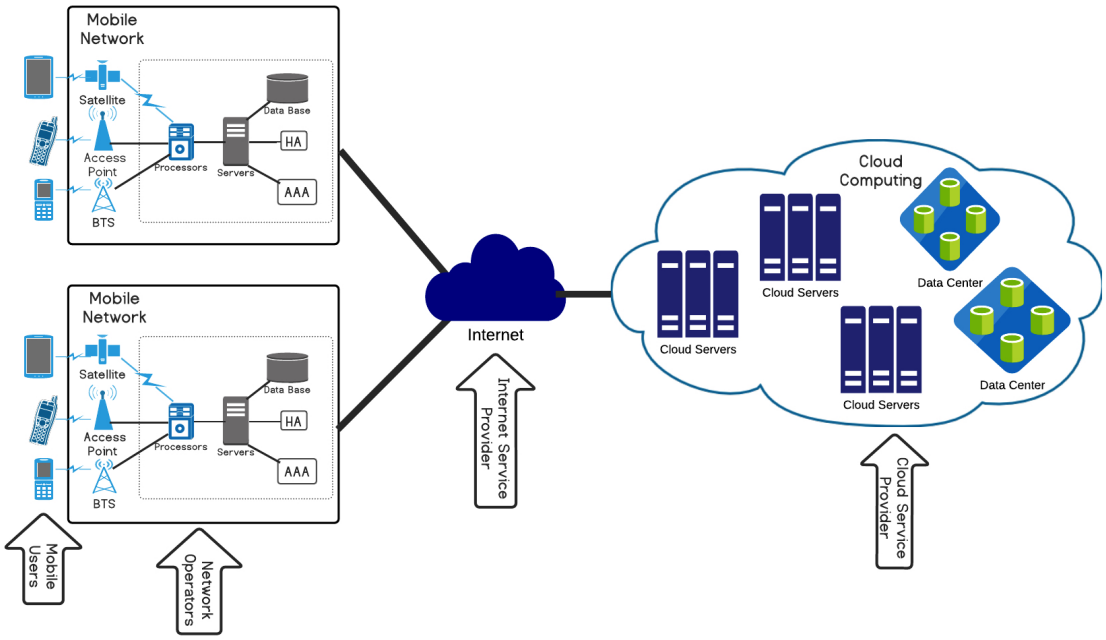


Figure 2.3: Mobile cloud computing architecture

2.4.1 Issues and Advantages of MCC

MCC is composed of MC and CC and therefore faces several challenges related to these two technologies. Mobile devices have limited energy sources to transmit and receive data to and from the cloud. In addition, since offloading typically occurs over wireless transmission channels, connection issues and low bandwidth communication channels can have an adverse effect on application response times and the energy usage of the mobile devices. As in many wireless networks, security is another major issue in mobile cloud computing.

Despite the challenges mentioned above, there are several promising advantages that can be gained using MCC. In MCC, processing happens outside the mobile device, in more powerful cloud servers. This reduces the mobile device energy consumption for task processing. Since the SMD can be relieved from the processing of heavy computational

tasks, its battery energy can be preserved, and it can be utilized for more extended periods. Also, by offloading the computational load to more powerful remote hosts, applications can benefit, which can lead to processing delay reduction. Furthermore, by enabling remote execution, more applications can now be supported, leading to an increase in the SMDs usefulness. As pointed out earlier, storage is one other mobile device constraint, and this can be overcome by MCC. Amazon's simple storage service and Dropbox are examples of cloud services that provide cloud storage. MCC also has improved reliability, by storing data and applications in redundant locations. Apart from this, the cloud provides security services such as virus scanning, authentication and malicious code detection. Another benefit of MCC is on-demand services which are accessible by mobile users in a "pay as you use" fashion and without having to install dedicated hardware or software.

2.5 Mobile Computation Offloading

MCC addresses the deficiencies of mobile device execution resources by performing mobile computation offloading (MCO). In MCO, computational tasks are offloaded to more powerful remote machines. Cloud service providers, such as Amazon Elastic Cloud Compute (EC2), and Windows Azure, offer several classes of virtual machines (VMs) with different processing/storage capabilities. A user can purchase a VM that best accommodates its requirements, and heavy computational tasks can be offloaded for execution. In what follows we discuss some important features of MCO.

A smart mobile device (SMD) can benefit from cellular or Wi-Fi connectivity to conduct offloading (Chen *et al.* (2017); Pan *et al.* (2017); Pu *et al.* (2016)). WLAN-based offloading typically incurs relatively low energy and latency costs. Unfortunately, wireless access point (WAP) availability is often intermittent, leaving cellular as the alternative,

which often suffers from low bandwith and varying channel conditions.

In general, offloading schemes can be classified as either delayed or on-the-spot methods (Mehmeti and Spyropoulos (2013)). In the first case, when a Wi-Fi link is unavailable, traffic can be delayed until the situation changes. If no WAP is detected up to some chosen time threshold, the offloading can be conducted using the cellular network. Postponing the offloading process in this way may be possible depending on the allowable task completion deadline. In the second case, when there is no Wi-Fi availability, offloading will be performed immediately using cellular communications.

In mobile computation offloading, the task execution completion time is dependent on the data transfer delay over the wireless link. Therefore, the data transmission rate and wireless channel status may have a significant impact on offloading decisions (Panigrahi *et al.* (2018); Hu *et al.* (2019)). In general, the task offloading duration includes the following items:

- Upload time of the data, code, or computation/application.
- Execution time in the offloading cloud server destination.
- Download time to return the execution results to the SMD.

Regarding delay in these components, the offloading operation can be categorized as either delay-sensitive or delay-tolerant (Liu *et al.* (2017); Wu *et al.* (2019)). Generally, in delay-tolerant applications, response time is not critical and power consumption is more important. In delay-sensitive applications however, low response time is an essential factor. Delay-tolerant applications with relatively large execution time deadlines are suitable for offloading using the delayed offloading method. On the other hand, delay-sensitive applications with tight application response time deadlines are the best candidates for on-the-spot

offloading. Offloading decisions can also be made either statically or dynamically. In the first case, decisions are made during program development, while in the second case, they can be made during program runtime.

In this thesis, we consider the offloading of delay-sensitive jobs where the task execution completion times are subject to hard time deadline constraints. It is assumed that a deadline is assigned to each computational job at the release time of the job, and the task execution results must be available at the mobile device before the deadline expires. It is assumed that the mobile device performs computation offloading through a wireless network with time varying link quality. The stochastic communication channels are modelled as discrete-time homogenous Markov chains, which transition amongst states of different channel conditions. It is also assumed that the uplink channel uses bit rate adaptation to accommodate random variations in channel conditions. Therefore, task uploading time is a random variable that depends on the evolution of the uplink channel state. However, we assume that transmit power control is used on the downlink, and therefore, downloading time as well as the remote execution time are deterministic. Offloading decisions are made dynamically during the runtime of the program.

2.6 Related Work

In general, the previous research conducted in the area of mobile cloud computing can be divided into two main categories. The first category is the offloading mechanism which focuses on implementing the function of application offloading. The second is the offloading policy to decide whether to offload applications from the mobile devices to the cloud and when to do so. The offloading mechanism has been studied extensively (Chun and Maniatis (2009); Chun *et al.* (2011); Satyanarayanan *et al.* (2009); Huerta-Canepa and Lee (2010)),

but research efforts on offloading policy are still limited. In this thesis, we will focus on developing optimal offloading policies to minimize the energy consumption of the mobile device.

Many mobile cloud computing issues and challenges regarding mobile computation offloading have been addressed in the past few years (Abolfazli *et al.* (2013); Othman *et al.* (2013); Liu *et al.* (2013); Guan *et al.* (2011)). To overcome MCO challenges, different influencing factors such as network bandwidth, mobility (Yu *et al.* (2018)), heterogeneity, security (Meng *et al.* (2018)), and QoS, cost and deadline constraints, should be considered. For this purpose, various techniques such as multi-criteria decision making (Nădăban *et al.* (2016); Zhang *et al.* (2017b)), optimization algorithms (Shuja *et al.* (2017)), game-theoretic models (Tao *et al.* (2017); Meskar *et al.* (2015)), and stochastic modelling have been applied to make optimal offloading decisions. The latter is the focus of this thesis.

Mobile computation offloading is often a challenging problem (Masdari and Khezri (2020)). In computation offloading, mobile execution energy can be decreased, but is offset by an increase in the communication energy needed to interact with the cloud server. Offloading will also incur additional latency that would not exist otherwise due to the time needed to exchange application data with the server. This latency may be partially compensated for by a faster task execution time at the cloud server. The basic tradeoffs involving these attributes and how they relate to the decision to offload task execution have been studied extensively (Wu and Wolter (2017); Mehmeti and Spyropoulos (2016); Zhang and Cao (2018); Zhang *et al.* (2015); Kim *et al.* (2016)).

Many early mobile computation offloading papers tended to take a static view of the communication channels. In (Kumar and Lu (2010)), the authors model the communication energy using statistical inputs, but the wireless channel is assumed to be static throughout

the offload. Reference Geng *et al.* (2018) addressed the problem of computation offloading on multicore-based mobile devices running multiple applications. They proposed a novel heuristic algorithm for solving an NP-hard problem that tries to find the offloading decision and task scheduling by prioritizing the tasks in order to meet the completion time constraint and task dependencies. This paper considers a static wireless channel with a fixed bitrate. This assumption, of course, may not always be the case, and in more realistic scenarios, mobile devices have to interact with the cloud servers over stochastic wireless communication channels. A significant part of the literature has considered mobile computation offloading issues under stochastic transmission channel and cloud server conditions (Zhang *et al.* (2013a); Zhang *et al.* (2014); Zhang *et al.* (2017b); Goudarzi *et al.* (2017); Oo *et al.* (2016b); Oo *et al.* (2016a)).

When computation offloading occurs over a stochastic wireless channel, the issue of application response time becomes important and should be taken into account. Under these circumstances, making optimal offloading decisions is particularly difficult when task execution times must satisfy *hard* time deadline constraints. The importance of task execution time constraints as an essential criterion for many interactive applications was highlighted in (Lagar-Cavilla *et al.* (2007)). This paper also discussed the difficulty of achieving this under random wireless channel conditions.

Various stochastic offloading schemes are provided in the literature. In what follows we study those that are most relevant to this thesis. All of these offloading schemes benefit from one or more stochastic models to help them in making better offloading decisions.

Markov chains have been routinely used to model stochastic wireless channels, and, as is often assumed, the Markovian transition probabilities are taken to be known, or have been learned dynamically (Gilbert (1960); Elliott (1963); Zhang *et al.* (2014); Zhang *et al.*

(2013a); Zafer and Modiano (2007); Johnston and Krishnamurthy (2006)). Several Markov chain-based stochastic offloading schemes are addressed in this subsection.

In (Zhang *et al.* (2013a)), the problem of mobile computation offloading under a stochastic wireless channel is considered. According to their proposed system model, the entire application can be executed either on the mobile device (i.e., local execution) or on the cloud servers (i.e., remote execution). The design objective is to develop an optimal application-execution policy which minimizes the energy consumed by the mobile device. This work used mobile CPU frequency scheduling and transmit power control to compensate for random wireless channel effects, so that remote task offload latencies can be controlled. When the application is executed in the mobile device, the computation energy can be minimized by optimally scheduling the CPU clock frequency of the mobile device via Dynamic Voltage Scaling (DVS). When the application is executed in the cloud servers, the transmission energy can be minimized by optimally scheduling the data transmission rate. They formulated these scheduling problems as convex optimization problems, with a constraint that the application should meet a time deadline with a certain probability. When the application execution fails to meet its time deadline, the mobile device will continue to execute at the maximum clock frequency for task completion. This approach cannot always ensure that task execution deadlines are met, but a parameter was introduced that controls the probability that task deadlines can be violated.

Energy optimal cloud computing was addressed in (Zhang *et al.* (2014)) which included a treatment of task execution time deadlines assuming random wireless channels. The offloading scheme provided in this work presents a scheduling policy for collaborative execution in mobile cloud computing. Each mobile application is partitioned into a sequence of tasks. Every task is then processed either locally on the mobile device or

remotely on the cloud servers. The design objective is to minimize the energy usage of a mobile device while meeting a time deadline. CPU frequency scheduling was used to control mobile task execution time delays when remote offloading occurs under random wireless channel conditions. This was combined with mobile transmit power control, to target task deadlines. However, since there are limits on radio transmit power and CPU frequency, satisfying execution time constraints is not always possible. For this reason, a violation parameter was used to characterize execution time deadline misbehaviour. This reference assumed the well-known Gilbert-Elliot Markovian channel model. They modeled the minimum-energy task scheduling problem as a constrained stochastic shortest path on a directed acyclic graph (DAG) and applied the canonical Lagrangian relaxation based aggregated cost (LARAC) algorithm to find the approximate solution to the problem. The authors concluded that a one-climb offloading policy is energy-efficient for the Markovian stochastic channel, in which at most one migration from the mobile device to the cloud takes place for the collaborative task execution. They showed that compared to standalone mobile execution and cloud execution, the optimal collaborative execution strategy can significantly reduce the energy consumed on the mobile device. Although task completion time deadlines were addressed in this work, their observance was considered statistically rather than as a hard constraint which has to be satisfied.

In (Wu and Wolter (2017)), the authors modeled the wireless network availability using a Markov chain. Applications, based on their response time, can be divided into two categories, namely: Delay-Tolerant Applications and Delay-Sensitive Applications. Delay-Tolerant Applications such as iCloud, social networking, and mobile healthcare are less sensitive to network delays. However, Delay-Sensitive Applications such as face recognition, video conferencing, vehicular communications, and authentication require a fast

response time comparable to typical human cognitive capabilities. The presented stochastic offloading method in this reference supports two different delayed offloading policies. The first scenario is called a partial offloading model, where jobs can leave the slow phase of the offloading process to be executed locally. The second scenario is called a full offloading model, where jobs can leave the Wi-Fi queue to be offloaded directly through the cellular network. In both scenarios the objective is to reduce the energy-response time-weighted product (ERTP) metric. For delay-sensitive applications, the partial offloading model is preferred, while for delay-tolerant ones with a more extended deadline, the full offloading is a more efficient choice. The authors developed queuing models for delayed offloading to leverage the Wi-Fi and cellular networks by choosing wireless interfaces for offloading. In their simulations, they considered the intermittently available links and found that when the Wi-Fi availability is low, the portion of jobs that abandon the queue is very high. For delay-sensitive applications, the partial offloading model is preferred for medium deadlines, while the full offloading model is better for delay-tolerant applications and can decrease energy consumption. The authors indicated that an optimal deadline to abort offloading in the partial offloading model and for the Wi-Fi in the full offloading model could be found. In both partial and full offloading policies the transmission rate of the Wi-Fi and the cellular networks are assumed to be fixed numbers which is not the case in real world scenarios and therefore satisfying a hard job deadline can not be guaranteed.

In (Wu *et al.* (2015)), the authors proposed two strategies for mobile cloud offloading and analyze them through analytic queuing models. The first one, called the interrupted method, chooses the best interface (e.g., cellular or Wi-Fi transmission) to send packets and interrupt the Wi-Fi connections for short periods, causing delays for packets. The second method, which is called the uninterrupted strategy, uses Wi-Fi (which is assumed to have

the best transmission characteristics) whenever possible but switches to a cellular interface if no Wi-Fi connection exists. The authors used 2D Markov chains to model the Wi-Fi network availability in both the uninterrupted and interrupted offloading methods and a regular 1D Markov chain to model the offloading through the cellular network. These two models are compared using various previously studied energy-response time tradeoff metrics such as Energy-Response Time Weighted Sum (ERTWS) and the Energy-Response Time Product (ERTP). Then they introduced the Energy-Response Time Weighted Product (ERTWP) metric, which combines the advantages of both previously studied metrics. According to their simulations, the interrupted strategy can save energy, especially if the tradeoff metric's focus lies in the energy aspect. Generally speaking, one can say that the uninterrupted strategy is faster, while the interrupted strategy is more energy efficient. A fast connection improves the response time much more than the fast repair of a failed connection. In conclusion, a short downtime of the transmission channel can mostly be tolerated, especially in delay-tolerant applications with more extended deadlines.

The stochastic framework presented in (Mehmeti and Spyropoulos (2016)) focuses on on-the-spot offloading as opposed to delayed offloading, which is discussed in previous references. On-the-spot offloading means when there is Wi-Fi availability, all data is sent over Wi-Fi; otherwise, traffic is transmitted over the cellular network. The authors have used a 2D Markov chain to proposed a queueing analytic model for the performance of on-the-spot mobile data offloading for a number of access technologies, and they validated it against realistic Wi-Fi network availability statistics. In this reference, Wi-Fi connectivity's availability ratio plays a crucial role in offloading performance in conjunction with the arrival rate. When Wi-Fi is not available at the release time of the job, the entire computational task will be transmitted through the cellular network, and it never switches back

to Wi-Fi, if Wi-Fi becomes available partway through the uploading. This makes their proposed method less energy-efficient. Furthermore, there is no local execution, and therefore in the case of low-quality network connections or complete channel outage, hard time deadlines can not be met.

The scheme introduced in (Zhang and Cao (2018)) proposed a hierarchical data offloading solution, which employed different offloading options with various priorities. To model the network availability, they used a 4-state Markov chain with a state-space $X = \{W, B, D, C\}$ which refers to offloading through Wi-Fi, base station, D2D, and cellular network, respectively. In their algorithm, they give the highest priority to offloading through Wi-Fi. The maximum delay mobile users can tolerate is specified as the delay-tolerance threshold. When the delivery deadline approaches the threshold, the currently paired offloading connection will be given up by the user, and all traffic will be switched back to the cellular network. However, with their proposed hierarchical offloading scheme, before totally falling back to cellular mode, alternative offloading options will be offloading through BS and D2D, respectively. They have shown that by prioritizing Wi-Fi over base station offloading, backhaul traffic load can be reduced.

Although studies have shown that computation offloading improves energy efficiency significantly, they mostly neglect the adverse effects of network disconnections. In (Berg *et al.* (2014)), the authors proposed a preemptive method for code offloading to improve energy efficiency under link failures. It transmits safe-points between server and mobile device during remote execution, enabling the re-use of partial remote results after link failures. The authors improve the time to create and transmit safe points to decrease the messaging overhead and maximize energy efficiency. They applied a predictive method that estimates the wireless link quality to send safe-points before any network's disconnection.

They consider task execution deadline constraints and show that this preemptive offloading method maximizes the probability of satisfying this deadline despite link failure. The evaluation results show that energy efficiency can be improved significantly using the predictive offloading approach. In this work, task completion time deadlines were considered statistically rather than as a hard constraint which has to be satisfied.

In (Wu *et al.* (2015)), the authors considered a delayed offloading model for leveraging the complementary advantage of cellular networks and Wi-Fi when selecting heterogeneous wireless interfaces for offloading. In this model, by postponing transmission until a fast and energy-efficient network (e.g., Wi-Fi) becomes available, it is possible to reduce the transmission time even if the extra waiting time is introduced, which leads to energy savings. A soft deadline T_d is assigned to each job in the slow phase. That is, each job, upon arrival, activates an individual "impatience timer," which is exponentially distributed. If the Offload Queue job is completely transmitted before the associated deadline has expired, we say that the job is successfully offloaded. If the system does not change its environment from the slow phase to the fast phase before the deadline expires, the job will be removed from the Offload Queue and join the Local Queue for immediate local processing. Therefore each task will be executed either solely locally or solely remotely. Optimality analysis of the power-delay balance is carried out using a queuing model with service interruptions and delay-sensitive workflows, which considers energy and performance metrics and intermittently available links.

The offloading framework presented in (Ko *et al.* (2017)) introduces an online prefetching method rather than conventional mobile computation offloading, which relies on offline prefetching. For computations depending on real-time inputs, the offline operation can lead

to fetching large volumes of redundant data over wireless channels, which results in unnecessary consumption of mobile-transmission energy. To address this issue, the authors proposed a technique of online prefetching for a large-scale program with numerous tasks, which seamlessly integrates task level computation prediction and real-time prefetching within the program runtime. It decreases the SMD's power consumption by preventing unnecessary prefetching and reduces the applications' execution time through parallel fetching and computing. By modelling the sequential task transition in an offloaded program as a Markov chain, stochastic optimization is used to design the online-fetching policies to minimize mobile energy consumption for transmitting fetched data over fading channels under a deadline constraint.

In (Gao *et al.* (2014)), the authors proposed to adaptively offload the computational load with respect to the run-time application dynamics. As opposed to conventional schemes that may inappropriately increase the energy consumption by transmitting a large number of program states over expensive wireless channels, the proposed approach can avoid wasting the energy by considering the dynamic patterns of applications' run-time execution for workload offloading. Their approach is to formulate the dynamic executions of SMD applications by utilizing a semi-Markov model. The semi-Markov process is a generalization of the Markov chain where the sojourn times (residence time) in the states are random variables, whose distribution function may depend on the two states between which the move is made (Barbu and Limnios (2009)). They make offloading decisions according to probabilistic estimations of the offloading power conservation. They introduced analytical modeling of the processing time and offloading cost. Task execution deadline constraints were not addressed in this reference.

The stochastic offloading scheme in (Tong and Gao (2016)) investigated the balance

between the power usage and performance of the SMDs through application-driven transmission scheduling. Wireless data transmissions incurred by remote execution consume a large amount of energy during transmission intervals when the network interface stays in the high-power state. On the other hand, deferring these transmissions increases the response delay of mobile applications. Traditional approaches support workload offloading through appropriate application partitioning and remote method execution but generally ignores the impact of stochastic wireless network characteristics on such offloading. In this reference, the authors dynamically adjust the tradeoff between energy efficiency and responsiveness of mobile applications by developing application-aware wireless transmission scheduling algorithms. They take both causality and run-time dynamics of application executions into account when deferring wireless transmissions to minimize the wireless energy cost and satisfy a soft constraint with respect to the practical system contexts.

A Markov decision process (MDP) is a well-known discrete-time mathematical framework applied for modeling decision making with uncertainty. An MDP model contains items such as decision epochs, states, actions, transition probabilities, and costs. In addition, an MDP is able to determine agents' actions and their intersections with the environment. Several algorithms such as the value iteration algorithm, policy iteration algorithm, and linear programming are used to solve the MDP models. However, the computational complexity of an MDP model increases with the number of states and the action spaces.

In (Zhang *et al.* (2015)), the authors presented an optimal offloading solution for the SMD to address the offloading failure caused by wireless connection deterioration and users mobility. They propose a MDP based offloading algorithm for mobile users in a cloudlet system. They obtain success probabilities of offloading actions with limited knowledge of network parameters. These probabilities are used in the MDP to obtain an optimal

offloading policy. The mobile user has an application to be executed. As the application is divided into code sections (referred to as phases), the mobile user can dynamically decide to execute application phases locally on the mobile device or offload to nearby cloudlets during the execution run-time. They formulate and solve the MDP model to obtain an optimal offloading policy to minimize computation and communication costs as well as job completion delays. Their proposed optimal offloading policy of the MDP has a threshold structure. They then introduced an algorithm with bounded errors for the mobile user to make offloading/local execution actions based on the threshold policy.

In (Zhang *et al.* (2017a)), the authors studied the mobile user's (MU's) policy to minimize their monetary cost and energy consumption under time-dependent pricing when choosing whether to offload their traffic from the cellular network to a complimentary wireless LAN. They formulated MU's wireless LAN offloading problem as a finite-horizon discrete-time Markov decision process and developed an optimal policy using a dynamic programming-based algorithm. Although simulation results showed that the proposed dynamic programming based offloading algorithm is effective under time-dependent pricing conditions, they are mainly focused on applications with data of relatively large size and delay-tolerance to download, for example, software updates.

In (Labidi *et al.* (2015)), the objective is to minimize the average energy consumption at the user terminal when running its mobile applications, either locally or remotely, while satisfying the mean delay constraints tolerated by these applications. They formulated this problem as a Constrained Markov Decision Problem (CMDP) and used dynamic programming to obtain offline solutions. The offline dynamic strategies can benefit from the prior knowledge on the application rates and the channel statistics to perform offloading decisions in good channel states and local processing or staying idle under bad channel

conditions. Since their proposed approach is offline, it is not able to cope with sudden changes in the propagation environment. Furthermore, since their optimization constraint is satisfied fron a statistical viewpoint their proposed approach can not guarantee a hard task execution deadline.

The approach provided in (Terefe *et al.* (2016)) presents a model to describe the energy consumption of multisite application execution. They used a two-state discrete-time Markov chain (DTMC), which transitions between a good and a bad state to model fading wireless mobile channels. The proposed DTMC is applied to compute the average transmission bit-rate of the communication channel. The authors adopt an MDP framework to formulate the multisite partitioning problem as a delay-constrained, least-cost shortest path problem on a state transition graph. They proposed the Energy-efficient Multisite Offloading Policy (EMOP) algorithm, built on a value iteration algorithm (VIA), which finds the optimal solution to the multisite partitioning problem. Note that their optimization problem is solved under some expectation of the channel state. As a result, their proposed approach can not tackle unstable channel conditions and offloading decisions become suboptimal and less energy-efficient when channel states rapidly transition between good and bad states.

The framework provided in (Liu *et al.* (2017)) studies the mobile data offloading problem under the architecture of mobile cloud computing and stochastic wireless channels, where mobile data can be delivered by cellular, Wi-Fi, and Device-to-Device (D2D) communication networks. Their objective is to optimize task offloading cost by reducing cellular network usage while satisfying deadline constraints. In the proposed model, a portion of the cellular data traffic is offloaded through D2D and Wi-Fi links. The authors formulated

the data offloading problem as a finite horizon Markov decision process (FHMDP). Although they solve the problem using a hybrid offloading algorithm for both delay-sensitive and delay-tolerant applications, there may be some data transmission tasks that cannot be completed before the deadline. For failed data transmissions, they set a penalty cost function, which by optimizing that, they maximize the ratio of the offloaded tasks.

In (Hyytiä *et al.* (2015)), the authors considered an MDP-based model to investigate dynamic offloading in the MCC. According to this reference, computational tasks can be classified into three types: (i) those that should be executed only locally in an SMD, (ii) those which are processed in the cloud, and (iii) those which can be processed either locally or remotely. For type (iii) tasks, the main concern is when they should be processed locally and in the cloud. Furthermore, for both type (ii) and (iii) tasks, there are typically two ways to access the cloud: through an expensive cellular connection or via intermittently available Wi-Fi local area network (WLAN) hotspots. The optimal strategy involves multi-dimensional considerations such as the availability of WLAN hotspots, energy consumption, communication costs, and expected delays. This challenging problem is addressed in the framework of Markov decision processes and queueing theory to derive a sub-optimal offloading policy concerned with the various performance metrics. Application response time constraints are analyzed from a statistical viewpoint rather than a hard constraint.

The offloading strategy provided in (Truong-Huu *et al.* (2014)) introduced a dynamic opportunistic offloading solution that enables the user to decide about offloading or deferring. In their work, they assume that the task offloading is performed only when cloudlets are in the short-range network area of the user's device. When cloudlets move out of range, the offloading is considered as failed, and the mobile user has to re-offload to another cloudlet or process the failed task on his own device and incur a penalty cost for failed

offloading. The authors presented an MDP model that enables mobile users to achieve an optimal offloading policy while decreasing the execution and offloading costs. However, their proposed offloading policy can only meet a long required offloading deadline, and it is not suitable for applications with short task processing deadlines.

In (Zhang *et al.* (2016)), the authors studied the complimentary Wi-Fi offloading problem from mobile users' point of view by considering delay-tolerance of traffic, monetary cost, energy consumption, as well as the availability of user's mobility pattern. They first formulate the Wi-Fi offloading problem as a finite-horizon discrete-time Markov decision process (FDTMDP) with a known mobility pattern and propose a dynamic programming-based offloading algorithm. Since the user's mobility pattern may not be known beforehand, they proposed a reinforcement learning (RL) based offloading approach, which can work well with unknown mobility patterns. Since their proposed approach cannot guarantee a firm task execution deadline constraint, in their problem formulation they have defined a penalty for the mobile user if the data transmission is not finished within a deadline.

The work in (Kim *et al.* (2016)) defines the problem of multi-flow offloading in which an SMD has some traffic flow with various deadlines and loads. They considered the multi-flow rate control as a FDTMDP. They develop a DP-based optimal rate control algorithm to maximize user satisfaction defined as offloading efficiency minus disutility due to deadline violations. As an advantage, their solution supports delayed offloading. However, their proposed framework cannot guarantee a hard task execution deadline.

The stochastic scheme introduced in (Mao *et al.* (2016b)) presented a Lyapunov optimization-based dynamic computation offloading (LODCO) approach to make optimal offloading decisions in order to minimize the energy consumption of the mobile devices using dynamic voltage and frequency scaling (DVFS). In their proposed approach, to satisfy

the deadline constraints, they use the CPU-cycle frequency adaption for mobile execution and transmit power adaption for computation offloading. Therefore, the algorithm's implementation requires solving a deterministic problem in each time slot, for which the optimal solution can be obtained. To keep their problem feasible, they have assigned a minimum required energy (E_{min}) for each application with a specific deadline constraint. If the available energy on the mobile device becomes less than E_{min}, the job execution deadline cannot be satisfied. Therefore, their approach does not guarantee a strict deadline constraint.

As discussed in this chapter, meeting strict task execution deadline constraints becomes a very challenging problem when offloading occurs over stochastic wireless channels. In the next chapter, the issue of satisfying hard task deadline constraints for continuous MCO is addressed.

Chapter 3

Continuous Offloading

3.1 Introduction

This chapter studies *continuous computation offloading* when the job uploading occurs uninterruptedly in one piece over a stochastic wireless channel, and job completion times are subjected to hard deadline constraints. This offloading mechanism is referred to as *1-Part Offloading*. Stochastic wireless channels are modelled as generalized Markovian processes. To address this problem an online optimum (*OnOpt*) computation offloading algorithm is proposed, which uses CLE to satisfy hard job deadline constraints. It is shown that OnOpt is optimum from a mean energy viewpoint, which means no other online 1-Part offloading algorithm can achieve lower mean energy while satisfying a hard deadline constraint. This is proven by augmenting the underlying channel model so that it forms a TDAMC. Dynamic programming is then used to establish a test that determines whether a given job should be offloaded at the current time or to wait for some future offload opportunity. The performance results presented use the Gilbert-Elliott channel model. In this case, closed-form results are derived that are used to test for optimal offload initiation

times. The job completion time probabilities can be computed recursively, and this results in a significant reduction in the computational complexity of the proposed algorithm. The performance of the proposed algorithm is compared to three simpler heuristics, namely, offload immediately (*Immediate Offloading*), wait until the channel condition improves to above a threshold (*Channel Threshold*), and execute the job only locally (*Local Execution*). An integer linear program (ILP) is formulated that computes an offline lower bound on mobile device energy consumption. This bound is used for comparisons in the performance results. Performance results show that the proposed algorithm can significantly improve mobile device energy consumption compared to the other approaches.

The rest of this chapter is organized as follows. Section 3.2, describes the system and presents a model for local and remote job execution that satisfies hard job execution deadline constraints. Then, in Section 3.3, an offline lower bound on the energy consumption is derived, which is plotted in the results section and compared to various offloading algorithms. Section 3.4 discusses the generalized Markovian channel model and how it is used to form a TDAMC. This construction allows to apply DP and come up with the energy optimum online algorithm OnOpt, proposed in Section 3.5. Then, in Section 3.6 we focus on the Gilbert-Elliot channel model, where it is shown that calculations can be performed efficiently, decreasing the complexity of OnOpt. Finally, in Section 3.7, performance results are presented that compare OnOpt with various other computation offloading algorithms that ensure that hard job deadlines are preserved.

3.2 System Model

A mobile device generates computational tasks (or jobs) that can either be executed locally, or can be offloaded over a stochastic communications channel to a cloud server for

remote execution. Following a common convention, the stochastic communication channel is modelled as a Markov Chain, which transitions amongst states of different bit-rates in every time-slot. It is assumed that the Markov channel model is known to the mobile device. We consider mobile computation offloading when CLE is allowed to enforce task execution time constraints. Our discussion will focus on single job offloads, but each job could be a sub-task associated with multiple local/remote job execution components (You *et al.* (2016); Chiang and Zhang (2016)). Figure 3.1 illustrates 1-Part offloading, where the entire job is uploaded continuously in one piece starting at t_o which is the optimal remote offload initiation time. In this model 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. The time slot duration is defined to accommodate the channel propagation model discussed in Section 3.4 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. This is marked on the left side of Figure 3.1. For convenience, we will assume that $t_R = 1$ (Since offloading is only affected by differences between time values in Figure 3.1, and not by the time values themselves, this assumption can be made without loss of generality).

t_D**:** *Hard deadline* of the job, i.e., the job execution results *must* be available at the mobile device by time t_D. This is shown on the right side of Figure 3.1, where $T_D = t_D - t_R + 1$ is the maximum number of time slots available for completing the job.

D**:** Number of local CPU cycles needed in order to execute 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.

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

3.2.1 Local Execution

Using the model described in (Kumar and Lu (2010)), i.e., the local execution energy consumption of a task to be processed is determined by its CPU workload, we assume that the *local execution energy consumption* E_L, and the *amount of time to complete local execution* T_L, are known at the time of task generation. So, while the task release times are random, they are known tasks and therefore E_L and T_L are deterministic. 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); Meskar *et al.* (2017); Meskar *et al.* (2015); Josilo and Dan (2017)). If the computation offloading algorithm elects to execute the job locally without any remote offloading, then the energy use of the mobile device is equal to E_L. We must ensure that the job deadline is always satisfied. Therefore, local execution must start no later than

$$t_L = t_D - T_L + 1, \tag{3.2.1}$$

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 the mobile device is still awaiting a remote response (Recall that time is quantized into time slots whose duration is normalized to 1, and since the local execution duration, T_L, must start from the beginning

of time slot t_L, 1 is added to the right hand side of (3.2.1)). In Figure 3.1, t_L is shown to occur before the remote execution cycle has completed and therefore local execution has started, i.e., CLE, as shown in orange. 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.

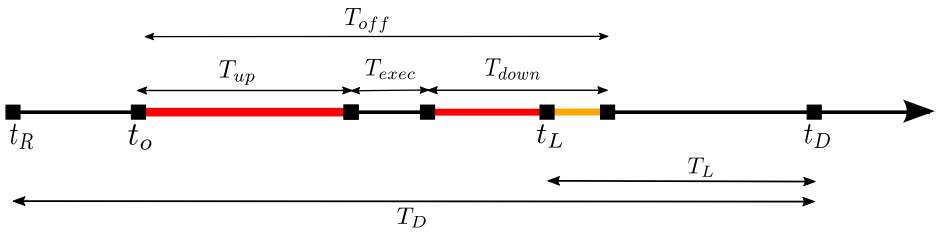


Figure 3.1: Concurrent local and remote 1-Part computational job offloading time line

3.2.2 Remote Execution

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 a fixed amount and 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 common in previous work on offloading, e.g., (Kumar and Lu (2010); Chen (2015); Cao and Cai (2018)). 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

$$T_{off} = T_{up} + T_{exec} + T_{down}. \tag{3.2.2}$$

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 during task upload. 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 $T_{exec} = T_{down} = 0$. Given the ensuing results, adding the effects of T_{exec} and T_{down} is straightforward and is deferred to Section 3.6. At that point we discuss, for example, how we include the effects of a random Markovian downlink channel. Generally speaking, in mobile computation offloading applications, since the mobile device usually uploads more data than it downloads, and moreover, the remote machine does not have limits on transmission power, T_{down} is much smaller that T_{up} (Masdari and Khezri (2020)).

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 the expected 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 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, described in detail in Section 3.4.

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.

3.3 Offline Bound

In this section, an offline lower bound on mobile device energy consumption for continuous offloading is derived. This lower energy bound is used in Section 3.7 for performance comparisons with various online computation offloading algorithms proposed in this chapter. 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 initiation offload

[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.

time so that its deadline is met and the energy needed is minimized. 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 ILP,

$$\min_{t_o} \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \tag{3.3.1}$$

$$s.t. \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \leq E_L \tag{3.3.2}$$

$$1 \leq t_o \leq t_D. \tag{3.3.3}$$

Objective (3.3.1) 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 (3.3.1) 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

$$e_t = \begin{cases} E_{tr}, & t < t_L \\ E_{tr} + \frac{E_L}{T_L}, & t \geq t_L \end{cases} \tag{3.3.4}$$

where E_{tr} is the energy cost per time slot for transmitting on the wireless channel. Constraint (3.3.2) ensures that the energy used in offloading does not exceed that of executing the job locally. Note that if the ILP is infeasible, then there is no feasible offloading start time t_o, i.e., it is best to execute locally without offloading.

3.4 Markovian Channel and the Time-Dilated Absorbing Markov Model

In many studies, homogeneous Markov chains have been used to model random wireless channel conditions (Gilbert (1960); Elliott (1963); Zhang *et al.* (2014); Zhang *et al.* (2013b) 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 Section 3.5 is an online computation offloading algorithm that attains the minimum expected execution energy. In this section we use the conventional channel state Markov chain (CSMC) to form a 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. 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 $P_{i,j}$ 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 offloading information into its structure.

The TDAMC models the job offloading progress if the offloading 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_{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 3.2, 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 3.2, 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., 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 3.2, 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 3.2, 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 set of absorbing states and $\mathcal{T}$ to be the set 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):

$$\mathbb{P}_{\text{TDAMC}} = \begin{bmatrix} Q & R \\ \mathbf{0} & I_{\mathcal{A}} \end{bmatrix}. \tag{3.4.1}$$

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

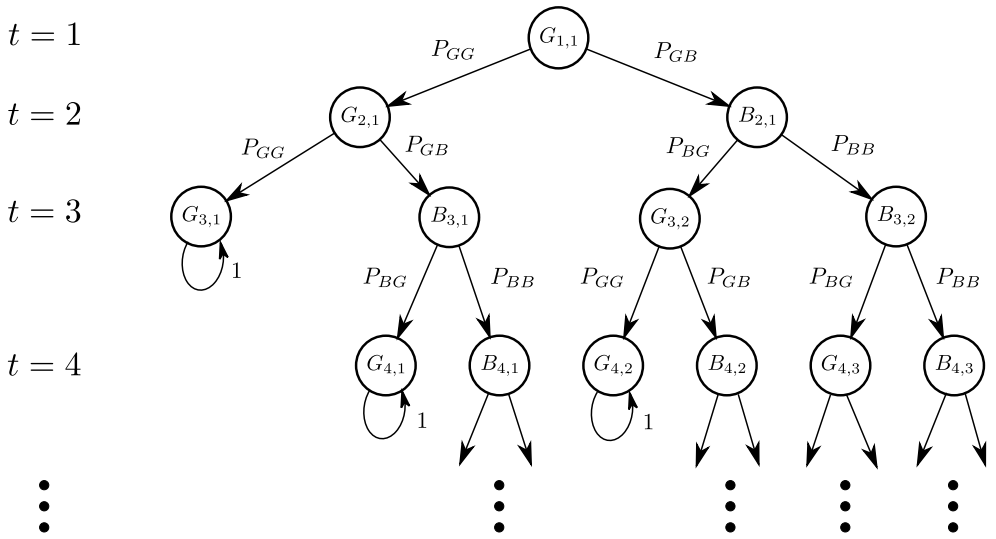


Figure 3.2: Time dilated absorbing Markov chain example corresponding to a Gilbert-Elliot channel model.

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 gives 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

$$N = (I - Q)^{-1}. \tag{3.4.2}$$

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 (3.4.2) 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

$$W = NR, \tag{3.4.3}$$

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

$$P_t(T_{off}, x) = \sum_{j \in \mathcal{S}} W[x, j] \tag{3.4.4}$$

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.

The ability to compute the $P_t(T, x)$ values allows for the computation of the energy costs for both offloading and local execution. 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 (3.4.5).

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 be executed 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

[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 $\left\lceil \frac{S}{B_{min}} \right\rceil$.

$$E_{off}(t,x) = \begin{cases} E_{tr} \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)T_{off}, & 1 \le t < t_D - \frac{S}{B_{min}} + 1 \\ E_{tr}\left(\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{t_D-t} P_t(T_{off},x)T_{off} + \sum\limits_{T_{off}=t_D-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)(t_D-t+1) \right), & t_D - \frac{S}{B_{min}} + 1 \le t \le t_D \\ 0 & t > t_D \end{cases} \tag{3.4.5}$$

$$E_L(t,x) = \begin{cases} \sum\limits_{T_{off}=t_L-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & 1 \le t < t_L \\ \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & t_L \le t \le t_D \\ E_L & t > t_D \end{cases} \tag{3.4.6}$$

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 starts at time t with the channel in state x, is given by (3.4.6). 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 $t - t_L + T_{off} > 0$.

3.5 Optimal Stopping and the OnOpt (Online Optimal) Algorithm

In this section we use the time-dilated absorbing Markov model construction of Section 3.4 and the theory of optimal stopping for Markov decision processes (Peskir and Shiryaev (2006)) to develop the OnOpt algorithm, and show that it achieves the optimal mean energy

for the mobile device. 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 (A general description of *stopping time* is provided in Definition 3.1). 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 *stopping time* $\tau^*_{t_D}$ that satisfies optimization problem (3.5.1) (where the choice $t = t_D + 1$ corresponds to no uploading).

Definition 3.1. *Let $X = \{X_n : n \geq 0\}$ be a stochastic process. A stopping time with respect to X is a random time such that for each $n \geq 0$, the event $\{\tau = n\}$ is completely determined by (at most) the total information known up to time n, $\{X_0, ..., X_n\}$.*

$$v_{t_D}(y) = \min_{t:t_s \leq t \leq t_D+1} E[g_t(X_t)|X_{t_s} = y] \tag{3.5.1}$$

$$= \min_{t:t_s \leq t \leq t_D+1} \sum_{z \in \mathcal{M}} Pr[X_t = z|X_{t_s} = y] g_t(z), \tag{3.5.2}$$

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 $X_t = x$. More specifically,

$$g_t(x) = E_{off}(t, x) + E_L(t, x), \tag{3.5.3}$$

where $E_{off}(t,x), E_L(t,x)$ are the expected offloading and local execution costs, respectively, as defined in (3.4.5) and (3.4.6), when offloading starts at time t with the channel in state $X_t = x$.

The optimization problem (3.5.1) 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 DP, i.e.,

$$V_t(x) = \begin{cases} E_L, & t \geq t_D \\ \min\{g_t(x), E[V_{t+1}|X_t = x]\}, & t = t_s, \cdots, t_D - 1 \end{cases} \tag{3.5.4}$$

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.,

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

It is well known (e.g., Theorem 1.7 in Peskir and Shiryaev (2006)) that (3.5.4) solves the original problem (3.5.1), i.e.,

$$v_{t_D}(y) = V_{t_s}(y), \tag{3.5.5}$$

and, moreover, the following lemma holds:

Lemma 1. (Peskir and Shiryaev (2006)) *The optimal stopping time for* (3.5.1) *is* $\tau^*_{t_D} = \arg\min_{t_s \leq t \leq t_D+1}\{V_t(x) = g_t(x)\}$.

Lemma 1 implies that the on-line algorithm OnOpt, given in Algorithm 1, 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 (3.5.3) and using DP to solve (3.5.4), respectively). If $g_{t_s}(x) \leq E[V_{t_s+1}|X_{t_s} = x]$, 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 1. If offloading finishes before a local execution finishes, then local execution is terminated (Line 10). At Line 12 we check to see if local execution should start so that the job's deadline can be guaranteed. Similarly, Line 15 tests if the local job has completed. In that case, any remote offload in progress will be terminated.

3.6 The Gilbert-Elliot Channel Case

In this section, we consider the well-known Gilbert-Elliot channel model (Gilbert (1960); Elliott (1963)), which has been used in many studies to model stochastic communication channels (Zhang *et al.* (2014); Zhang *et al.* (2013b); Zafer and Modiano (2007); Johnston and Krishnamurthy (2006); Zed *et al.* (1995)) and will be used in the results section of this chapter. 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 1 with the specific energy costs of (3.4.5) and (3.4.6), 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.

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.

$t_o := \infty$ ▷ Offloading initially disabled (t_o is offload start time)
for all $t_s \in \{1, \ldots, t_D\}$ **do**
  if $t_s < t_o$ **then**
    $\overline{c_{t_s}} := g_{t_s}(x)$ ▷ Expected energy cost of offloading at t_s.
    $\overline{c_{t_s+1}} := E[V_{t_s+1}|X_{t_s} = x]$ ▷ Expected energy cost of waiting until $t_s + 1$.
    if $\overline{c_{t_s}} \leq \overline{c_{t_s+1}}$ **then**
      $t_o := t_s$ ▷ Start offloading.
    end if
  else if offloading terminates at t_s **then**
    Abort local execution (if active). ▷ Remote offload response has been received.
    return
  end if
  if $t_s = t_D - T_L + 1$ **then**
    Start local execution. ▷ Ensure that the job deadline is satisfied.
  end if
  if $t_s = t_D$ **then**
    Abort remote offload (if active). ▷ local execution has completed.
    return
  end if
end for

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

$$\frac{(T_{off} - 1)B_g - S}{B_g - B_b} < b \leq \frac{T_{off}B_g - S}{B_g - B_b} \tag{3.6.1}$$

Define $\mathcal{B}$ as a set of integers b satisfying (3.6.1). For any $b \in \mathcal{B}$, the actual transmitted

number of bits, $\hat{S}$, is given by

$$\hat{S} = bB_b + (T_{off} - b)B_g. \tag{3.6.2}$$

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

$$P_t(T_{off}, X_t) = \sum_{b \in \mathcal{B}} \hat{P}_t(T_{off}, b, X_t). \tag{3.6.3}$$

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 2. *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 2 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 (3.6.4).
- $X_t = G$ and $\hat{S} - S < B_b$: See (3.6.5).
- $X_t = B$ and $\hat{S} - S \geq B_b$: See (3.6.7).
- $X_t = B$ and $\hat{S} - S < B_b$: See (3.6.8).

$$\hat{P}_t(T_{off}, b, X_t) = \begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{3.6.4}$$

$$\hat{P}_t(T_{off}, b, X_t) = \begin{cases} \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-2} + \\ + \sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k+1} P_{BG}^{k} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} & b > 0 \\ P_{GG}^{T_{off}-1} & b = 0 \end{cases} \tag{3.6.5}$$

(3.6.6)

$$\hat{P}_t(T_{off}, b, X_t) = \sum_{k=0}^{\min(b-1, T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} \tag{3.6.7}$$

$$\begin{aligned} \hat{P}_t(T_{off}, b, X_t) = & \sum_{k=0}^{\min(b-1, T_{off}-b-1)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k} P_{BG}^{k+1} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k-1} + \\ & + \sum_{k=1}^{\min(b-1, T_{off}-b)} \binom{b-1}{k}\binom{T_{off}-b-1}{k-1} P_{GB}^{k} P_{BG}^{k} P_{BB}^{b-k-1} P_{GG}^{T_{off}-b-k} \end{aligned} \tag{3.6.8}$$

Although equations (3.6.4)-(3.6.8) 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, (3.6.4) applies. We assume $b > 0$ (case $b = 0$ is trivial). Then, (3.6.4) for $b > 0$ implies

$$\hat{P}_t(T_{off}, b, Good) = \sum_{k=0}^{\min\{b-1, T_{off}-b-2\}} Z(k, T_{off}) \tag{3.6.9}$$

where

$$Z(k, T_{off}) = \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1} \; P_{BG}^{k+1} \; P_{BB}^{b-k-1} \; P_{GG}^{T_{off}-b-k-2} \quad (3.6.10)$$

and

$$Z(0, T_{off}) = (T_{off} - b - 1) P_{GB} \; P_{BG} \; P_{BB}^{b-1} \; P_{GG}^{T_{off}-b-2} \quad (3.6.11)$$

Then, it is easy to see that

$$Z(k \; + \; 1, T_{off}) \quad = \quad \frac{(b-k-1)(T_{off}-b-k-2)}{(k+1)(k+2)} \frac{P_{GB}P_{BG}}{P_{BB}P_{GG}} Z(k, T_{off}) \quad (3.6.12)$$

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 (3.6.11) and (3.6.12) 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 (3.6.9); eventually, $O(1)$ $\hat{P}_t$ values are combined to compute each $P_t(T_{off}, G)$ from (3.6.3) (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, (3.4.5) and (3.4.6) 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 (3.5.3) 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 (3.5.4). After this $O(S^2 + ST_D)$ preprocessing, Algorithm 1 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.

3.6.1 Cloud Execution and Downloading

As stated in Section 3.2, 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., $Toff = T_{up} + T_{exec} + T_{down}$ and $S = S_{up} + S_{down}$. This is done as follows. The TDAMC of Figure 3.2, 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 3.2, 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. The definitions of E_{off}, E_L, as well as the calculations carried out in Sections 3.4, 3.5, and 3.6 are then extended accordingly. In what follows throughout this thesis, we define $T_{rest} = T_{exec} + T_{down}$.

3.7 Simulation Results

In this section, computer simulation is used to study the performance of the proposed OnOpt Algorithm. As discussed in Section 3.6, a Gilbert-Elliot channel is assumed when offloading. 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. For comparison, we also plot the offline bound given in Section 3.3, *Local Execution* and two other 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.

In the results, there are three sets of simulations, which span a wide range of parameter values. This was done to assess the relative performance of the offloading algorithms in widely varying situations. 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. 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_b, 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.

3.7.1 Simulation Set 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 3.3a shows the average energy consumption versus T_G, the long term 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.

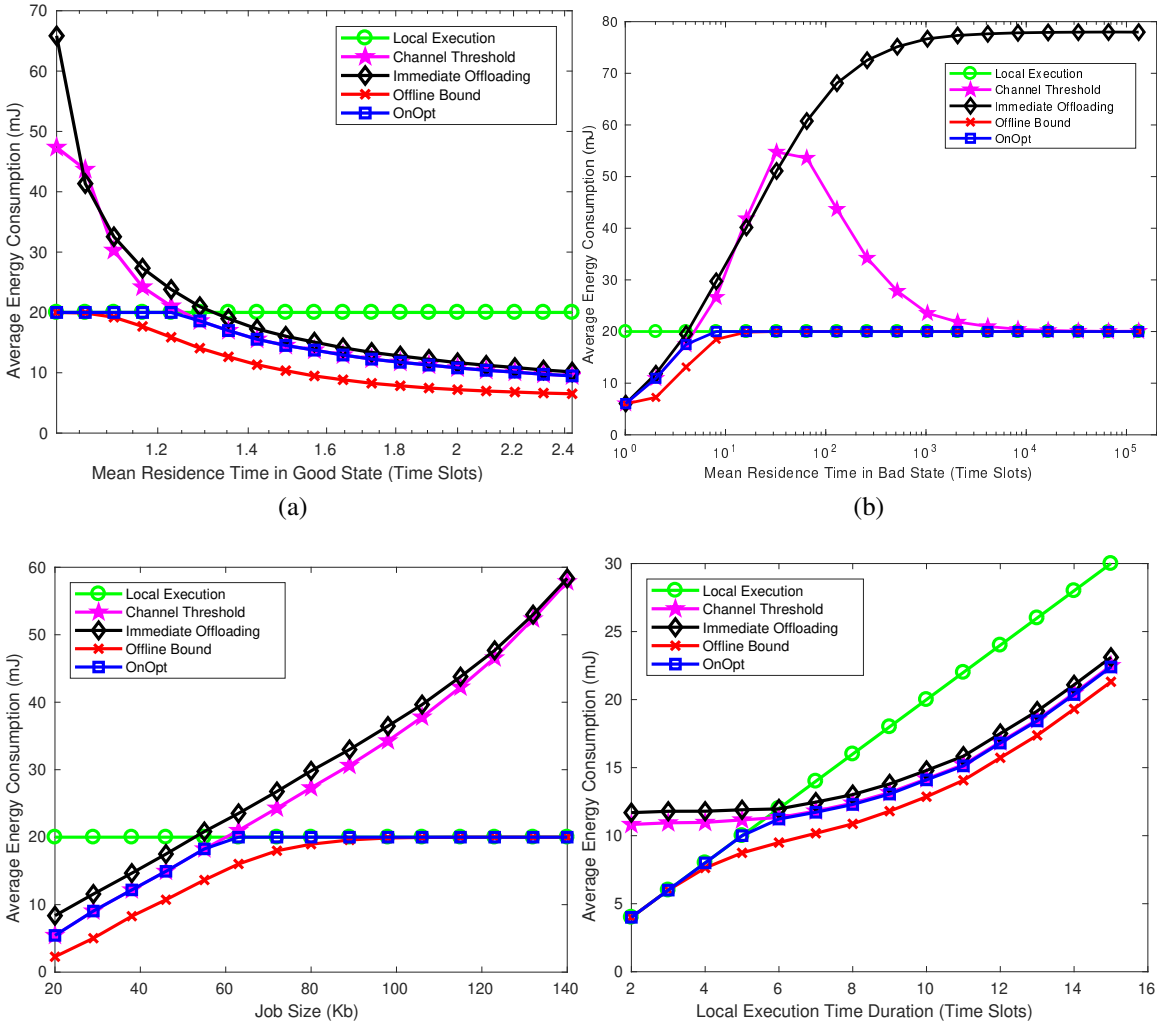
Note that since the energy required for running the online algorithm on the mobile device is negligible compared to that for the offloading transmission, the average mobile energy consumption shown in the graphs does not include this component of energy. This is normally considered to be the case when the amount of transmitted task data is relatively large.

Figure 3.3b shows the average energy consumption versus T_B, the long term mean channel residence time in the bad state, where $T_B = \frac{1}{P_{BG}}$. Figure 3.3b 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 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 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 3.3c 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 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 consumption by offloading unnecessarily.

It is worth mentioning that the energy needed to execute the task locally only depends on the number of CPU cycles to process the job, and not the job size of the task, where the



(a) (b)

(c) $P_{GG} = 0.2$ (d) $P_{GG} = 0.5$

Figure 3.3: Average energy consumption vs job size and local execution time duration.

latter is the number of bits to be processed. Therefore, in this figure, the energy consumed by local execution does not change as job size increases.

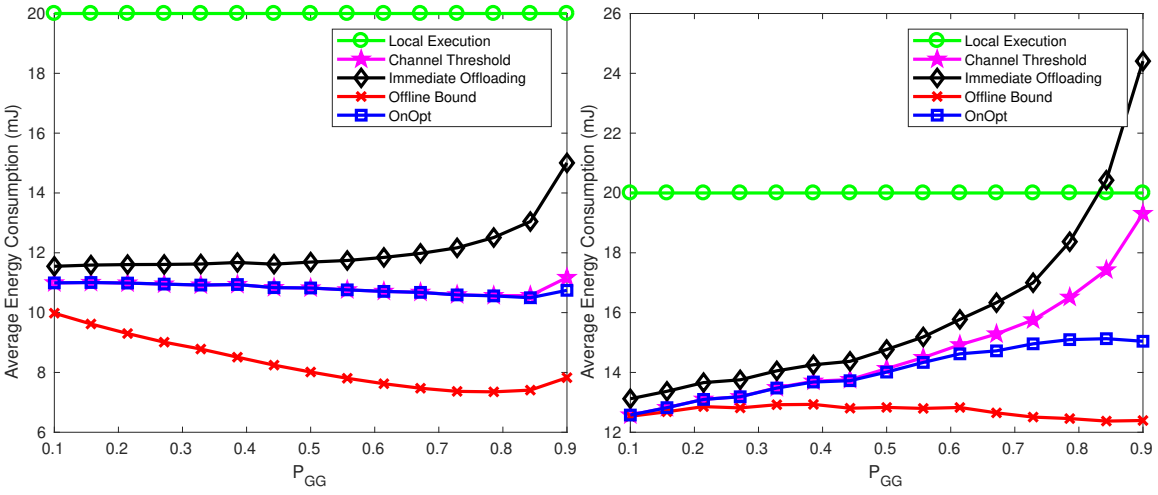
Figure 3.3d shows the average energy consumption versus the local execution duration time. Here, we change D from 2 to 15 CPU Mega-cycles. Deadline T_D is set to 20 time slots in order to make the deadline tighter and observe the effects of increasing T_L. When E_L is small, the OnOpt algorithm does 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.

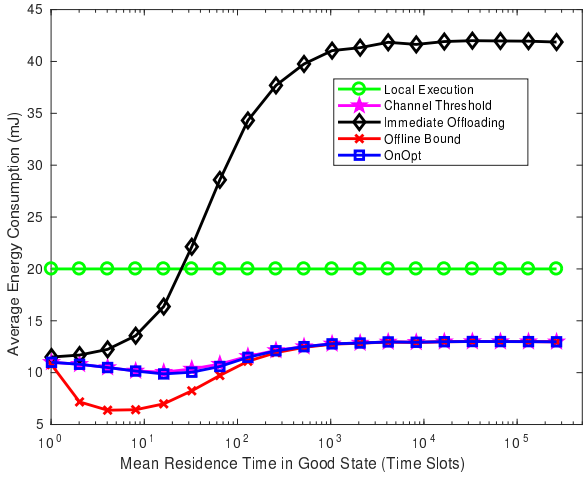
3.7.2 Simulation Set 2

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. 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 3.4a for $t_D = 40$ time slots and Figure 3.4b 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



(a) $t_D = 40$ time slots (b) $t_D = 20$ time slots



(c) $t_D = 40$ time slots

Figure 3.4: Average energy consumption vs P_{GG} and T_G.

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 Threshold algorithm increases. The OnOpt and the Channel Threshold algorithms are very

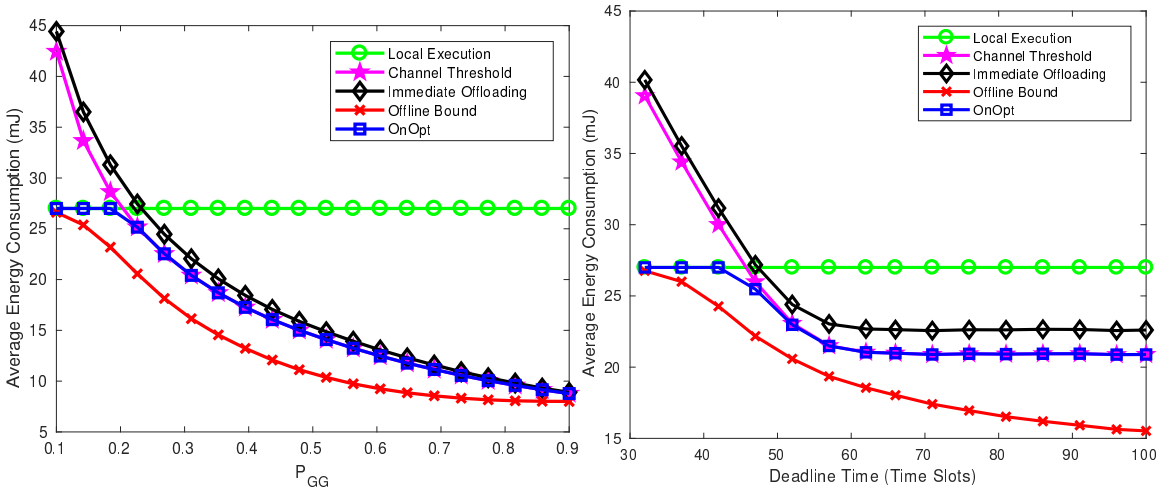
close when $t_D = 80$ time slots since the time constraint is loose enough for the OnOpt algorithm to offload at the first time slot with good channel conditions. When $t_D = 35$ time slots, the difference between the two algorithms starts increasing as P_{GG} becomes large. This is because OnOpt has 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 3.4c shows the average energy consumption versus T_G, which is the long term 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 3.4a. 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 OnOpt and Channel Threshold algorithms are essentially the same, since both decide to offload at the first time slot with the good channel state.

3.7.3 Simulation Set 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_b, 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 simulations 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.



(a) (b) $P_{GG} = 0.3$

Figure 3.5: Average energy consumption versus P_{GG} and t_D

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 3.5a 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 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. As opposed to Immediate Offloading and Channel Threshold algorithms, the proposed OnOpt algorithm avoids wasting energy by deciding not to offload when P_{GG} is low, which results in the same energy consumption as local execution. When P_{GG} is larger, 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 3.5b 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. 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.

In continuous offloading, once the job uploading commences, unless the deadline expires, it will continue without being interrupted, until the entire job gets uploaded. In this scenario, falling into bad channel conditions or channel outages for a long time may increase the mobile energy consumption dramatically. This issue is addressed in the next chapter, where the tasks are considered to be segmented into multiple job upload parts. This offloading scheme gives the system a higher chance for adaption to the varying wireless channel conditions at the end of each upload part.

Chapter 4

Multipart Offloading

4.1 Introduction

The previous chapter addressed the problem of MCO with hard task completion deadline constraints by proposing CLE. In CLE, a hard deadline is guaranteed by permitting simultaneous remote and local task execution when it is needed to ensure task completion times. In this chapter, the OnOPT algorithm is extended by allowing the uploaded data to be split into an arbitrary number of parts when CLE is used. More specifically, the task to be remotely executed is segmented into K parts, with K associated upload initiation time decisions. It is assumed that both K, as well as the bit-sizes of the K parts, are predetermined. We refer to this computation offloading mechanism as *K-Part offloading*. Splitting the uploading data into multiple parts helps further reduce the energy consumption of the mobile device when wireless channel conditions change during the computation offload, and this energy benefit increases with K. Since the task is uploaded in separate parts, separate offload initiation time decisions are needed for each so that mobile device energy consumption is minimized. These decisions have to be made using CLE so that the system

always satisfies the given hard task execution time constraint.

The chapter considers the Markovian wireless channel case. Under this assumption, a new computation offloading algorithm, (*MultiOpt*), is introduced for K-Part offloading. MultiOpt is shown to be energy optimal, in the sense that no other CLE online computation offloading algorithm can achieve a lower mean mobile device energy consumption. This is shown by creating a new Markov process, which incorporates time and other offloading information in the given Markovian channel model, and then by using optimal Markovian stopping theory. The continuous OnOpt Algorithm can be considered as a special version of MultiOpt when $K = 1$.

Since the computational complexity of MultiOpt can be significant, simpler, and more computationally efficient CLE heuristics, which also respect the hard task execution deadline, may be used. This chapter introduces two such heuristics, the *Immediate Offloading*, and *Multi Threshold* algorithms. The energy performance of MultiOpt is compared to these heuristics, as well as to *Local Execution* without offloading and an *Offline Bound* which gives a lower bound on mobile device energy. Simulation results show that MultiOpt performs significantly better when compared to the proposed heuristics, as well as when K increases.

The main contributions of this chapter are summarized as follows:

- An online offloading decision algorithm, i.e., MultiOpt (Multi-decision online Optimal), is introduced. It is proven that the algorithm satisfies hard deadline application constraints, but also achieves the minimum mean mobile device energy possible for homogeneous Markovian wireless channels.

- An integer program (IP) is formulated that computes a lower bound on mobile device energy. This bound is used for comparisons later in Section 4.6.

- An optimal *online* algorithm (MultiOpt) is proposed, based on Dynamic Programming. Its optimal expected energy consumption is proven using Markovian stopping theory.

- Although the proposed MultiOpt algorithm satisfies hard deadlines and is proven to be energy optimal, performance results are also presented that compare it with *Local Execution*, as well as heuristics *OnOpt* (from the previous chapter), *Immediate Offloading*, and *Multi Threshold*, which also employ CLE, to ensure that job execution time constraints are satisfied.

The rest of this chapter is organized as follows. In Section 4.2, system modelling assumptions are presented that include both local and remote job execution where hard job execution deadlines are satisfied using CLE. Following this, in Section 4.3, an offline lower bound on energy consumption is derived, which is plotted and compared to various offloading algorithms in the results section. In Section 4.4, the Markovian channel model is discussed and how it is used to define an absorbing Markov chain that permits us to show the energy optimality of the proposed MultiOpt algorithm, which is introduced in Section 4.5. Finally, in Section 4.6, results are presented that compare MultiOpt with various other computation offloading algorithms that ensure that hard job deadlines are preserved.

4.2 System Model

The system model presented in this chapter matches that of Chapter 3. However, there are some differences related to the K-Part offloading scenario which are discussed in what follows. We assume that each job can be split into K parts for offloading, each with a (known) number of bits to be transmitted through the uplink channel. S_{up_i} is the Bit-size of

the i^{th} piece, where $S_{up} = S_{up_1} + S_{up_2} + \ldots + S_{up_K}$. Splitting the upload in this way can be advantageous when channel conditions change during the offload. For example, it may be better, energy-wise, to delay further uploading when channel conditions worsen, hoping that it will improve in time to complete the computation offload.

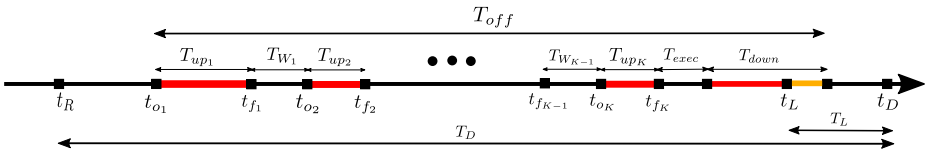


Figure 4.1: Job offloading timing parameters

The K-Part offloading timing sequence is shown in Figure 4.1. The first job piece begins uploading at t_{o_1}, and ends at t_{f_1}, then there are T_{W_1} time slots before the second piece begins uploading at t_{o_2}, and so on, until the Kth piece is uploaded. Note that, by definition, if $t_{o_1} > t_D$, then there is only local execution. It is assumed that the uplink channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, time intervals $T_{up_i} = t_{f_i} - t_{o_i} + 1,\ i = 1, \ldots, K$, are random variables, dependent on the evolution of the uplink channel state as a given upload occurs. After its uploading is complete, the job is executed on the server in T_{exec} time slots, and its execution results are downloaded to the mobile device in T_{down} time slots. We assume that the execution time T_{exec} is assigned when the job is released, 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). We assume that T_{down} is also communicated to the mobile device at this time; more generally, we can treat the dowloaded results as the $K + 1$'th job piece transmitted over the channel, but we will avoid doing this, in order to simplify our presentation. In this chapter, we assume that power control is used on the downlink, so that T_{down} is known before the upload. Therefore, the total offloading time

T_{off} is given by

$$T_{off} = \sum_{i=1}^{K} T_{up_i} + \sum_{i=1}^{K-1} T_{W_i} + T_{exec} + T_{down}, \tag{4.2.1}$$

where T_{W_i} is the number of time slots that elapse after uploading the ith piece and before uploading the $(i+1)$th piece. If the dowloaded results are received before the deadline t_D, any local execution of the job is automatically terminated. Notice that after uploading each part, uploading the next part will be delayed if the algorithm finds itself in a bad channel condition. This mechanism may lead to higher values of T_{off}, which results in triggering local execution with higher probability. However, saving energy from not offloading in bad states compensates for extra energy consumed due to larger concurrent local execution, and results in lower overall energy consumptions compared to continuous offloading.

In what follows, we define

$$T_{rest} = T_{exec} + T_{down}. \tag{4.2.2}$$

4.3 Offline Bound

In this section, an offline lower bound on mobile device energy use for the K-Part offloading is obtained. We use this lower bound in Section 4.6 for performance comparisons with different online computation offloading algorithms proposed throughout this chapter. Similar to Chapter 3, since the bound is offline, we have complete knowledge of future wireless channel states in advance, i.e., we know the bit rate (in bits per time slot) at all times $1 \leq t \leq t_D$ (note that t_R is considered to be 1). When a job is released, the bound chooses the job offload times so that its deadline is satisfied and the energy needed to offload the job is minimized. Let $t_{f_i}(t_{o_i}),\ 1 \leq i \leq K$, be the upload finishing time as a function of

the uploading starting time t_{o_i} for the ith part, and define $t_{f_0}(t_{o_0}) = 0$. E_{tr} and E_{rc} are the energy costs per time slot for channel transmitting and receiving, respectively. The optimal values for t_{o_i} are found by solving the integer programming (IP) program (4.3.1)-(4.3.3).

$$\min_{t_{o_1},\ldots,t_{o_K}} \quad \frac{\max(t_{f_K}(t_{o_K}) + T_{rest}, t_L) - t_L}{T_L} E_L + E_{tr} \sum_{i=1}^{K} (t_{f_i}(t_{o_i}) - t_{o_i} + 1) + E_{rc} T_{down} \tag{4.3.1}$$

$$s.t. \ \frac{\max(t_{f_K}(t_{o_K}) + T_{rest}, t_L) - t_L}{T_L} E_L + E_{tr} \sum_{i=1}^{K} (t_{f_i}(t_{o_i}) - t_{o_i} + 1) + E_{rc} T_{down} \leq E_L \tag{4.3.2}$$

$$t_{f_{i-1}}(t_{o_{i-1}}) + 1 \leq t_{o_i} \leq t_D, \quad i = 1, \ldots, K. \tag{4.3.3}$$

The first term in (4.3.1) is the local execution energy cost incurred, the second term accounts for the energy cost of uploading the K job parts, and the last term is the cost for downloading the results from the cloud. Constraint (4.3.2) 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 uploading start times t_{o_i}, i.e., it is best to execute the job solely locally without offloading.

4.4 Markovian Channel and the Time-Dilated Absorbing Markov Model

Similar to what was explained in Section 3.4, we assume 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. Each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded in that state. The CSMC

transition matrix is defined as $\mathbb{P} = [P_{i,j}]$, where $P_{i,j}$ is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. As defined, CSMC is memoryless, but to estimate the expected energy and time-period needed to perform the K-Part offloading, we will need to incorporate time and other offloading information into the Markov chain structure. Therefore, we will consider a TDAMC (which was proposed in Chapter 3) 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. 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 as $TDAMC_1$ below), endowed with energy costs and absorbing states.

The part of the TDAMC that 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 4.2. To simplify the exposition, the diagram shows the two-state Gilbert-Elliot channel case, but the procedure is valid for any Markovian channel. In the Gilbert-Elliot case, the channel 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 4.2 models the offloading process initiated at time slot t_s, when the channel state that has been reached at that time is $s_{t_s}^{19}$. 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

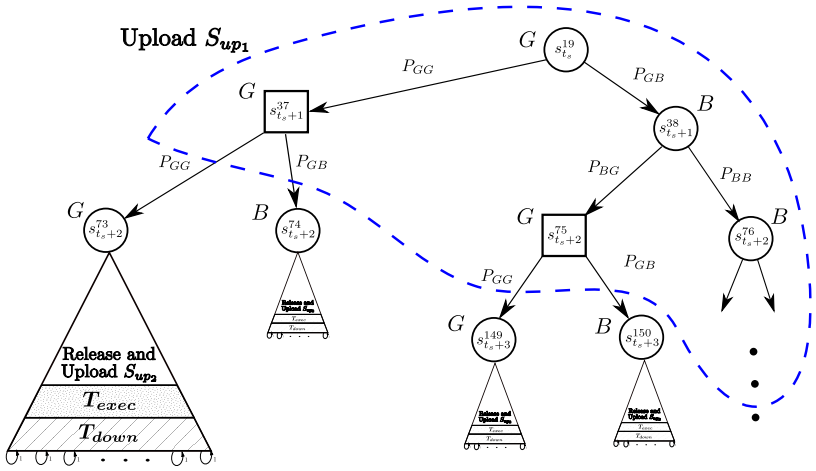


Figure 4.2: $TDAMC_1$ when offloading S_{up_1} starts at time t_s.

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 4.2). 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 rest of the K pieces, and 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 for $K=2$ is depicted in Figure 4.2 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 $S_{up_2}, \ldots, S_{up_K}$, is solved in Section 4.5. Then the energy cost of each subtree is the optimal expected cost (over all paths) of completing offloading, when uploading S_{up_1} 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 can treat states $X_{t_{f_1}+1}$ as absorbing states, each with cost equal to the energy cost of its own subtree. This process can obviously be repeated, in order to build the corresponding $TDAMC_i$ for any piece i.

The probability of uploading S_{up_i} bits in T_{up_i} time slots, starting at time slot t_{o_i}, and a state $X_{t_{o_i}}$, for $i = 1, \ldots, K$, can be calculated by building a separate $TDAMC_i$, with a set of absorbing states $\mathcal{A}_i$, and a set of transient states $\mathcal{T}_i$, for $i = 1, \ldots, K$. It encodes the evolution of the channel starting at time slot t_{o_i} and state $X_{t_{o_i}}$, and until S_{up_i} bits are uploaded, at which point an absorbing state in $\mathcal{A}_i$ is reached. Its transition matrix can be written Grinstead and Snell (2006) as

$$\mathbb{P}_i = \begin{bmatrix} Q_i & R_i \\ \mathbf{0} & I_{\mathcal{A}i} \end{bmatrix}, \tag{4.4.1}$$

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

The theory of absorbing Markov chains implies that various statistics can be computed by forming the fundamental matrix $N_i = (I - Q_i)^{-1}$, where $N_i[l, m]$ gives the expected number of times that $TDAMC_i$ is in transient state m if the system is started in transient state l. Given the structure of $TDAMC_i$, N_i can be easily decomposed and calculated as in Chapter 3, since the particular structure of matrices Q_i, N_i, N_i^{-1} is the same simple one as in Chapter 3. The absorption probabilities matrix W_i for all absorbing states is given by

$$W_i = N_i R_i, \tag{4.4.2}$$

where W_i is a $|\mathcal{T}_i| \times |\mathcal{A}_i|$ matrix, and $W_i[l, m]$ gives the probability that absorbing state m will be reached when starting in transient state l. Therefore, the probability of uploading the ith part with size S_{up_i} in T_{up_i} time slots, starting at time slot t_{o_i} and state $X_{t_{o_i}}$, is

$$P_{t_{o_i}}(S_{up_i}, T_{up_i}, X_{t_{o_i}}) = \sum_{j \in \mathcal{S}_i} W_i[X_{t_{o_i}}, j], \tag{4.4.3}$$

where $\mathcal{S}_i$ is the set of absorbing states in $TDAMC_i$ reached by a path of length $T_{up_i} + 1$ from the root $X_{t_{o_i}}$.

$$\hat{E}_{off_i}(S_{up_i}, T_{up_i}, t_{o_i}) = E_{tr}[\min\{t_D, t_{o_i} + T_{up_i} - 1\} - \min\{t_D, t_{o_i} - 1\}] \tag{4.4.4}$$

$$\begin{aligned}\hat{E}_{off_K}(S_{up_K}, T_{up_K}, t_{o_K}) = {} & E_{tr}[\min\{t_D, t_{o_K} + T_{up_K} - 1\} - \min\{t_D, t_{o_K} - 1\}] \\ & + E_{rc}[\min\{t_D, t_{o_K} + T_{rest} - 1\} - \min\{t_D, t_{o_K} + T_{exec} - 1\}]\end{aligned} \tag{4.4.5}$$

$$E_{off_i}(S_{up_i}, X_{t_{o_i}}) = \sum_{T_{up_i} = \frac{S_{up_i}}{B_{max}}}^{\frac{S_{up_i}}{B_{min}}} P_{t_{o_i}}(S_{up_i}, T_{up_i}, X_{t_{o_i}}) \hat{E}_{off_i}(S_{up_i}, T_{up_i}, t_{o_i}) \tag{4.4.6}$$

$$\hat{E}_{L_i}(T_{up_i}, t_{f_{i-1}}, t_{o_i}) = \begin{cases} 0, & t_{o_i} \leq t_L - T_{up_i} \text{ or } t_{f_{i-1}} \geq t_D \\ \frac{\min\{t_D, t_{o_i} + T_{up_i} - 1\} - \max\{t_L, t_{f_{i-1}} + 1\} + 1}{T_L} E_L, & \text{otherwise} \end{cases} \tag{4.4.7}$$

$$\hat{E}_{L_K}(T_{up_K}, t_{f_{K-1}}, t_{o_K}) = \begin{cases} 0, & t_{o_i} \leq t_L - (T_{up_i} + T_{rest}) \text{ or } t_{f_{i-1}} \geq t_D \\ \frac{\min\{t_D, t_{o_K} + T_{up_K} + T_{rest} - 1\} - \max\{t_L, t_{f_{K-1}} + 1\} + 1}{T_L} E_L, & \text{otherwise} \end{cases} \tag{4.4.8}$$

$$E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) = \sum_{T_{up_i} = \frac{S_{up_i}}{B_{max}}}^{\frac{S_{up_i}}{B_{min}}} P_{t_{o_i}}(S_{up_i}, T_{up_i}, X_{t_{o_i}}) \hat{E}_{L_i}(T_{up_i}, t_{f_{i-1}}, t_{o_i}) \tag{4.4.9}$$

For $i = 1, 2, \ldots, K-1$, if the uploading of S_{up_i} starts at time slot t_{o_i}, the expected offloading energy cost when offloading starts at time slot t_{o_i} in state $X_{t_{o_i}}$ and finishes exactly in T_{up_i} time slots or at t_D (whichever comes first), is given by equation (4.4.4),

where E_{tr} is the transmission energy of the mobile device during one time slot, while the expected energy cost of uploading S_{up_K} is given by equation (4.4.5), where E_{rc} is the energy consumption of the mobile device during one time slot when receiving from the server. Then, the expected offloading energy cost E_{off_i} for $i = 1, 2, \ldots, K$ is computed by equation (4.4.6), where B_{max} and B_{min}, respectively, are the bit rates at the best and worst channel states.

Similarly, the local execution energy cost corresponding to the uploading of the ith job piece, for $i = 1, 2, \ldots, K-1$ is given by (4.4.7)[1], while the local execution energy cost due to the Kth job piece and the rest of offloading time T_{rest} is given by (4.4.8). Then, the expected local execution energy cost E_{L_i} for $i = 1, 2, \ldots, K$ is computed by equation (4.4.9).

4.5 Optimal Algorithm for K-Part Offloading

In this section we use the TDAMC's constructed in Section 4.4, and the theory of optimal stopping for Markov decision processes (Peskir and Shiryaev (2006)), in order to define optimal offloading algorithms, and prove their optimality. Recall that, for simplicity, we have set $t_R = 1$. A high-level description of algorithm MultiOpt (cf. Algorithm 2) 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 for time slot $t+1$. Once the uploading of S_{up_1} finishes, for the rest of the parts the algorithm repeats the

[1] We set $t_{f_0} = t_R - 1$.

same decision process at every time slot (using $TDAMC_i$ to compute expected costs), to determine the time $t^*_{o_i}$ to start uploading S_{up_i} for $i = 2, 3, \ldots, K$.

At any time slot t (and state X_t), and given that pieces $1, 2, \ldots, i-1$ have already been uploaded, MultiOpt decides the uploading starting time $t^*_{o_i} \geq t$ for S_{up_i}. Its decisions $t^*_{o_i}$ for $i = 1, 2, \ldots, K$ are optimal iff they are the solutions of the minimization problems (one for each i) (4.5.1) (4.5.2), where $\mathcal{S}_i$ is the set of states reachable after running the

$$v_i(t_{f_{i-1}}, X_t) = \begin{cases} 0, & t > t_{f_{i-1}} \geq t_D \\ \min\limits_{t \leq t_{o_i} \leq t_D+1} \sum\limits_{X_{t_{o_i}} \in \mathcal{S}_i} Pr[X_{t_{o_i}}|X_t]\Big(E_{off_i}(S_{up_i}, X_{t_{o_i}}) + \\ E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) + \sum\limits_{X_{t_{f_i+1}} \in \hat{\mathcal{S}}_i} W_i[X_{t_{o_i}}, X_{t_{f_i+1}}] v_{i+1}(t_{f_i}, X_{t_{f_i+1}})\Big), & \\ & t_{f_{i-1}} < t \leq t_D \end{cases} \tag{4.5.1}$$

$$v_K(t_{f_{K-1}}, X_t) = \begin{cases} 0, & t > t_{f_{K-1}} \geq t_D \\ \min\limits_{t \leq t_{o_K} \leq t_D+1} \sum\limits_{X_{t_{o_K}} \in \mathcal{S}_K} Pr[X_{t_{o_K}}|X_t]\Big(E_{off_K}(S_{up_K}, X_{t_{o_K}}) \\ +E_{L_K}(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}})\Big), & t_{f_{K-1}} < t \leq t_D \end{cases} \tag{4.5.2}$$

$$g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) = E_{off_i}(S_{up_i}, X_{t_{o_i}}) + E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) + \sum_{X_{t_{f_i+1}} \in \mathcal{S}_i} W_i[X_{t_{o_i}}, X_{t_{f_i+1}}] V_{i+1}(t_{f_i}, X_{t_{f_i+1}}), \quad i = 1, \ldots, K \tag{4.5.3}$$

$$g_K(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}}) = E_{off_K}(S_{up_K}, X_{t_{o_K}}) + E_{L_K}(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}}) \tag{4.5.4}$$

channel until time slot t_{o_i}, $\hat{\mathcal{S}}_i$ is the set of absorbing states of $TDAMC_i$ rooted at $X_{t_{o_i}}$,

and $v_{i+1}(t_{f_i}, X_{t_{f_i}+1})$ is the optimal expected energy cost for the rest of the offloading when S_{up_i} finishes uploading at time t_{f_i}, i.e., the cost of the absorbing state $X_{t_{f_i}+1}$ of $TDAMC_i$. In (4.5.1)-(4.5.2), $g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}})$ is the expected energy cost of uploading S_{up_i}, if uploading of $S_{up_{i-1}}$ finishes at $t_{f_{i-1}}$ and uploading S_{up_i} starts at time slot t_{o_i} and state $X_{t_{o_i}}$, and is given by (4.5.3)-(4.5.4). Note that we allow the algorithm to decide not to offload or stop offloading if this is to its benefit, by allowing uploading decisions to take the value $t_D + 1$.[2] For every time slot t_{o_i} and state $X_{t_{o_i}}$, we define the expected cost $V_i(t_{f_{i-1}}, X_{t_{o_i}})$ recursively in (4.5.5), for $i = 1, \ldots, K$. $V_i(t_{f_{i-1}}, X_{t_{o_i}})$, which can be computed using

$$V_i(t_{f_{i-1}}, X_{t_{o_i}}) = \begin{cases} 0, & t_{o_i} > t_{f_{i-1}} \geq t_D \\ \frac{t_D - \max\{t_{f_{i-1}}+1, t_L\}+1}{T_L} E_L, & t_{o_i} \geq t_D > t_{f_{i-1}} \\ \min\{g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}), E[V_i(t_{f_{i-1}}, X_{t_{o_i}+1})|X_{t_{o_i}}]\}, & t_D > t_{o_i}. \end{cases} \tag{4.5.5}$$

Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_i} at time slot t_{o_i} and state $X_{t_{o_i}}$, and the expected cost of postponing that decision to time slot $t_{o_i} + 1$,

$$E[V_i(t_{f_{i-1}}, X_{t_{o_i}+1})|X_{t_{o_i}}] = \sum_{X_{t_{o_i}+1} \in \mathcal{T}_i} Pr[X_{t_{o_i}+1}|X_{t_{o_i}}] V_i(t_{f_{i-1}}, X_{t_{o_i}+1}),$$

where $\mathcal{T}_i$ is the set of states reachable after running the channel for $t_{o_i} + 1$ time slots. Note that (4.5.5) implies a *policy*, that dictates whether at any time t_{o_i} and state $X_{t_{o_i}}$ the algorithm should start uploading S_{up_i} (if the $\min$ is attained by g_i), or should otherwise wait.

We prove optimality by (reverse) induction. It is well known (e.g., Theorem 1.7 in

[2]Equations (4.4.4), (4.4.5), (4.4.7), (4.4.8) have been set up to reflect this.

(Peskir and Shiryaev (2006))) that policy V_K is *optimal*, i.e., solves the original problem (4.5.2), since

$$v_K(t_{f_{K-1}}, X_{t_K}) = V_K(t_{f_{K-1}}, X_{t_K}),\ \forall t_K > t_{f_{K-1}}, X_{t_K}. \tag{4.5.6}$$

Hence the following holds:

Lemma 3. (Peskir and Shiryaev (2006)) *The optimal time for starting uploading S_{up_K} is*

$$t^*_{o_K} = \underset{t_{f_{K-1}} < t_{o_K} \leq t_D}{\arg\min} \{V_K(t_{f_{K-1}}, X_{t_{o_K}}) = g_K(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}})\} \tag{4.5.7}$$

Assuming that decisions $t^*_{o_K}, t^*_{o_{K-1}}, \ldots, t^*_{o_{i+1}}$ are optimal, i.e.,

$$v_k(t_{f_{k-1}}, X_{t_k}) = V_k(t_{f_{k-1}}, X_{t_k}),\ \forall t_k > t_{f_{k-1}}, X_{t_k} \tag{4.5.8}$$

holds for $k = K, K-1, \ldots, i+1$, we prove that the ith decision $t^*_{o_i}$ of MultiOpt is also optimal. Note that (4.5.1) becomes (4.5.9).

$$v_i(t_{f_{i-1}}, X_t) = \begin{cases} 0, \quad t > t_{f_{i-1}} \geq t_D \\ \min\limits_{t \leq t_{o_i} \leq t_D+1} \sum\limits_{X_{t_{o_i}} \in \mathcal{S}_i} Pr[X_{t_{o_i}} | X_{t_i}] \Big(E_{off_i}(S_{up_i}, X_{t_{o_i}}) + E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) \\ + \sum\limits_{X_{t_{f_i}+1} \in \hat{\mathcal{S}}_i} W_i[X_{t_{o_i}}, X_{t_{f_i}+1}] V_{i+1}(t_{f_i}, X_{t_{f_i}+1}) \Big), \quad t_{f_{i-1}} < t \leq t_D \end{cases} \tag{4.5.9}$$

But then, Theorem 1.7 in (Peskir and Shiryaev (2006)) can be applied again, to show

Lemma 4. (Peskir and Shiryaev (2006)) *The optimal time for starting uploading S_{up_i} is*

$$t^*_{o_i} = \underset{1\le t_{o_i}\le t_D}{\arg\min}\{V_i(t_{f_{i-1}}, X_{t_{o_i}}) = g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}})\} \tag{4.5.10}$$

Lemmata 3 and 4 imply that the on-line algorithm MultiOpt (Algorithm 2) is optimal for general Markovian channels.

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}, ..., S_{up_K}.

$i = 1$
for all $t = 1, \ldots, t_D$ **do**
  if job finished uploading **then**
    Break
  end if
  if currently uploading **then**
    Continue
  end if
  if min in (4.5.5) is g_i **then** ▷ part $i-1$ has been uploaded but i has not started uploading
    start uploading part i
    $i = i + 1$
  end if
end for

4.6 Simulation Results

In this section, computer simulation is used to study the performance of the proposed K-Part offloading algorithm for $K = 2$, 3 and 4. For comparison, we also plot the *Offline Bound* given in Section 4.3, and performance of *Local Execution* and three other algorithms, referred to as *OnOpt Algorithm*, *Immediate Offloading*, and *Multi Threshold*. These algorithms all employ CLE to ensure that job execution time constraints are satisfied. The

Local Execution algorithm executes the entire job locally without doing any offloading. The OnOpt Algorithm is the online algorithm proposed in the previous chapter which finds the optimum offloading decision to minimize the expected energy consumption of the mobile device. It is the special version of MultiOpt when $K = 1$. For the Immediate Offloading algorithm, a job is offloaded immediately at the release time if $S/B_{max} + T_{rest} \leq T_D$; otherwise, the job is executed locally without offloading since offloading cannot be completed before the job deadline even with contiguous best wireless channel states. For the Multi Threshold algorithm, uploading for the first piece starts at the first time slot when the channel condition is above a given threshold, if the remaining time before the job completion deadline is at least $S_{up}/B_{max} + T_{rest}$; otherwise no offloading is performed for the entire job. For the Gilbert-Elliot channel, any threshold between the good and bad states can be used, i.e., uploading starts at the first time slot with the good channel state. After the $(i-1)$th piece is uploaded, uploading for the ith piece starts as soon as the channel state becomes above the threshold, if the remaining time before the job completion deadline is no less than $\sum_{k=i}^{K} S_{up_k}/B_{\max} + T_{rest}$; otherwise, uploading is stopped. In both the Immediate Offloading and the Multi Threshold algorithms, local execution starts at time slot t_L if offloading (includes uploading to, remote execution at, and downloading from the server) is not completed at time slot $t_L - 1$, i.e., they ensure that the job deadline is satisfied.

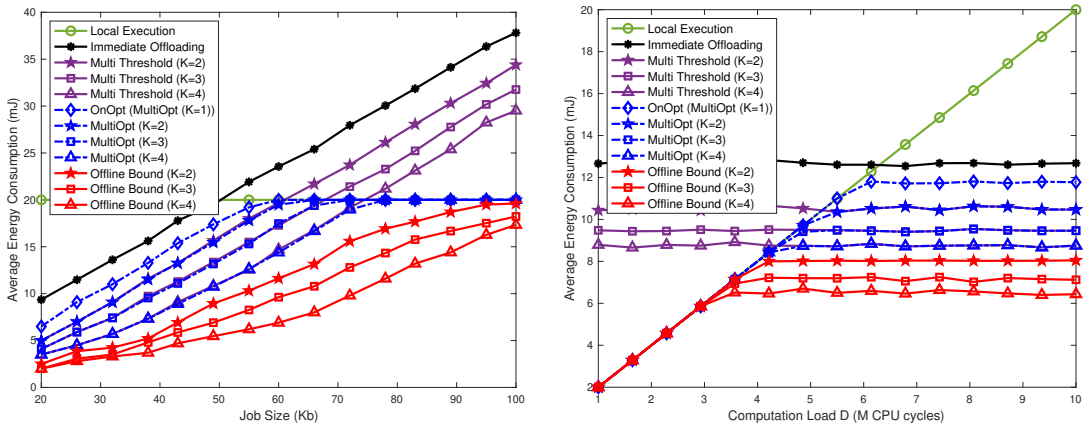
In this simulation, the job size S_{up}, i.e., total amount of data to be uploaded, is split into K equal parts i.e., $S_{up_1} = S_{up_2} = \cdots = S_{up_K} = S_{up}/K$. The default parameters used in the simulations are given as follows. Each time slot is 1 ms. The transmit and receive power is 1 W and 0.5 W, respectively, which means that the transmission and receive energy during each time slot is $E_{tr} = 1$mJ and $E_{rc} = 0.5$mJ, respectively. In our offloading results we have used the well-known Gilbert-Elliot channel model, that is often used to model

random wireless channels (Zhang *et al.* (2014); Zhang *et al.* (2013b); Gilbert (1960); Elliott (1963); Johnston and Krishnamurthy (2006); Zafer and Modiano (2007)). This model is often used to characterize burst noise effects in wireless links, where the channel can abruptly transition between good and bad conditions. This is a good test for computation offloading algorithms with hard execution time constraints, since the channel may be less predictable than those where channel conditions are more correlated and predictable. We assume that $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. In this case, $P_{GB} = P_{BB}$, $P_{BG} = P_{GG}$, the equilibrium channel state probabilities are given by $P_g = P_{GG}$ and $P_b = P_{BB}$, and P_{GG} can be used as a measure of the average channel quality. 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. Note that in the two-state channel model, $B_{max} = B_g$ and $B_{min} = B_b$. In order to produce the results below, each value of average energy consumption is obtained by the averaging of 1500 random i.i.d. runs of the wireless channel, which follows the Gilbert-Elliot model described above.

4.6.1 Simulation Set 1

In this section we consider a job with $D = 10$M CPU cycles and $T_D = 60$ time slots. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$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. We consider that the remote execution time is $T_{exec} = 1$ time slot, i.e., the remote processing speed is 10 times as fast as local processing. The download time T_{down} is assumed to be 1 time slot.

Figure 4.3a shows the average energy consumption of the mobile device as the data



(a) $P_{GG} = 0.2$ (b) $P_{GG} = 0.5$

Figure 4.3: Average energy consumption versus data size S_{up} and D

size S_{up} increases. The energy used by Local Execution is constant for all S_{up}. When S_{up} is smaller, the delay constraint is less stringent, and it is more likely for offloading (without local execution) to meet the delay constraint due to a shorter channel uploading time. In this case, the Multi Threshold and MultiOpt algorithms have approximately the same average energy consumption, since it is more likely for the MultiOpt algorithm to decide to start uploading as soon as the channel state is good, making it start the upload at the same time slot with the Multi Threshold algorithm. Compared to OnOpt, the energy consumption of MultiOpt with $K > 1$ is lower, indicating that splitting the job uploading into multiple pieces brings more flexibility that helps the mobile device to avoid uploading during bad channel states; on the other hand, since OnOpt uploads the job continuously, its uploading is more likely to encounter the bad channel states, and therefore, takes a longer time and consumes more energy. The Immediate Offloading algorithm consumes higher energy as compared to the other algorithms, because there is a certain probability to encounter the bad channel state at the job release time and the following time slots, and the probability becomes higher when P_{GG} is lower. As S_{up} increases, a longer time is needed for wireless transmissions, and the offline bound and the MultiOpt algorithms may

decide not to offload, resulting in the same energy consumption as Local Execution, while the Immediate Offloading and Multi Threshold algorithms waste energy consumption by offloading unnecessarily and result in much higher energy consumption.

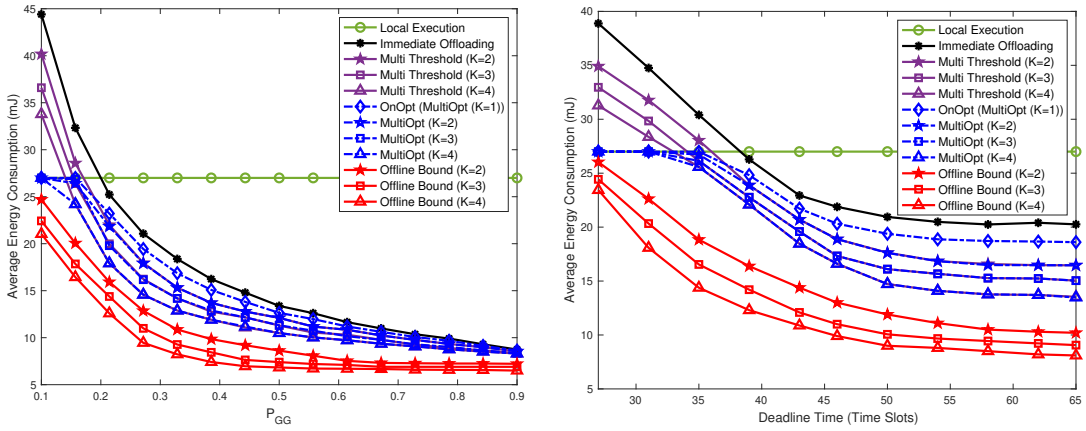
Comparing the MultiOpt algorithm with $K = 1$ (i.e., the OnOpt algorithm), 2, 3, and 4, we can see that splitting the job into multiple pieces helps reduce the energy consumption of the mobile device, since doing this can avoid uploading during some bad channel states, provided the job completion deadline can be satisfied.

Figure 4.3b shows the average energy consumption of the mobile device versus the amount of computation load D. Deadline T_D is set to 40 time slots. When D is small, the MultiOpt algorithm (including OnOpt) does not offload because the local execution energy is low and less than the energy needed to upload the data. As D increases, the energy required for local execution increases, and it becomes more likely that offloading consumes less mobile device energy than local execution. The energy consumption for MultiOpt becomes constant when D is sufficiently large. This is because the delay constraint is relatively loose in the simulated system, which allows offloading to be completed before t_L. Therefore, when D is relatively large, the energy consumption is the same as the energy consumption for wireless transmissions, which does not depend on D. The figure also shows that MultiOpt can save mobile device energy by splitting the job into multiple pieces and uploading separately.

4.6.2 Simulation Set 2

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 $S_{up} = 60$Kb, $D = 18$M CPU cycles, and $T_D = 60$ 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, and the download time T_{down} is 1 time slot.



(a) $P_{GG} = 0.3$ (b) $P_{GG} = 0.3$

Figure 4.4: Average energy consumption versus P_{GG} and T_D.

Figure 4.4a shows the average energy consumption of different algorithms as P_{GG} varies. The offline bound is close to the energy consumption of Local Execution only when P_{GG} is close to 0, in which case the channel is almost always in the bad state and local execution is almost always the optimum choice. The Immediate Offloading and Multi Threshold algorithms result in much higher energy consumption when P_{GG} is small. Since the channel is in the bad state in most time slots, uploading data requires a long time. Therefore, there is a high probability that offloading cannot meet the delay constraint and/or consumes high energy; furthermore, due to the long uploading time, there may be a long overlap time between offloading and local execution. As a result, energy is unnecessarily wasted in the Immediate Offloading and Multi Threshold algorithms 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 for

all the offloading algorithms. The Multi Threshold algorithm consumes slightly lower energy than Immediate Offloading. By delaying the uploading (of each piece of the job) until the channel state becomes good, it reduces the transmission time and saves energy consumption. The difference is more obvious when P_{GG} is smaller, since the probability of encountering bad channel states is higher.

When P_{GG} is low, the MultiOpt (including OnOpt) algorithm chooses to not offload, and therefore, results in the same energy consumption as Local Execution; and when P_{GG} is larger, the algorithm more likely chooses to offload, since channel conditions become better and a shorter time and less energy is needed to offload. Figure 4.4a also shows that, the energy consumption of MultiOpt is lower when K is larger, if the mobile device decides to offload, since splitting the job into more pieces brings more flexibility that helps the mobile device avoid transmissions in bad channel conditions.

By comparing the MultiOpt and Multi Threshold algorithms, we can see that for given K, when P_{GG} is relatively large, the two algorithms consume almost the same energy. This is because the channel condition in general is good, so that the time required for uploading the data is relatively short, and the time required to complete offloading is much less than T_D. For the MultiOpt algorithm, if the decision is to offload the next piece of the job, it is most likely the first time slot with a good channel state, which is the same as the Multi Threshold algorithm. The gap between the MultiOpt algorithm and the offline bound is due to the fact that the MultiOpt algorithm can only use statistical channel information, while the offline bound has knowledge of the channel states of all future time slots.

For all the offloading solutions, the mobile device energy consumption decreases as P_{GG} increases, since the probability of having the good channel state increases, which reduces the time needed to upload the data and makes it more likely for offloading to meet

the delay constraint and consume less energy. When P_{GG} is relatively small, the energy decreases fast as P_{GG} increases; and when P_{GG} is sufficiently large, further increasing P_{GG} does not help reduce the energy consumption significantly, since each piece of the job has to be uploaded in consecutive time slots.

Figure 4.4b shows the average energy consumption of the algorithms as the job deadline T_D changes. When T_D is small, the MultiOpt (including OnOpt) algorithm decides to not offload most of the time, resulting in the same energy consumption as Local Execution; the offline bound result in lower energy consumption than Local Execution, since it foresees the future channel states and can decide to offload at a future state; while Immediate Offloading and Multi Threshold most likely result in concurrent offloading and local execution, since offloading cannot meet the delay constraint, and therefore, result in higher energy consumption than Local Execution. The Multi Threshold algorithm achieves a lower energy consumption than the Immediate Offloading algorithm by delaying the offloading until the first time slot with the good channel state.

As T_D increases, more time is available to offload before triggering local execution, resulting in even lower energy consumption for all the offloading algorithms. When T_D is sufficiently large, all the offloading algorithms can achieve lower average energy consumption than using Local Execution. For given K, the MultiOpt and the Multi Threshold algorithms result in the same average energy consumption.

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 there are six consecutive time slots with good channel states before t_L. In general, a larger T_D increases the possibility of having a shorter time interval to complete the uploading, thus reducing the energy consumption. When T_D

is small, increasing T_D helps reduce the energy consumption significantly; and when T_D is relatively large, further increasing T_D does not help reduce the energy consumption, since it is almost always possible to complete uploading with six time slots before t_L.

The main problem left open to be discussed in the next chapter is the case of preemptive offloading, i.e., when the protocol allows for offloading a piece of the task at any time-slot, and not necessarily in K contiguous parts. A new Dynamic Programming algorithm will have to be developed along the lines of Section 4.5, one that will have to take into account the task bits already offloaded. Even if such an optimal algorithm is developed, it may be of only theoretical interest. One of the main conclusions of our work is that as the number of job pieces K increases, it becomes harder to computationally perform the calculations needed by MultiOpt. Preemption will increase K to be proportional to the bit-size of the task, and, therefore, a DP approach becomes impractical. In this case, the development of fast heuristics with good (i.e., close to the optimal) performance is also another objective.

Chapter 5

Preemptive Offloading

5.1 Introduction

This chapter uses CLE and considers Preemptive Mobile Computation Offloading, referred to as *Preemptive Offloading* for short. In the preemptive case, the algorithm makes separate upload initiation time decisions at the start of *every* time slot. Based on its inputs, it decides whether to use the current time slot to continue the upload or defer uploading to future time slots. Preemptive offloading can be used to reduce mobile energy use when wireless channel conditions change during the computation offload. These decisions are made such that the system always satisfies the given hard task execution time constraint using concurrent local execution offloading. The objective is to minimize mobile device energy consumption, subject to this constraint. It is very easy to illustrate the value of preemption in computation offloading. For example, consider a mobile device that has a large job offload over a 2-state Markovian channel (i.e., a Gilbert-Elliot channel), and toggles between good and bad channel states with the same mean time in each state, and at a rate much smaller than the upload time. Then the offload energy needed for the non-preemptive

case would be roughly twice that of using preemption. The statistics of the channel can easily be such that the advantage to the preemptive case is far greater.

We consider the homogeneous Markovian channel case, which is commonly used to model random wireless channel conditions. Under this assumption, computation offloading algorithms are introduced for preemptive offloading. In this chapter we show that the proposed algorithm is energy optimal and is referred to as *Optimal Preemptive Offloading (OPO)*. This algorithm is optimal in the sense that no other online computation offloading algorithm can achieve a lower mean mobile device energy consumption. The energy optimality of this algorithm is shown by creating a time-dilated absorbing Markov processes and using optimal Markovian stopping theory. More specifically, it is shown that a dynamic programming approach can be used to compute the expected energy cost in case offloading occurs or not in the current time slot, and the algorithm bases its (optimal) decision on these costs.

Since the computational complexity of OPO can be significant, we introduce three computationally efficient techniques, motivated by OPO, namely, *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. These methods operate by first identifying a target set of future time slot types to use for the offload. This set is defined based on predicted bit rates using the Markovian channel statistics. Time slots are then assigned using "water-filling", which prioritizes those with higher expected bit rates. The algorithms vary based on how the computations are performed. For each method, two variations are considered. The first (*Equ*) uses the equilibrium channel state probabilities in its offloading decision calculations. The second variant (*Exp*), while more accurate, uses Markovian transition matrix exponentiation, which is more computationally expensive. This results in six algorithms, namely, *Water-Filling with Equilibrium (WF-Equ)*,

Water-Filling with Exponentiation (WF-Exp), *Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch)*, *Water-Filling with Exponentiation and Scheduling (WF-Exp-Sch)*, *Generalized Water-Filling with Equilibrium (Gen-WF-Equ)* and *Generalized Water-Filling with Exponentiation (Gen-WF-Exp)*. The performance of the proposed algorithms is compared on Markovian channels with different characteristics, in order to show the tradeoffs between complexity and mobile energy saving performance.

The main contributions of this chapter are summarized as follows:

- Preemption is introduced to concurrent local execution, used in mobile computation offloading in order to ensure that hard job execution deadlines are satisfied. Compared to previous chapters where offloading occurs using known job offload parts, allowing preemption results in a much more complex problem formulation. This happens because the number and size of the job pieces offloaded are not known in advance and must be determined by the online offloading algorithm. Due to this complexity, it is not feasible to run the optimal online decision algorithm in real time, and this motivates the use of the proposed computationally efficient techniques.

- An online offloading decision algorithm, i.e., Optimal Preemptive Offloading (OPO), is introduced. It is shown that OPO satisfies hard deadline application constraints, and also achieves the minimum mean mobile device energy possible for homogeneous Markovian wireless channels.

- An integer program (IP) is formulated that provides a lower bound on mobile device energy. This calculation is used for comparisons with online algorithms in the results section.

- We introduce three computationally efficient techniques based on OPO: *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. The algorithms vary based on how the reduced offloading decision computations are performed. For each method, two variations (*Equ* and *Exp*) are introduced. The algorithms all employ concurrent local execution so that task execution time constraints are always satisfied.

- Performance results are presented that show the various complexity and energy performance tradeoffs for the proposed algorithms.

The rest of this chapter is organized as follows. In Section 5.2, system modelling assumptions are presented that include both remote and local execution where hard task execution deadlines are ensured using CLE. In Section 5.3, an offline lower bound on energy consumption is derived, which is compared to the proposed computation offloading algorithms in the results section. Following this, in Section 5.4 the Markovian channel model is discussed and how it is used to define an absorbing Markov chain that permits us to show the energy optimality of the proposed algorithm, OPO, which is introduced in Section 5.4. Section 5.5 then introduces the online algorithms motivated by OPO, but with varying degrees of running time reduction. Finally, in Section 5.6, results are presented that compare the various algorithms and show the tradeoffs between computational complexity and mobile energy savings.

5.2 System Model

In this chapter, we follow the system model presented in Chapter 3 with a few differences regarding the preemptive job uploading scenario, which will be highlighted below.

Preemptive offloading is illustrated in Figure 5.1. In this case, the offloading decision is made at each particular time slot and a piece of the job will be uploaded only at that time slot if the algorithm decides to do so i.e., *preemption* is allowed. This process continues for all subsequent time slots until the job is completely uploaded to the server. In Figure 5.1, t_{o_i} is the time slot for offloading the ith job piece, followed by T_{W_i} time slots of waiting before the next uploading. Remote execution will be completed $t_{o_n} - t_r + 1 + T_{exec} + T_{down}$ time slots after t_r, where t_{o_n} is the optimal offloading time of the last piece of job.

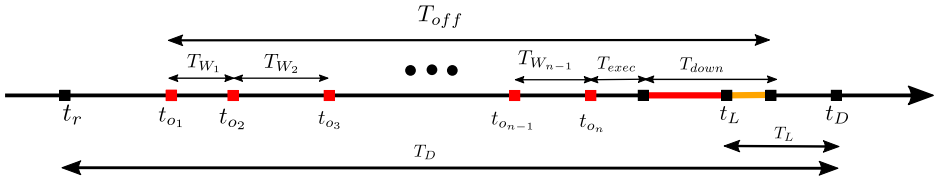


Figure 5.1: Concurrent local and remote preemptive computational job offloading time line

As shown in the figure, at the time of remote execution completion, local execution is terminated, provided that the remote offload response arrives before t_D. As can be seen, the total offloading time is $T_{off} = t_{o_n} - t_{o_1} + 1 + T_{exec} + T_{down}$. It is assumed that the channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, T_{off} is a random variable, dependent on the evolution of the uplink channel state as a given upload occurs.

Note that the satisfaction of hard job execution deadline constraints would be problematic if there were uncontrolled preemption in the mobile device itself. For this reason, we have assumed that the features normally associated with a real-time operating system are in place so that the job execution is assigned a known fraction of the local processor. In this way, T_L can be calculated when the task is released. Hence, according to the CLE model we are proposing, there may be overlapping of the local and remote executions of

the task, but the hard job deadline is *always* respected, even if there is channel contention or extended channel outages.

In the figure, local execution has started before the remote offload process has completed, and therefore local execution has started, shown in orange. The problem addressed in this chapter is deciding the best offloading time slots $t_{o_1}, t_{o_2}, \ldots$, i.e., offloading time slots which minimize the expected total energy cost.

In the mobile energy consumption formulations described in this chapter, we have chosen not to include the local energy requirements needed to execute the online scheduling algorithms. This is consistent with the common assumption that the energy needed for offloading the job data is large, by comparison. However, in Section 5.6, graphs of the relative running times of the algorithms have been included so that this component can be accounted for, if required.

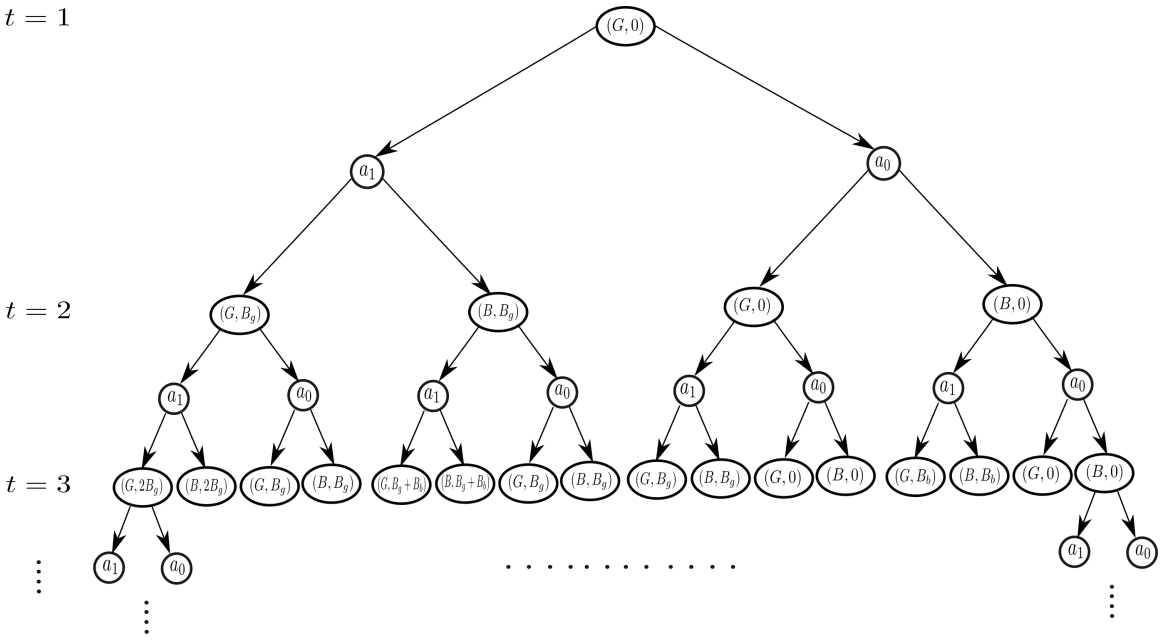


Figure 5.2: Time dilated absorbing Markov process (TDMP) for a two-state (G, B) channel.

5.3 Offline Bound

In this section, an offline energy lower bound for the preemptive offloading scenario, presented in this chapter, is achieved. This lower energy bound is used in Section 5.6 for performance comparisons with various online computation offloading algorithms proposed in this chapter. Similar to Chapter 3, since the bound is offline, we assume that the wireless channel states are known beforehand, i.e., we know the bit rate (in bits per time slot) at all time slots $1 \leq t \leq t_D$ (recall that we have set $t_R = 1$). When a job is released, the bound chooses the job offload times so that its deadline is satisfied and at the same time total energy consumption to offload the job, is minimized.

Let x_i be the decision of offloading at time slot t_i, i.e., $x_i = 0$ when we decide to offload at time slot i, and $x_i = 1$ otherwise. Let t_f be the upload finishing time. B_i is the bit rate at time slot i, and E_{tr} is the energy cost per time slot for channel transmitting. The optimal x_i's are found by first solving the following set of IPs (one for each possible finishing time t_f), and then output the minimum amongst them and E_L (if E_L is the minimum, then the optimal solution is to not offload at all).

- $t_f < t_L - T_{rest}$

$$\min_{x_1,\cdots,x_{t_f}} E_{tr} \sum_{i=1}^{t_f} x_i + T_{down} E_{rc} \tag{5.3.1}$$

$$\text{s.t.} \sum_{i=1}^{t_f} B_i x_i \geq S_{up} \tag{5.3.2}$$

$$x_{t_i} \in \{0, 1\} \text{ for } i = 1, 2, \cdots, t_f - 1 \tag{5.3.3}$$

$$x_{t_f} = 1 \tag{5.3.4}$$

- $t_L - T_{rest} \leq t_f \leq t_D$

$$\min_{x_1,\cdots,x_{t_f}} E_{tr} \sum_{i=1}^{t_f} x_i + \frac{t_f + T_{rest} - t_L + 1}{T_L} E_L + T_{down} E_{rc} \tag{5.3.5}$$

$$\text{s.t.} \sum_{i=1}^{t_f} B_i x_i \geq S_{up} \tag{5.3.6}$$

$$x_{t_i} \in \{0, 1\} \text{ for } i = 1, 2, \cdots, t_f - 1 \tag{5.3.7}$$

$$x_{t_f} = 1 \tag{5.3.8}$$

The first case corresponds to finishing offloading before local execution begins, and the second to finishing offloading after local execution begins. The first term in (5.3.1) and (5.3.5) is the uploading energy cost. The second term in (5.3.5) is the portion of the local energy cost we incur if offloading finishes at t_f. Constraints (5.3.2) and (5.3.6) ensure all job bits are offloaded. Constraints (5.3.4) and (5.3.8) ensure that uploading finishes at t_f.

5.4 Optimal Algorithm for Preemptive Offloading

In this section, we develop the online algorithm OPO (Optimal Preemptive Offloading), and prove its optimality. In order to simplify our exposition of the algorithm, we will assume that all offloading deadlines, job sizes (in bits), and energy costs are related only to job uploading, i.e., we assume that T_{exec}, T_{down} are known and already incorporated in the energy costs and the job deadline. Given the ensuing results, adding the effects of T_{exec} and T_{down} is straightforward.

As mentioned above, 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. Each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded in that state. The CSMC transition matrix is defined as $\mathbb{P} = [P_{i,j}]$, where $P_{i,j}$ is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. As defined, the CSMC is memoryless, but in what follows we will need to incorporate time in it, the already offloaded bits, and the decision of uploading or not at every time slot. Therefore, we construct a new Markov decision process, the *time-dilated Markov process (TDMP)* as follows: For every time $1 \leq t \leq t_D + 1$, we define a set of states (X_t, S_t), where X_t is a channel state and $0 \leq S_t \leq S_{up}$. We can think of the states as arranged in layers for $t = 1, 2, \ldots, t_D + 1$. The set of actions contains two actions, a_0, a_1, corresponding to not offloading, or offloading, respectively. Figure 5.2 shows the initial layers for a TDMP for a Gilbert-Elliot channel with two states, G and B, with bitrates B_g, B_b, respectively. A state (X_t, S_t) branches to a_0 and a_1, and then a_0 branches to states (X_{t+1}, S_t) with probabilities $P_{X_t, X_{t+1}}$, and a_1 to states $(X_{t+1}, S_t + \min\{B_{X_t}, S_{up} - S_t\})$ with the same probabilities, where B_{X_t} is the bit rate of channel state X_t. States of the form (X_t, S_{up}) lead only to action a_0 (no offloading). At layer $t = 1$ there is only one state $(X_1, 0)$, where X_1 is the initial channel state, while all states at layer $t_D + 1$ are absorbing.

The energy cost when offloading in time slot t is given by:

$$E_{off}(t) = \begin{cases} E_{tr}, & t_R \leq t \leq t_D \\ 0, & t < t_R \text{ or } t > t_D \end{cases} \tag{5.4.1}$$

The energy cost for the local execution between the previous offloading time slot t_{prev}

and the current time slot t (note that $t > t_{prev}$) is given by:

$$E_L(t_{prev}, t) = \begin{cases} 0, & t < t_L \text{ or } t_{prev} \geq t_D \\ \frac{\min\{t, t_D\} - \max\{t_{prev}+1, t_L\}+1}{T_L} E_L, & \text{o/w} \end{cases} \tag{5.4.2}$$

The expected energy cost of offloading, when offloading occurs at time t in TDMP state X_t, the last offloading time slot is t_{prev}, and S_t bits have already uploaded, is

$$g(X_t, S_t) = E_{off}(t) + E_L(t_{prev}, t) + \sum_{X_{t+1} \in \mathcal{M}} Pr[X_{t+1}|X_t] V(t, X_{t+1}, S_t + B_{X_t}) \tag{5.4.3}$$

Note that the definition of g includes energy costs due to local execution that were accumulated in the time slots between the last offloading at t_{prev} and the current one at t.

For every time slot t and state X_t, and with the last uploading time slot being t_{prev}, we define the expected cost $V(t_{prev}, X_t, S_t)$ recursively in (5.4.4). In (5.4.4), we have

$$V(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \frac{t_D - \max\{t_{prev}+1, t_L\}+1}{T_L} E_L, & t > t_D > t_{prev} \text{ and } S_t < S_{up} \\ \min\{g(X_t, S_t), E[V(t_{prev}, X_{t+1}, S_t)|X_t]\}, & t_D \geq t > t_{prev} \text{ and } S_t < S_{up} \end{cases} \tag{5.4.4}$$

$$E[V(t_{prev}, X_{t+1}, S_t)|X_t] = \sum_{X_{t+1} \in \mathcal{T}} Pr[X_{t+1}|X_t] V(t_{prev}, X_{t+1}, S_t),$$

and $\mathcal{T}$ is the set of states reachable after running the channel for $t+1$ time slots. The first case has an expected cost of 0, since there is no more offloading. The second case incurs only local energy consumption, because the time t is beyond the deadline t_D, and,

therefore, offloading does not happen. The third case assigns as the expected cost the minimum between the expected cost of offloading at t, and the expected cost of postponing that decision to time slot $t+1$.

The formal definition of algorithm OPO is Algorithm 3. Recall that, for simplicity, we have set $t_R = 1$. A high-level description of the algorithm is as follows: At time slot $t = 1$

Algorithm 3 OPO (Optimal Preemptive Offloading)

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

```
1: for all t = 1, ..., t_D do
2:     if uploading the whole job is finished then
3:         Break
4:     end if
5:     if min in (5.4.4) is g then
6:         upload at time slot t
7:     end if
8: end for
```

the job is released. At each time slot t, and with S_t bits already offloaded in past offloading time slots, the algorithm considers the TDMP, in order to determine the expected cost of the whole remaining offloading process, if uploading happens at time slot t. If that cost is less than the expected offloading cost when the algorithm waits one more time slot, then $t^* = t$ (t becomes an offloading time slot, with bit rate B_{X_t}, and $S_{t+1} = S_t + B_{X_t}$); otherwise, the algorithm postpones its decision to time slot $t+1$, and $S_{t+1} = S_t$. The algorithm repeats the same decision process at every time slot (using the TDMP corresponding to the bits offloaded so far, to compute expected costs), in order to determine the sequence of optimal offloading time slots $\{t_1^*, t_2^*, t_3^*, \ldots\}$. $V(t_{prev}, X_t, S_t)$ in Line 5 can be computed using Dynamic Programming (DP).

We use the theory of optimal stopping for Markov decision processes Peskir and Shiryaev (2006), in order to show that it achieves the optimal expected energy for the

mobile device, i.e., no other online computation offloading algorithm can achieve a lower mean mobile device energy consumption. This is done by proving that at any time slot t, the algorithm makes exactly the same decision as an algorithm that has the ability, starting from t, to decide *all* future offloading decisions that minimize the expected energy cost. The high level idea of the proof of optimality is as follows: Algorithm OPO is an *on-line* algorithm that runs at each time slot t. At this time slot t, an expected energy consumption minimization problem can be defined, which computes *all* optimal offloading time slots in the time period from t to deadline t_D, given the last offloading time slot t_{prev} and the number of job bits S_t that have already been offloaded. This minimization problem is defined recursively, by calculating the expected energy cost of *future optimal* offloading decisions for each possible next offloading time $t, t+1, t+2, \ldots, t_D+1$; let $t^* \geq t$ be the next offloading time which achieves the minimum expected energy. We will prove that if $t^* > t$, algorithm OPO also decides to *not* offload at t, and if $t^* = t$ algorithm OPO also decides to offload at t. Note that t^* is the first of a series of optimal offloading time slot decisions that minimize the expected energy cost *given our current knowledge of the channel*. At every time slot, the latter changes (the new channel state is revealed), algorithm OPO has to make a new decision, and a new minimization problem is defined.

The minimization problem is formulated recursively as in (5.4.5), where $\mathcal{S}$ is the set of states reachable after running the channel for t^* time slots, $\mathcal{M}$ is the set of states the channel can transit to from X_{t^*}, $B_{X_{t^*}}$ is the bit rate of the state X_{t^*}, and $v(t^*, X_{t^*+1}, S_t + B_{X_{t^*}})$ is the optimal expected energy cost for the rest of the offloading, when the algorithm decides to upload $B_{X_{t^*}}$ bits at time t^*. We set $S_0 = 0$. Note that problem (5.4.5) is defined only for $t > t_{prev}$, i.e., after the last offloading time slot. In order to prove the optimality of on-line algorithm OPO, we prove that, for all $i = 1, 2, \ldots$, when algorithm OPO offloads for the

$$v(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \min\limits_{t \leq t^* \leq t_D+1} \Bigg\{ \sum\limits_{X_{t^*} \in \mathcal{S}} Pr[X_{t^*}|X_t] \Big(E_{off}(t^*) + E_L(t_{prev}, t^*) + \\ \sum\limits_{X_{t^*+1} \in \mathcal{M}} Pr[X_{t^*+1}|X_{t^*}] v(t^*, X_{t^*+1}, S_t + B_{X_{t^*}}) \Big) \Bigg\}, \\ & t_{prev} < t \leq t_D \text{ and } S_t < S_{up} \end{cases} \tag{5.4.5}$$

i-th time, i.e., $V(t_{i-1}, X_t, S_t) = g(X_t, S_t)$ in Line 5, the maximization problem optimal solution also offloads, i.e., $t_i^* = t$.

Theorem 5. *The sequence of optimal times* $\{t_0^* = 0, t_1^*, t_2^*, \ldots\}$ *for uploading satisfies*

$$t_i^* = \underset{t_{i-1}^* < t \leq t_D}{\arg\min} \{V(t_{i-1}^*, X_t, S_t) = g(X_t, S_t)\},\ i = 1, 2, \ldots, last$$

Proof. We prove the theorem by induction on i. The base case of $i = 0$ is trivially true. We assume that it is true up to $i = k - 1$, i.e., the $k - 1$ previous offloading decision times of OPO coincide with the first offloads $t_0^*, t_1^*, \ldots, t_{k-1}^*$ of the maximization problems defined at time slots $t_0^*, t_1^*, \ldots, t_{k-1}^*$. We prove the case of t_k^*. In what follows, whenever definitions (5.4.3), (5.4.4) are used, $t_{prev} := t_{k-1}^*$ (since the immediately previous offloading time for OPO is t_{k-1}^*, by the inductive hypothesis).

First, using (reverse) induction on t, S_t, and given t_{k-1}^*, S_{up}, we show the following:

$$\forall t > t_{k-1}^*, S_t \leq S_{up} : v(t_{k-1}^*, X_t, S_t) = V(t_{k-1}^*, X_t, S_t).$$

The base case of $t > t_D$ and $S_t \leq S_{up}$ is obviously true. Assuming that the equation holds for all time values $t + 1, t + 2, \ldots$ and all size values $0 \leq S_{t+1}, S_{t+2}, \ldots \leq S_{up}$, we can

$$v(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \min\limits_{t \leq t^* \leq t_D+1} \Bigg\{ \sum\limits_{X_{t^*} \in \mathcal{S}} Pr[X_{t^*}|X_t] \Big(E_{off}(t^*) + E_L(t_{prev}, t^*) + \\ \sum\limits_{X_{t^*+1} \in \mathcal{M}} Pr[X_{t^*+1}|X_{t^*}] V(t^*, X_{t^*+1}, S_t + B_{X_{t^*}}) \Big) \Bigg\}, \\ & t_{prev} < t \leq t_D \text{ and } S_t < S_{up} \end{cases} \tag{5.4.6}$$

show that it is also true for time t and all $0 \leq S_t \leq S_{up}$, by applying Theorem 1.7 in Peskir and Shiryaev (2006).

Then $v(t_k^*, X_{t_k^*+1}, S_t + B_{X_{t_k^*}})$ in the RHS of (5.4.5) can be replaced by $V(t_k^*, X_{t_k^*+1}, S_t + B_{X_{t_k^*}})$, to get the RHS of (5.4.6). By this substitution, the original maximization problem (5.4.5) is no longer recursive (i.e., dependent on the future v values), but is transformed to an equivalent minimization problem (5.4.6), that is a function of (computable, using DP) V. Then, the definition (5.4.3) of $g(X_t, S_t)$ implies that the optimal solution of (5.4.6) (which is also the optimal solution for problem (5.4.5)) can be obtained at any time slot t by performing the test of Line 5 in OPO, which is the property in the theorem statement.

□

Compared to the previous chapters, which studied the same problem when offloading is done in a *predetermined* number of job pieces, each with *known* bit size, preemption changes the nature of the problem significantly, since the number and sizes of the offloaded job pieces are initially *unknown* (note that the upper bound $last$ for i in the statement of Theorem 5 is unknown). This means that, in preemption, the number of offloads is implicitly a decision variable, together with the exact offloading times. The crucial idea of the analysis above is that by allowing the DP to store some extra information (the number of bits that have already been offloaded), the recursive definition of the minimization problem

(5.4.5) and expected cost (5.4.4) do not need to know the number of offloads. Therefore, although the optimal algorithm and its analysis look similar to those in Chapters 3 and 4, preemption requires a more complicated (and computationally costly) treatment, both in its computations and its analysis (e.g., note the double induction needed in the proof of Theorem 5).

5.5 Practical Heuristics

Although algorithm OPO is provably optimal, its computational complexity is of order $\Theta(S_{up}^{f(S_{up})})$, for some polynomial function f. Therefore, the running time of the algorithm is prohibitive, even when we consider the simple two-state Gilbert-Elliot Markovian channel, T_D is not much larger than $\frac{S_{up}}{B_{min}}$ (otherwise the optimal algorithms always offload when at high bit-rate states), and even when S_{up} is relatively small. Hence, good heuristics that may be suboptimal, but run fast enough to be used on-line, have to be developed.

We present three such heuristics, motivated by the optimal OPO algorithm. We observe that the prohibitively slow step in Algorithm 3 is line 5, where the DP calculation of g is performed at every time slot, in order to compute the exact expected energy consumption for offloading the remaining task bits. Our heuristics will maintain the flexibility allowed by preemption, by also running at every time slot. But they replace the costly calculation of line 5 in OPO with an approximation of the expected energy cost, using either the *equilibrium* or *invariant* probabilities π, i.e., the solution to equation $\pi P = \pi$, where P is the transition matrix of the MC (e.g., see lines 27-30 in Algorithm 4), or transition matrix exponentiation (e.g., see lines 32-33 in Algorithm 5). Hence, for each heuristic, there are two variations: the first (Equ) uses equilibrium probabilities, and the second (Exp), which

is more accurate but also more computationally-intensive, uses transition matrix exponentiation.

5.5.1 Water-Filling

In wireless communications, water-filling is a fundamental power allocation mechanism for capacity maximization under a given total transmit power (Ling *et al.* (2012); He *et al.* (2013)). Here, we have used the general idea behind the conventional water-filling algorithm to develop our heuristics. The basic scheme of the proposed algorithm is the computation of a "most efficient" set $\mathcal{F}$ of channel states in a greedy "water-filling" fashion, and offloading only when the current channel state is in this set, provided that offloading is beneficial at all. In accordance with the variations discussed above, there are two water-filling variants, as follows.

- **Water-Filling with Equilibrium (WF-Equ) (Algorithm 4)**: In this variation, the algorithm is given the equilibrium probabilities π (e.g., computed in preprocessing). As described above, these probabilities are used in order to compute a "most efficient" set $\mathcal{F}$ of channel states. This set is used in a greedy "water-filling" fashion, i.e., offloading only when the algorithm finds itself in this set (if it decides to offload at all). More specifically, the channel states are ordered from the highest to the lowest bit rate. Let m be the state with the highest bit rate from the state space M. If there is nothing remaining to offload, the algorithm terminates (lines 2-4). Initially $\mathcal{F} = \{m\}$ (line 5). Lines 7-20 calculate the expected finishing time to offload the remaining job, if only states in the current $\mathcal{F}$ are used, with an average bit rate $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$ ($Br[i]$ is the state i bitrate). As long as the states in $\mathcal{F}$ are not sufficient to meet the deadline, the next highest bit-rate state is added to $\mathcal{F}$, until

either there are no more states to add, and the offloading is aborted (lines 21-23), or the algorithm proceeds with the offloading decision. Namely, if the current state does not belong to $\mathcal{F}$, no offloading happens at the current t (lines 24-26); otherwise, the offloading of $Br(X_t)$ occurs, if the expected offloading energy is still less than that using local execution (lines 27-39).

- **Water-Filling with Exponentiation (WF-Exp) (Algorithm 5)**: This variation uses exponentiation of the MC transition matrix in order to calculate the expected finishing time and the number of offloading time slots to offload the remaining job. Obviously, this is a more accurate approximation than the one performed by Algorithm 4, since it takes into account the current state and the exact number of steps needed in order to reach another state. More specifically, Algorithm 5 replaces lines 8-10, 16-18, 27-29 of Algorithm 4 with lines 14-16, 21-23, 32, respectively.

5.5.2 Water-Filling with Scheduling

This algorithm is a more sophisticated version of the previous Water-Filling algorithm. Again, there are two variations, one that uses equilibrium probabilities for its computation, and one that uses transition matrix exponentiation.

- **Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch) (Algorithm 6)**: The algorithm works similarly to WF-Equ, with the important difference that it tries to figure out the *best* offloading finishing time t_f. In order to do that, the algorithm goes over all possible values for t_f from t to t_D. For each t_f, it applies the calculations of Algorithm 4, but on time range $t_f - t$ (instead of $t_D - t$), in order to calculate the expected offloading energy cost. Note that sets $\mathcal{F}$ may differ for different t_f. Then, the algorithm picks the minimum offloading energy consumption calculated over all

Algorithm 4 Water-Filling with Equilibrium (WF-Equ)

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, equilibrium probabilities π, state space of the Markov chain $\mathcal{M} = \{1, 2. \ldots, m\}$, current state X_t, remaining size $S_r = S$.

for all $t = 1, \ldots, t_D$ **do**
  if $S_r = 0$ **then**
    break
  end if
  $\mathcal{F} = \{m\}$ ▷ m is the highest bit-rate state
  $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$
  if $X_t \notin \mathcal{F}$ **then**
    $t_f = t + \frac{S_r}{B_{avg}} + T_{rest}$
  else
    $t_f = t + \frac{S_r - Br(X_t)}{B_{avg}} + T_{rest}$
  end if
  while $(t_f > t_D)\ and\ (\mathcal{M} \neq \mathcal{F})$ **do**
    add the state with the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ to $\mathcal{F}$
    $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$
    if $X_t \notin \mathcal{F}$ **then**
      $t_f = t + \frac{S_r}{B_{avg}} + T_{rest}$
    else
      $t_f = t + \frac{S_r - Br(X_t)}{B_{avg}} + T_{rest}$
    end if
  end while
  if $t_f > t_D$ **then**
    break
  end if
  if $X_t \notin \mathcal{F}$ **then**
    continue ▷ Move to time $t + 1$
  end if
  $B_F = \frac{\sum_{i \in \mathcal{F}} \pi[i] Br[i]}{\sum_{i \in \mathcal{F}} \pi[i]}$
  $E_{up} = (1 + \frac{\max\{0, S_r - Br(X_t)\}}{B_F}) E_{tr}$
  $t_f = t + \frac{\max\{0, S_r - Br(X_t)\}}{B_{avg}} + T_{rest}$
  $E_l = (\max(t_f + 1, t_L) - \max(t, t_L)) E_L / T_L$
  if $t_f \leq t_D$ **then**
    $E_{off} = E_{up} + E_l + T_{down} E_{rc}$

33: **else**
34: $E_{off} = \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$
35: **end if**
36: **if** $E_{off} < \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ **then**
37: Offload at t
38: $S_r = \max\{0, S_r - Br(X_t)\}$
39: **end if**
40: **end for**

finishing times and compares this minimum value to the energy required to process the rest of the job locally and offloads at t if the former is smaller than the latter. Algorithm 6 is the equilibrium variation of Water-Filling with Scheduling.

- **Water-Filling with Exponentiation and Scheduling (WF-Exp-Sch)**: This variation of WF-Equ-Sch uses transition matrix exponentiation instead of equilibrium probabilities in its calculations, exactly as WF-Exp does.

5.5.3 Generalized Water-Filling

This algorithm is a generalization of the first Water-Filling algorithm, and its two variations are as follows:

- **Generalized Water-Filling with Equilibrium (Gen-WF-Equ)**: This algorithm is a generalization of WF-Equ. Instead of defining a single set $\mathcal{F}$ (when it exists) as the minimal prefix of the state ordering according to bitrates that allow offloading before the deadline, it considers *all* possible such prefixes, and keeps the best. More specifically, if the states $S_1, S_2, \ldots, S_M$ are ordered in decreasing bitrates, and $\mathcal{F}_1$ as defined by Algorithm 4 is $\mathcal{F}_1 = \{S_1, S_2, \ldots, S_l\}$, then the algorithm repeats the same computations for sets $\mathcal{F}_1 = \{S_1, S_2, \ldots, S_l\}, \mathcal{F}_2 = \{S_1, S_2, \ldots, S_{l+1}\}, \ldots, \mathcal{F}_{M+1-l} = \{S_1, S_2, \ldots, S_M\}$, and keeps the set of minimum expected offloading energy costs;

Algorithm 5 Water-Filling with Exponentiation (WF-Exp)

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, transition probability matrix P, state space of the Markov chain $\mathcal{M} = \{1, 2. \ldots, m\}$, current state X_t, remaining size $S_r = S$.

for all $t = 1, \ldots, t_D$ **do**
  if $S_r = 0$ **then**
    break
  end if
  for all $i = 1, \ldots, m$ **do**
    if $i == X_t$ **then**
      $\alpha[i] = 1$
    else
      $\alpha[i] = 0$
    end if
  end for
  $\mathcal{F} = \{m\}$ ▷ m is the highest bit-rate state
  if $X_t \notin \mathcal{F}$ **then**
    Find minimum t_f s.t. $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
  else
    Find minimum t_f s.t. $S_r - Br(X_t) - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
  end if
  while $(t_f > t_D)\ and\ (\mathcal{M} \neq \mathcal{F})$ **do**
    add the state with the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ to $\mathcal{F}$
    if $X_t \notin \mathcal{F}$ **then**
      Find minimum t_f s.t. $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
    else
      Find minimum t_f s.t. $S_r - Br(X_t) - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
    end if
  end while
  if $t_f > t_D$ **then**
    break
  end if
  if $X_t \notin \mathcal{F}$ **then**
    continue ▷ Move to time $t + 1$
  end if
  $E_{up} = \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot E_{tr}$
  $E_l = (\max(t_f + 1, t_L) - \max(t, t_L))E_L/T_L$

if $t_f \leq t_D$ **then**
  $E_{off} = E_{up} + E_l + T_{down} E_{rc}$
else
  $E_{off} = \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$
end if
if $E_{off} < \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ **then**
  Offload at t
  $S_r = \max\{0, S_r - Br(X_t)\}$
end if
end for

Algorithm 6 Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch)

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, equilibrium probabilities π , total number of states M, current state X_t, remaining size S_r.

for all $t = 1, \ldots, t_D$ **do**
  for all $t_f = t, \ldots, t_D$ **do**
    Calculate the expected offloading cost $E_{off}^{(t_f)}$ for period $[t, \ldots, t_f]$ using Algorithm 4
  end for
  $E_{off} = \min_{t \leq t_f \leq T_D} E_{off}^{(t_f)}$
  if $E_{off} < \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ **then**
    Offload at t
  end if
end for

then it proceeds exactly as in WF-Equ.

- **Generalized Water-Filling with Exponentiation (Gen-WF-Exp)**: This algorithm is the same as Gen-WF-Equ, except that it uses transition matrix exponentiation instead of equilibrium probabilities in its calculations.

As will become apparent by the simulation results in Section 5.6, the use of exponentiation by our heuristics dramatically increases their running times, and, therefore, their use is mostly as benchmarks for their versions that use equilibrium calculations.

5.6 Simulation Results

In this section, computer simulation is used to study the performance of the proposed preemptive offloading algorithms. In accordance with concurrent local execution, in all algorithms, local execution starts at time slot t_L if offloading (includes uploading to, remote execution at, and downloading from the server) is not completed at time slot $t_L - 1$. This ensures that the job deadline is satisfied either locally or remotely. In order to investigate the performance of the preemptive offloading algorithms in different wireless transmission conditions, we use four different Markov chains to model the communication channel. Two channels are modelled by a 9-state Markov chain shown in Figure 5.3 and two are modelled by a 5-state Markov chain shown in Figure 5.4. In both cases, state 1 represents the channel outage that may occur because of shadowing or other similar phenomena; and for the other states, higher states have higher bit rates. In the 9-state channels, the assigned bit rates to states 1 to 9 are $0, 6, 9, 12, 18, 24, 36, 48, 54$ K bits per time slot, respectively, which are the data rates of the IEEE 802.11g standard. In the 5-state channels, the assigned bit rates to states 1 to 5 are $0, 12.5, 25, 37.5$ and 50 K bits per time slot, respectively. For each of the Markovian channel models, we consider two state transition probability matrices, as given in (5.6.1)-(5.6.4), where $\mathbb{P}_1$ and $\mathbb{P}_2$ are for the 9-state model, and $\mathbb{P}_3$ and $\mathbb{P}_4$ are for the 5-state model. The channels with $\mathbb{P}_1$ and $\mathbb{P}_3$ have equal steady state probability at all the states and are referred to as "uniform channels", and that with $\mathbb{P}_2$ and $\mathbb{P}_4$ have higher steady state probabilities at the lowest (outage) and the highest states and are referred to as "non-uniform channels". The intent behind the channel model selections is to create channels with varying degrees of channel state predictability, so that differences between the algorithms can be properly assessed. All these channels have approximately the same average channel bit rate.

$$\mathbb{P}_1 = \begin{bmatrix} 0.12 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.10 & 0.55 & 0.35 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.60 \end{bmatrix} \tag{5.6.1}$$

$$\mathbb{P}_2 = \begin{bmatrix} 0.84 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.10 & 0.05 & 0.85 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.05 & 0.85 \end{bmatrix} \tag{5.6.2}$$

$$\mathbb{P}_3 = \begin{bmatrix} 0.40 & 0.15 & 0.15 & 0.15 & 0.15 \\ 0.15 & 0.55 & 0.30 & 0.00 & 0.00 \\ 0.15 & 0.30 & 0.25 & 0.30 & 0.00 \\ 0.15 & 0.00 & 0.30 & 0.25 & 0.30 \\ 0.15 & 0.00 & 0.00 & 0.30 & 0.55 \end{bmatrix} \tag{5.6.3}$$

$$\mathbb{P}_4 = \begin{bmatrix} 0.88 & 0.03 & 0.03 & 0.03 & 0.03 \\ 0.10 & 0.05 & 0.85 & 0.00 & 0.00 \\ 0.10 & 0.30 & 0.05 & 0.55 & 0.00 \\ 0.10 & 0.00 & 0.30 & 0.05 & 0.55 \\ 0.10 & 0.00 & 0.00 & 0.05 & 0.85 \end{bmatrix} \tag{5.6.4}$$

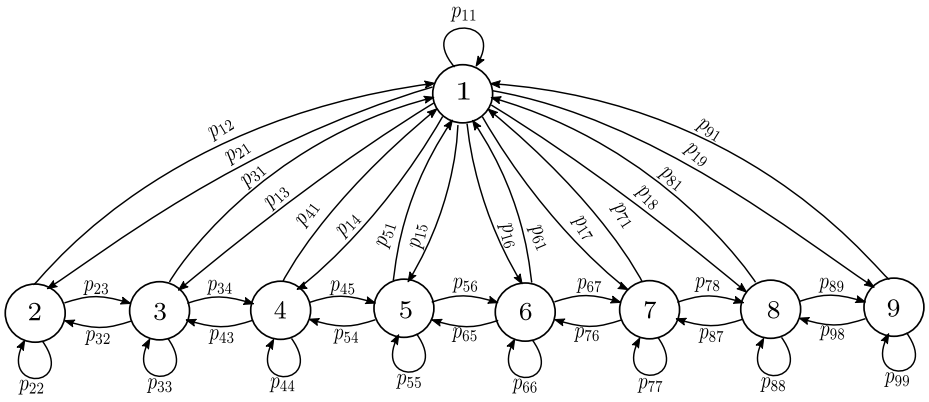


Figure 5.3: Markov Chain for $\mathbb{P} = \mathbb{P}_1$ and $\mathbb{P}_2$

The default parameters used in the simulations are given as follows. Each time slot is taken to be 1 msec, which is also used to normalize all system time values. The transmit and receive power is 1 W and 0.5 W, respectively, which corresponds to the transmission and

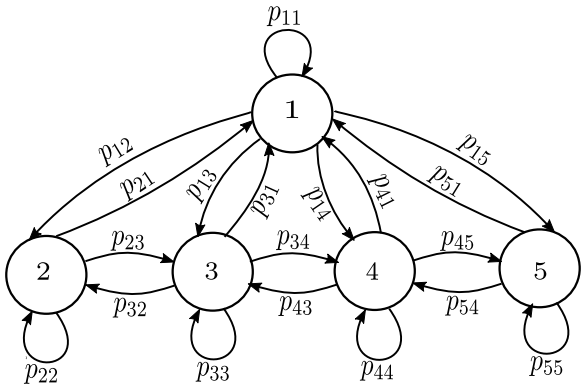
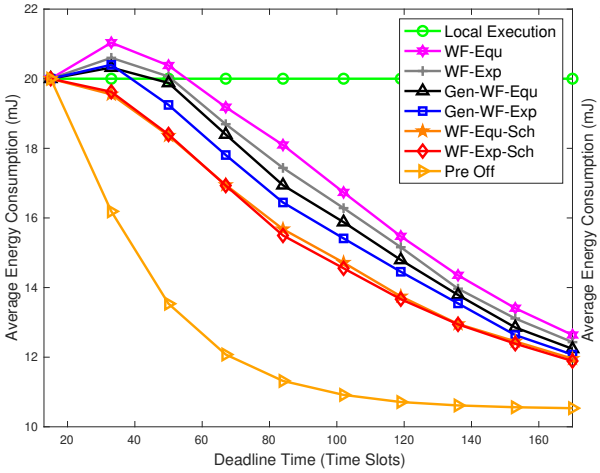


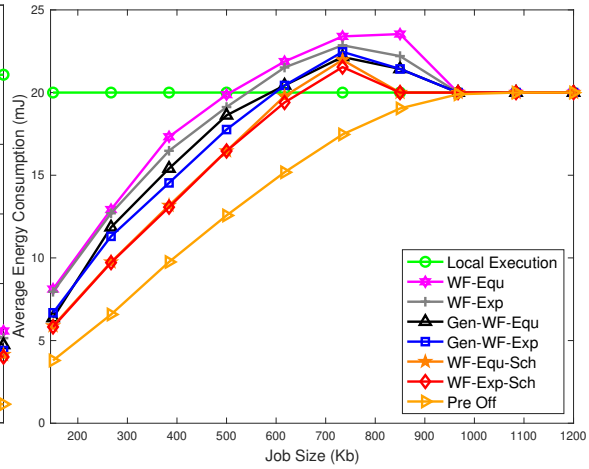
Figure 5.4: Markov Chain for $\mathbb{P} = \mathbb{P}_3$ and $\mathbb{P}_4$

receive energy during each time slot as $E_{tr} = 1$ mJ and $E_{rc} = 0.5$ mJ, respectively. A job with computational load $D = 10$M CPU cycles is considered. The job completion deadline T_D is set to 60 time slots. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$ mJ and the local computation speed is $f_l = 1$M CPU cycles per time slot, see (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. We consider that the remote execution time is $T_{exec} = 1$ time slot, i.e., the remote processing speed is 10 times that of local processing. The download time T_{down} is assumed to be 1 time slot. In all the simulations, we collect both average energy consumption of the mobile device and the running time of the algorithms, where the running time is the average amount of time needed to make the offloading decision at one time slot. Each value of average energy consumption and running time is obtained after repeating the simulation for 1,000 runs. In addition, for the exponentiation-based algorithms, namely, WF-Exp, Gen-WF-Exp, and WF-Exp-Sch, the calculation of the channel matrix exponentiation is not counted in the running time because this work can be done in the background before the mobile device initiates an offload.

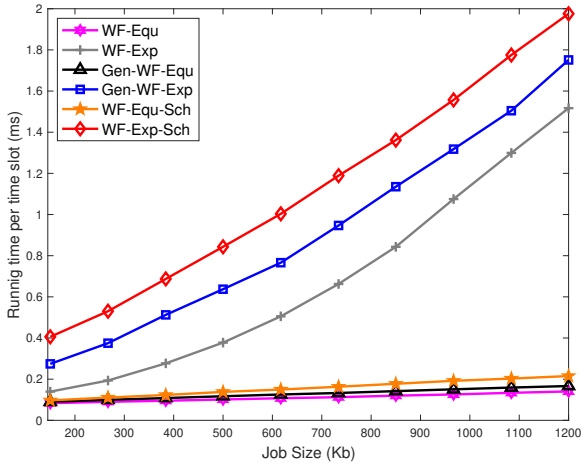
For comparison, we also plot the offline bound (Pre Off) given in Section 5.3 and *Local*



(a) S_{up}=500 kb



(b) T_D=60 ms



(c) T_D=60 ms

Figure 5.5: 9-states uniform distribution: $\mathbb{P} = \mathbb{P}_1$

Execution that executes the entire job locally without doing any offloading. When collecting the simulation results, the energy consumption for running the online algorithm at the mobile device was assumed to be negligible compared to that for transmitting to the cloud server. This is a common assumption when the amount of data for uploading the task is large. Despite this, we have included graphs of the relative running times of the algorithms so that this component could be included if required.

Figure 5.5 shows simulation results of the offloading algorithms over the 9-state wireless channel with state transition probability matrix $\mathbb{P}_1$. Figure 5.5a shows the average energy consumption of the mobile device as the job deadline T_D changes. When T_D is small, offloading cannot meet the tight delay budget even when the channel is always at the best state. In this case, all the offloading algorithms decide to not offload, resulting in the same energy consumption as Local Execution. As T_D increases, the heuristic algorithms may decide to offload at some time slots but offloading most likely cannot be completed before t_L due to the small time budget. This triggers local execution, and results in overall energy consumption that is higher than Local Execution. For a certain range of T_D, the energy consumption may increase with T_D and then decrease. This is because as T_D increases, the offloading algorithms all attempt to upload at more time slots, while T_D is still insufficient to allow offloading to be completed in time. As T_D further increases, offloading may be completed before T_D, which reduces the concurrent local execution energy consumption; and as T_D further increases, offloading is more likely completed before t_L, in which case, concurrent local execution is not needed, and the average energy consumption of the mobile device using the offloading algorithms further decreases with T_D.

By comparing the different offloading algorithms, we find that in general, using "Exponentiation" (i.e., the *Exp* algorithms) helps reduce the average energy consumption, compared with using "Equilibrium" computations (i.e., the *Equ* algoritms) only. Using the "Generalization" approach helps reduce the average energy consumption; and using "Scheduling" can further reduce the average energy consumption, compared with using "Generalization". Overall, the WF-Equ-Sch and WF-Exp-Sch algorithms achieve the lowest average energy consumption among all the offloading algorithms.

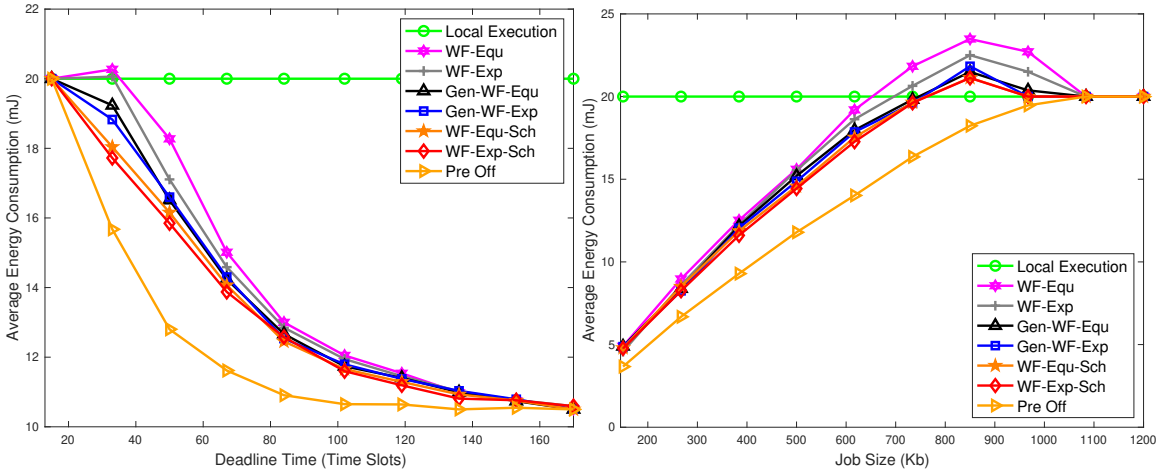
Comparing with the offline bound, the average energy consumption of using the heuristic offloading algorithms is the same as the bound when T_D is very small as all the algorithms decide to not offload. As T_D increases, the gap between the offline bound and the heuristic offloading algorithms increases first, then decreases. When T_D is sufficiently large, the heuristic offloading algorithms may decide to upload only when the channel is in the best state, and the average energy consumption of the offloading algorithms asymptotically approaches the offline bound.

Figure 5.5b shows the average energy consumption of the mobile device as the job size S_{up} changes. When S_{up} is small, the offloading algorithms may decide to upload only when the channel condition is sufficiently good, and offloading most likely can be completed before t_L. In this case, the average energy consumption of the offloading algorithms is close to the offline bound. As S_{up} increases, it becomes less likely that the task can be completed before t_L or even before T_D by offloading to the server. In this case, more concurrent local execution is needed that increases the energy consumption of the mobile device. When S_{up} is sufficiently large, all the offloading algorithms decide to not offload, resulting in the same energy consumption as in Local Execution.

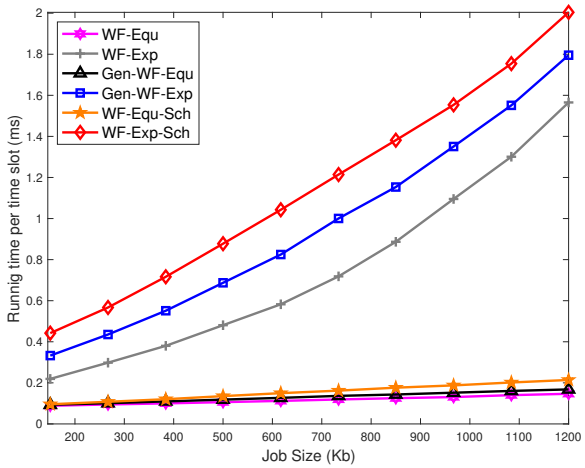
Figure 5.5c shows the running time versus job size for the offloading algorithms. As can be seen from this figure, the WF-Exp, WF-Exp-Sch and Gen-WF-Exp algorithms are much more time consuming than WF-Equ, WF-Equ-Sch and Gen-WF-Equ. This is mainly due to a more complicated process to find the "most efficient" set $\mathcal{F}$ that considers the number of steps needed from the current channel state to other states in order to finish uploading the task. As a result, the running time of the exponentiation-based algorithms increases quickly with the task size, because more channel transition steps should be checked. In contrast, the equilibrium-based algorithms are much less sensitive to task size increase

in terms of running time. Although the longer running time of the exponentiation-based algorithms makes them less practical in online situations, they do make more accurate offloading decisions than the equilibrium-based algorithms. Meanwhile we also notice that, the equilibrium-based algorithms (e.g., WF-Equ-Sch) achieve average energy consumption very close to the corresponding exponentiation-based algorithms (e.g., WF-Exp-Sch). Furthermore, doing generalization and scheduling increases the running time, and the running time of WF-Equ-Sch is slightly larger than that of Gen-WF-Equ, which is slightly larger than that of WF-Equ, although the running times of all three equilibrium-based algorithms are very close to each other.

Figure 5.6 shows simulation results of the offloading algorithms over the 9-state wireless channel with state transition probability matrix $\mathbb{P}_2$, which represents a non-uniform channel. Comparing Figures 5.6a and 5.5a we find that the non-uniform channel results in lower average energy consumption than the uniform channel when T_D is small; and as T_D increases, the curves in Figure 5.6a drop and approach the offline bound much faster than in Figure 5.5a. The non-uniform property of the channel helps the online algorithms make better offloading decisions and save energy consumption, compared to the uniform channel. For this reason, the performance difference between different offloading algorithms is much smaller than in the uniform-channel case. Similar observations can be obtained by comparing Figures 5.6b and 5.5b, from which we can see that the curves in Figure 5.6b rise slower with S_{up} than in Figure 5.5b. All the offloading algorithms have almost the same energy consumption performance except for a small range of S_{up} where the average energy consumption is above the Local Execution energy. Figure 5.6c further shows that the running time of WF-Exp, WF-Exp-Sch and Gen-WF-Exp is much more than WF-Equ, WF-Equ-Sch and Gen-WF-Equ, which is consistent with Figure 5.5c, and the non-uniform



(a) S_{up}=500 kb (b) T_D=60 ms

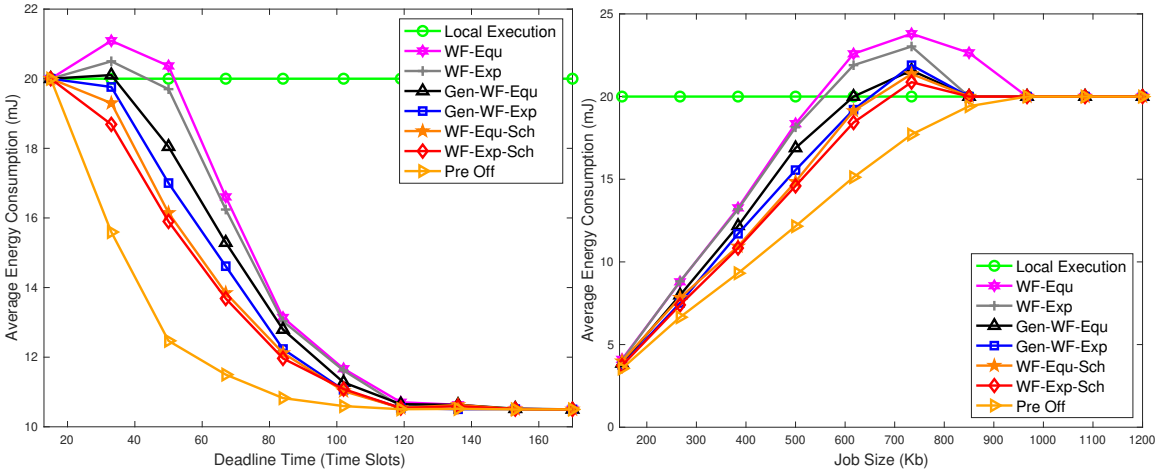


(c) T_D=60 ms

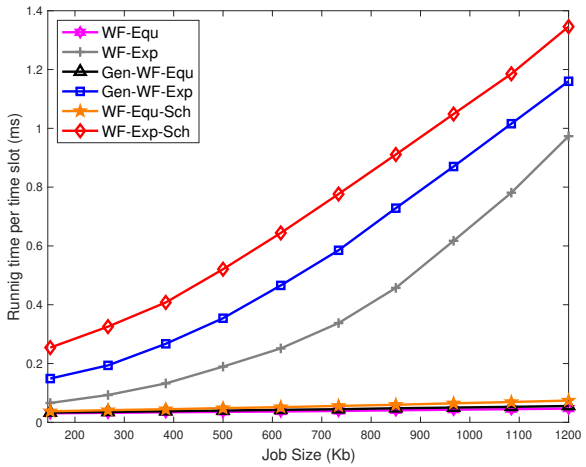
Figure 5.6: 9-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_2$

property of the channel does not affect the running time of the algorithms in an obvious way.

Figure 5.7 shows simulation results of the offloading algorithms over the 5-state wireless channel with state transition probability matrix $\mathbb{P}_3$, which represents a uniform channel. Comparing Figures 5.7a and 5.5a we find that the 5-state channel results in approximately the same average energy consumption as the 9-state uniform channel when T_D is small;



(a) S_{up}=500 kb (b) T_D=60 ms



(c) T_D=60 ms

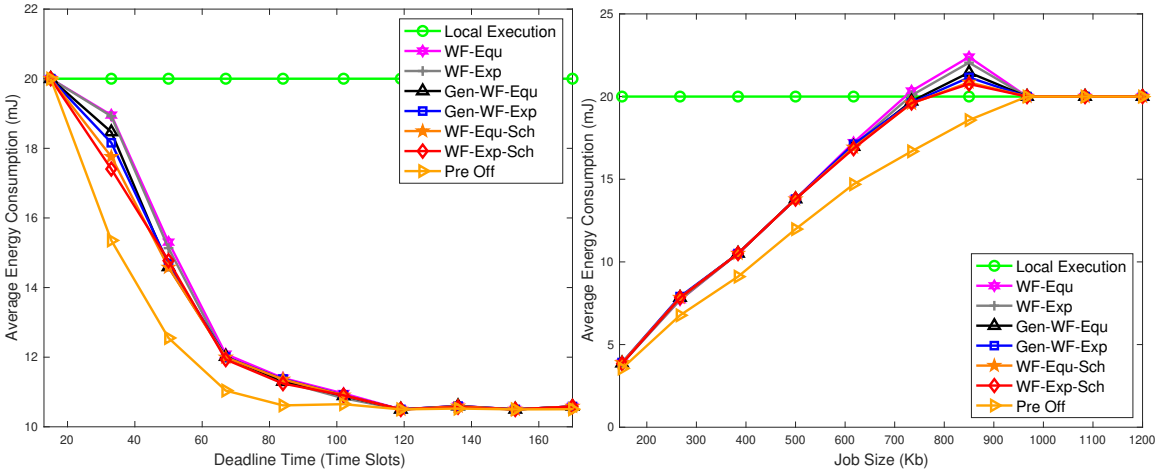
Figure 5.7: 5-states uniform distribution: $\mathbb{P} = \mathbb{P}_3$

and as T_D increases, the curves in Figure 5.7a drop and approach the offline bound much faster than in Figure 5.5a. The smaller number of channel states makes it easier for the online algorithms to make better offloading decisions that helps improve the energy consumption performance. Similar observations can be obtained by comparing Figures 5.7b and 5.5b, which shows that the average energy in Figure 5.7b increases slightly faster with

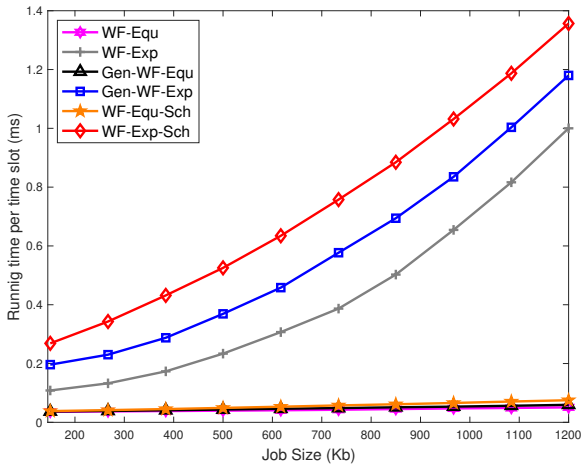
S_{up} than in Figure 5.5b. Figure 5.7c further shows that the running time of the equilibrium-based algorithms (i.e., WF-Equ, WF-Equ-Sch and Gen-WF-Equ) is much less than that of the exponentiation-based ones (i.e., WF-Exp, WF-Exp-Sch and Gen-WF-Exp) and is not significanatly affected by the task size increase. In addition, by comparing Figures 5.5c and 5.7c, we find that the running time of the algorithms in the 9-state uniform channel is higher than that in the 5-state channel case for all the algorithms, since more calculations are needed for the channel with more states.

Figure 5.8 shows simulation results of the offloading algorithms over the 5-state wireless channel with state transition probability matrix $\mathbb{P}_4$, which represents a non-uniform channel. Comparing with Figure 5.7, in terms of energy consumption, there is only a slight difference between the different offloading algorithms, and the performance of these heuristic offloading algorithms is very close to the offline optimum. The running time of the algorithms in this case is not obviously different from that in the 5-state uniform channel case.

In summary, among all the heuristic algorithms, WF-Exp-Sch achieves the lowest mobile device energy consumption and WF-Equ consumes the shortest running time, while WF-Equ-Sch is the best choice because its energy consumption is very close to WF-Exp-Sch and its running time is almost the same as WF-Equ. The difference between the heuristic algorithms is relatively small for very large size tasks with tight deadline or very small size tasks with loose deadline, because the decision is most likely to always execute the task locally (for the former) or always offload (for the latter). In terms of running time (complexity), the exponentiation-based algorithms are sensitive to the number of channel states and the task size, while the equilibrium-based algorithms are not affected as much by these parameters. In terms of energy consumption of the mobile device, the difference among



(a) S_{up}=500 kb (b) T_D=60 ms



(c) T_D=60 ms

Figure 5.8: 5-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_4$

the heuristic algorithms is more obvious in the uniform channel than in the non-uniform channel case.

All the above results are generated based on the parameter setting that has the cloud server CPU processing speed 10 times of the local CPU speed. Given the local processing speed, if the processing speed at the cloud server is higher, the probability that offloading can meet the delay constraint of the task is higher, and more energy consumption of the

mobile device may be saved by offloading. However, this benefit is eventually limited by the quality of the wireless channel, i.e., the amount of time and energy needed for uploading/downloading the task.

As discussed earlier, the computational complexity of online energy-optimal algorithms introduced throughout this thesis, especially OPO, is restrictive, and therefore, they may not be practical in real-world applications. In the next chapter, some novel approximation methods are introduced to address the issue of algorithmic efficiency.

Chapter 6

Approximate Solutions

6.1 Introduction

In this chapter, we consider the problem of algorithmic efficiency in mobile computation offloading while CLE is used to enforce hard task deadlines. Although online algorithms that are proposed for different job offloading scenarios (i.e., 1-Part, K-Part and preemptive offloading) in the previous chapters are proved to be energy-optimal, their high computational complexity prohibits their application in practice. We introduce ways of mitigating this complexity using three different mechanisms: Markovian Compression (MC), Time Compression (TC), and Preemption Using Continuous Offloading (Preemption-CO). The new methods are based on a time-dilated absorbing Markov chain (TDAMC) that was used to define the optimal offloading decisions in previous chapters. The TDAMC size dictates the time complexity of online offloading algorithms. In MC, the TDAMC is reduced by aggregating its states using a novel method based on the notion of geometric similarity. In TC, the TDAMC is traversed in aggregated time steps so that algorithm computation is reduced. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every

time slot is based on calculations performed on the (much smaller) 1-Part TDAMC. These methods are used (alone or in combination) to construct practical offloading algorithms, whose running times are greatly reduced, without suffering high performance degradation. In order to demonstrate this point, several comparisons between energy performances and running times of all possible combinations of the three methods are made through simulation.

The main contributions of this chapter are summarized as follows:

- While CLE guarantees the satisfaction of hard deadline constraints for mobile computational offloading, the running times of its online implementation in both the preemptive and non-preemptive settings may be prohibitive. We introduce three computationally efficient approximation methods (Markovian Compression (MC), Time Compression (TC), and Preemption Using Continuous Offloading (Preemption-CO)), which can be used as building blocks for the development of efficient CLE algorithms. The development presented applies to the general wireless Markovian channel case.

- We develop efficient CLE algorithms for both the non-preemptive (1-Part) and the preemptive setting, by combining these approximation methods. More specifically, algorithms 1-Part-MC, 1-Part-TC, 1-Part-MC-TC are presented for 1-Part offloading, and algorithms Preemption-CO-MC, Preemption-CO-TC, Preemption-CO-MC-TC for Preemptive offloading.

- Performance evaluation results are provided that show the running time and energy performance tradeoffs for the proposed algorithms. According to these results, by applying the approximation methods above, the running times of the algorithms are significantly reduced without suffering significant performance loss.

The rest of this chapter is organized as follows: Section 6.2, provides a brief review of the system modelling assumptions for 1-Part and preemptive offloading (from Chapters 3 and 5), which guarantee the satisfaction of hard job execution deadline constraints by using concurrent local execution (CLE). Following that, Section 6.3 describes the problem formulation and gives a summary of its optimal solution. Section 6.4 introduces new techniques used to approximate the calculations performed by the optimal online algorithms. In Section 6.5, simulation results are provided that compare the tradeoff between energy-saving and computing performance for different combinations of approximation methods.

6.2 System Model

In this chapter, we have introduced three approximation techniques that can be applied (alone or in combination) on online optimal algorithms proposed for 1-Part and preemptive offloading to reduce their computational complexity. Therefore the system model that is considered in this chapter is the same as that of Chapters 3 and 5 for 1-Part and preemptive offloading, respectively.

6.3 Problem Formulation and Optimal Solution

We are interested in developing online algorithms which solve the *CLE offloading problem*, i.e., given the CLE setting, decide whether to upload part of the task or not at every time slot, so that the expected task energy consumption is minimized. Note that the decision to upload is made once (if at all) for 1-Part offloading, while it is made repeatedly (if at all) for Preemptive offloading.

As discussed earlier, in order to develop optimal algorithms for the CLE offloading

problem, we incorporated both offloading and time in a new Markov Chain, called *a time-dilated absorbing Markov chain ($TDAMC$)*, to model the task uploading evolution, when it starts at the current time slot t and is done contiguously (for 1-Part offloading), or preemptively (for Preemptive offloading). Recall that we assume prior knowledge of the channel Markovian states, and the transition probabilities P_{ij} for all states i, j. An example of a $TDAMC$ for a 2-state (i.e., Gilbert-Elliot) channel and 1-Part offloading can be seen at the top of Figure 6.1: The channel goes through two states G, B (i.e., Good or Bad channel conditions), with bit-rates B_{max}, B_{min}, respectively, with transition probabilities $P_{GG}, P_{GB}, P_{BG}, P_{BB}$. For a general Markovian channel, the $TDAMC(t)$ is constructed by (i) unrolling the evolution of the stochastic channel Markov Chain from the current time slot t, up to absorbing states indicating the time of task execution completion, and (ii) uploading at every state according to its bit-rate (in case of 1-Part offloading), or branching according to whether the decision for uploading at the current time slot is made or not. In Figure 6.1, the $TDAMC(1)$ root state $G_{1,1}$ indicates that the current $t = 1$ channel state is G, and the task is being uploaded with bit-rate B_{max}. Then ($t = 2$) the channel either transitions to B with probability P_{GB}, or remains in G with probability P_{GG}, and the $TDAMC(1)$ transitions to $B_{2,1}$ or $G_{2,1}$, respectively. The number of branches out of an initial state for an arbitrary Markovian channel is equal to the number of transitions out of that state in the Markov chain. In the two-state example shown, the evolution of the $TDAMC(1)$ continues in the same fashion, with its states layered as $G_{t,l}, B_{t,m}$ for time slot t, and $l, m = 1, 2, \ldots$ In general, $TDAMC(t)$ is a layered tree-like Markov Chain, starting with the current state as the root at time slot t, and going through layers corresponding to time slots $t + 1, t + 2, \ldots$ up to time slot $t_D + 1$, or earlier absorbing states if offloading was finished earlier than t_D.

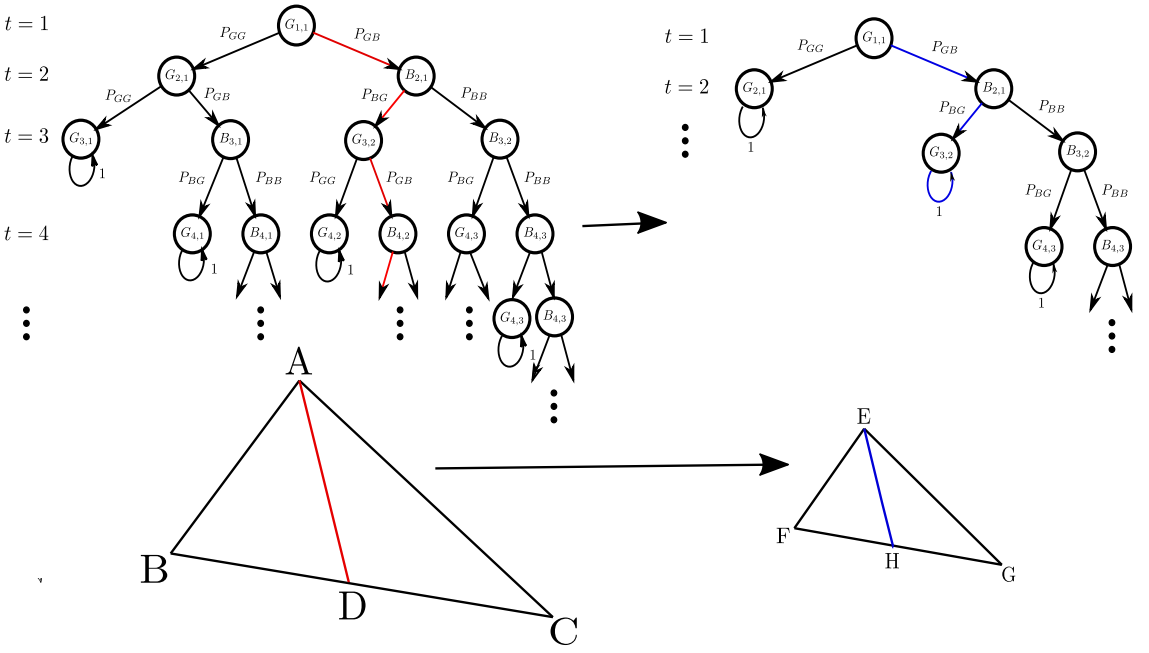


Figure 6.1: Markovian Compression of the original TDAMC (left) to a smaller TDAMC^{approx} (right).

For the Preemption offloading case, the initial ($t = 1$) $TDAMC(1)$ is enhanced to be a Markov decision process as follows: For every time $1 \leq t \leq t_D + 1$, we define a set of states (X_t, S_t), where X_t is a channel state and $0 \leq S_t \leq S$ is the number of task bits that are uploaded up to t. Let B_{X_t} be the bit rate of channel state X_t. The $TDAMC(1)$ states are again arranged in layers for $t = 1, 2, \ldots, t_D + 1$. The set of actions contains two actions, a_0, a_1, corresponding to not uploading, or uploading, respectively. A state (X_t, S_t) branches to a_0 and a_1; then a_0 branches to states (X_{t+1}, S_t) with probabilities $P_{X_t, X_{t+1}}$, and a_1 branches to states $(X_{t+1}, S_t + \min\{B_{X_t}, S - S_t\})$ with the same probabilities. States of the form (X_t, S) branch only to action a_0 (no uploading). At layer $t = 1$ there is only one state $(X_1, 0)$, where X_1 is the initial channel state, while the states at layer $t_D + 1$ are absorbing. $TDAMC(t)$ is similarly constructed for any current time slot t.

Using classical Markovian stopping theory (cf. Peskir and Shiryaev (2006)), as we

proved in Chapter 3, the following simple online algorithm solves the CLE offloading problem *optimally* for 1-Part offloading (and the same can be shown for Preemptive offloading[1]): At every time slot t, compare the expected energy cost of starting uploading at t, to the expected energy cost if we wait to check again at $t+1$, by using Dynamic Programming (DP) on $TDAMC(t)$ and $TDAMC(t+1)$ respectively; if the former is less than the latter, then upload at t.

6.4 Approximate Solutions

The DP running time of the optimal algorithm of Section 6.3 depends on the size of $TDAMC(t)$, since the DP recursion subproblems correspond to $TDAMC(t)$ states. As a result, the optimal algorithm becomes impractical even for simple channel models, such as the 2-state Gilbert-Elliot model; this phenomenon is even more pronounced in the Preemptive offloading case, since the $TDAMC(t)$ states are much more numerous, because they have to also record the remaining task bits to be uploaded.

More specifically, we identify three sources of inefficiency when implementing the optimal offloading algorithm:

A. The size of the $TDAMC$ as a Markov Chain (i.e., number of states and transitions).

B. Running the optimal algorithm at *every* time slot, imposing a large computational load on the mobile device.

C. In Preemptive offloading, several different uploading time slots must be picked (instead of a single uploading time slot for 1-Part offloading). Hence, every state of

[1]The Dynamic Programming for Preemptive offloading is more complicated due to the fact that $TDAMC$ states also record the remaining bits to be uploaded, but exactly the same arguments go through.

the $TDAMC$ has to record not only the channel state at a particular time, but the remaining task bits to be uploaded, in order to accurately capture the multiple uploading time slot combinations.

We address the forbiddingly high running time of the optimal algorithm by designing algorithms addressing each one of these factors. We will evaluate their performance, separately or combined, in Section 6.5.

6.4.1 Markovian Compression (MC)

The large size of the $TDAMC$ in the optimal algorithm, is the main reason for the high DP calculation of expected energy consumption. In effect, the DP traverses recursively all possible root-to-leaf tree-paths, and collects the energy spent on each path, weighted by the path probability, in order to compute the total mean energy consumption (the exact recursive process for 1-Part offloading is described in detail in Section 3.5, and it is similar for Preemptive offloading). If one is to replace the original $TDAMC$ with a smaller $TDAMC^{approx}$, then the latter must approximate well this energy computation, i.e., its (much fewer) paths must be of about the same expected mixture of bit-rates, as in the original $TDAMC$ paths. The key observation on how to do this, is motivated by a geometric analogy (see Figure 6.1).

We create a new channel Markov Chain model (which will generate $TDAMC^{approx}$) with the same number of states as the original Markov Chain that generated $TDAMC$. In order to determine the new state bit-rates and transition probabilities in $TDAMC^{approx}$, we sort the original $TDAMC$ paths from the shortest to the longest (see Figure 6.1, left). Recall that each path corresponds to the uploading of S bits along its states, so the left-most (i.e., shortest) path consists of only highest bit-rate states, and the right-most (i.e.,

longest) path consists of only lowest bit-rate states. Intuitively we would like $TDAMC$ and $TDAMC^{approx}$, seen as 'triangles', to be similar in the following sense: The energy consumption on shortest paths AB, EF should equal the ratio of energy consumption on longest paths AC, EG, while the corresponding number of states ratios should also be equal. Hence, in order for triangles (ABC) and (EFG) to be similar with a scaling factor of l_{MC}, we scale up all the original state bit rates by a factor of l_{MC}; these are the new bit-rates for the channel Markov Chain generating $TDAMC^{approx}$. Next, focusing again on paths AB, EF or AC, EG, we observe that the transitions in the new Markov Chain are in fact transitions in the original Markov Chain, transitioning from the last state of a group of l_{MC} states with bit-rates B_{min} or B_{max}, respectively, to the first state of the next group. Hence, we set the transition probabilities of the new Markov Chain to be *equal* to the original transition probabilities, so that a path AD's probability ends up away from AB's probability by about the same amount that its similar path EH's probability ends up away from EF's probability. To summarize, if $\mathbb{P}, \mathbb{B}$ are the original transition matrix and state bit-rate vector, then $\mathbb{P}^{approx} = \mathbb{P}, \mathbb{B}^{approx} = l_{MC} \cdot \mathbb{B}$ for the new approximate channel Markov Chain.

After $TDAMC^{approx}$ has been defined, our algorithm runs the optimal online algorithm on $TDAMC^{approx}$ of Section 6.3. For example, in the case of 1-Part offloading, the expected cost of offloading at time slot t with the channel state $x = X_t$ can be approximated by $g_t^{approx}(X_t)$, defined as follows:

$$g_t^{approx}(x) = E_{off}^{approx}(t, x) + E_L^{approx}(t, x). \tag{6.4.1}$$

In Equation (6.4.1), $E_{off}^{approx}(t, x)$ and $E_L^{approx}(t, x)$ are approximations of the offloading and local expected energy cost when the offloading starts at time slot t, and they can be

$$E_{off}^{approx}(t,x)=\begin{cases} l_m\cdot E_{tr}\sum\limits_{T_{off}=\frac{S}{l_m\cdot\mathbb{B}(m)}}^{\frac{S}{l_m\cdot\mathbb{B}(1)}} P_t(T_{off},x)T_{off}, & 1\le t<t_D-\frac{S}{\mathbb{B}(1)}+1 \\ l_m\cdot E_{tr}\left(\sum\limits_{T_{off}=\frac{S}{l_m\cdot\mathbb{B}(m)}}^{\frac{t_D-t}{l_m}} P_t(T_{off},x)T_{off}\right. & \\ \left.\quad+\sum\limits_{T_{off}=\frac{t_D-t+1}{l_m}}^{\frac{S}{l_m\cdot\mathbb{B}(1)}} P_t(T_{off},x)(\frac{t_D-t+1}{l_m})\right), & t_D-\frac{S}{\mathbb{B}(1)}+1\le t\le t_D \end{cases} \tag{6.4.2}$$

$$E_{L}^{approx}(t,x)=\begin{cases} l_m\cdot\left(\sum\limits_{T_{off}=\frac{t_L-t+1}{l_m}}^{\frac{S}{l\cdot\mathbb{B}(1)}} P_t(T_{off},x)\left(\frac{\min\{\frac{t_D+1}{l_m},\frac{t}{l_m}+T_{off}\}-\frac{t_L}{l_m}}{T_L}E_L\right)\right), & 1\le t<t_L \\ \frac{t-t_L}{T_L}E_L+l_m\cdot\sum\limits_{T_{off}=\frac{S}{l_m\cdot\mathbb{B}(m)}}^{\frac{S}{l_m\cdot\mathbb{B}(1)}} P_t(T_{off},x)\left(\frac{\min\{\frac{t_D-t+1}{l_m},T_{off}\}}{T_L}E_L\right), & t_L\le t\le t_D \end{cases} \tag{6.4.3}$$

obtained by Equations (6.4.2) and (6.4.3), respectively. The compressed $TDAMC^{approx}$ is used in order to calculate the probability $P_t(T_{off},x)$ of the offloading taking exactly T_{off} time slots, when it starts at time t and $TDAMC^{approx}$ is in state x. In Chapter 3 we showed that the simple online strategy of Section 6.3 (i.e., deciding whether to offload or wait for one more time slot), coincides with the optimal solution of the offline optimization problem

$$t_{off}=\arg\min_{1\le t\le t_D+1} E[g_t^{approx}(X_t)|X_1], \tag{6.4.4}$$

which calculates the optimal offloading time t_{off}, when the job deadline is t_D and the initial state of $TDAMC^{approx}$ is X_1 (if $t_{off}=t_D+1$, then it is best not to offload at all).

By solving the problem optimally for $TDAMC^{approx}$, an approximate solution for the

original problem is obtained.

6.4.2 Time Compression (TC)

In order to avoid running the optimal algorithm at every time slot, our algorithm simply runs it every l_{TC} time slots, and compares the expected energy consumption of $TDAMC(t)$ and $TDAMC(t + l_{TC})$ (instead of $TDAMC(t)$ and $TDAMC(t + 1)$). Note that in this case, if the decision is made to upload, the uploading starts at time $t + l_{TC}$ with the channel in state $X_{t+l_{TC}}$, while the DP computations were done assuming that the current channel state is X_t. This is an additional source of approximation error for the algorithm.

6.4.3 Preemption Continuous Estimate (Preemption-CO)

In Preemptive offloading, the optimal algorithm DP is run over a $TDAMC$ whose states record also the remaining task bits to be uploaded, in order to account for all possible combinations of uploading time slots, when it is estimating the expected energy consumption. In our algorithm, we propose that the *estimate* of expected energy consumption by the DP be done not on the *preemptive* $TDAMC$, but on the *1-Part* one, instead. Note that this algorithm should not be confused with the 1-Part optimal offloading algorithm; if the algorithm decides to upload at t, it uploads B_{X_t} bits (where B_{X_t} is the bit-rate at the channel state X_t), but continues to check whether to upload some of the remaining bits in the next time slot, etc., as opposed to 1-Part offloading where once uploading is initiated, it continues automatically until it finishes.

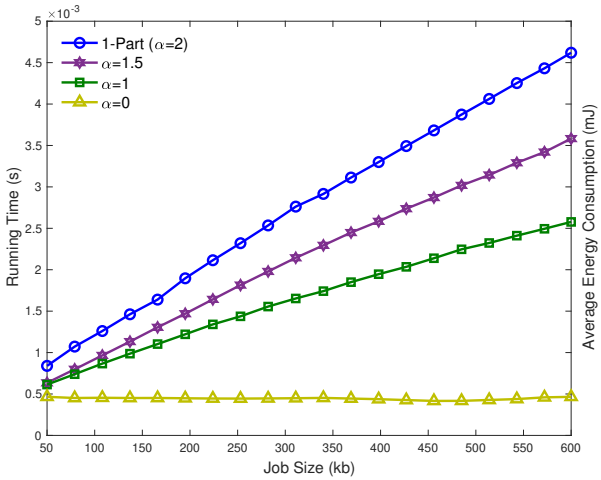
6.5 Simulation Results

In this section, computer simulation is used to evaluate the performance of the proposed approximation methods. A Gilbert-Elliot channel model is used for the simulations. This model is widely used to characterize the effects of burst noise in wireless channels, i.e., where the channel can abruptly transition between good to bad conditions (Zhang *et al.* (2014); Zhang *et al.* (2013a); Blazek and Mecklenbräuker (2018); Botta and Pescapé (2015); Elliott (1963); Zafer and Modiano (2007); Johnston and Krishnamurthy (2006)). This type of channel is a difficult one for computation offloading algorithms to deal with, compared to those where there is much more correlation in the channel quality as the offloading progresses. It is this harsh channel environment that we subject the algorithms to, for our comparisons. The channel is modelled as a two-state Markov chain with a "Good" (G) state having bit rate B_g and a "Bad" (B) state having bit rate B_b, and $B_g > B_b$. We also assume that the transmit power control is used on the downlink, and therefore, T_{down} as well as T_{exec} are deterministic. Their effects can therefore be accounted for by modifying the remote offload end-times used in the analysis. Simulations are conducted by applying the proposed approximation methods to the 1-Part Offloading and Preemption Offloading algorithms. For comparison, we also simulate Local Execution, in which the entire task is executed locally on the mobile device without doing any offloading.

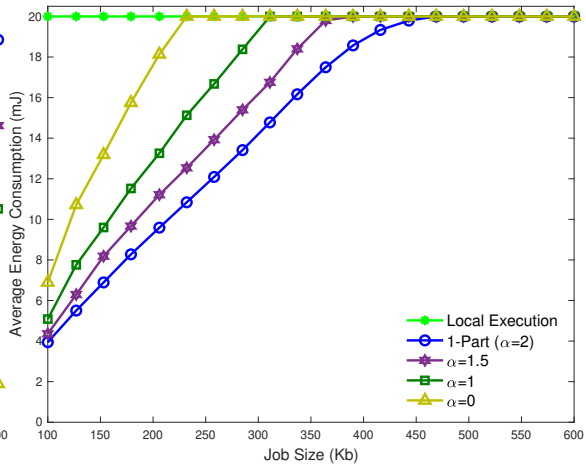
In all offloading algorithms, if $\frac{S}{B_g} + T_{rest} > t_D$, then the offloading cannot be completed before the job deadline even under the best channel conditions, in which case the job is executed locally without offloading. Furthermore, according to concurrent local execution, to ensure that the job completion deadline is always satisfied, local execution starts at time slot t_L if offloading (includes uploading to, remote execution at, and downloading from the server) is not completed at time slot $t_L - 1$. The default parameters used in the simulations

are given as follows. All tasks are released at time slot zero. Each time slot is 1 ms. The transmit and receive power is 1 W and 0.5 W, respectively, which means that the transmit and receive energy during each time slot is $E_{tr} = 1$mJ and $E_{rc} = 0.5$mJ, respectively. 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, see (Nir *et al.* (2014); Huang *et al.* (2012)). The computation load of each task is $D = 10$M CPU cycles, and the local execution time is $T_L = D/f_l = 10$ time slots. The local energy consumption $E_L = v_l D = 20$mJ. We consider that the remote execution time is $T_{exec} = 1$ time slot, i.e., the remote processing speed is 10 times local processing. The download time T_{down} is assumed to be 1 time slot. The data transmission rates are $B_g = 50$ Mbps and $B_b = 12.5$ Mbps. The channel state transition probabilities are $P_{GG} = 0.3$ (from G to G) and $P_{BB} = 0.7$ (from B to B). The selection of these channel quality parameters is based on the results obtained in Chapter 3, which shows that the offloading decision is either almost always to offload (when the channel condition is sufficiently good) or almost always to not offload (when the channel condition is sufficiently bad). The channel-related parameters are chosen so that they represent a channel condition under which rigorous calculations are required to make the correct offloading decisions, in which case using more efficient algorithms for making the offloading decisions is important. In addition, since the proposed algorithms are not designed for specific applications, the job size and job completion deadline in the simulations are changed in order to cover a wide range of real-world scenarios. In the results below, each value of average energy consumption or running time is obtained by averaging 1000 random i.i.d. runs of the wireless channel.

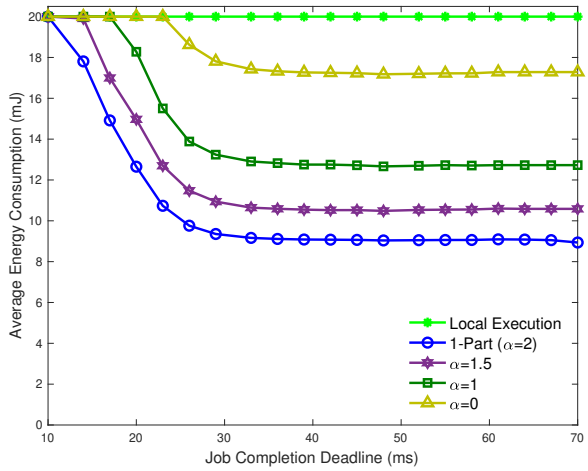
1-Part-MC: By applying the proposed MC method on the 1-Part Offloading algorithm, an approximate solution for the continuous offloading optimization problem can be found.



(a) t_D=60ms



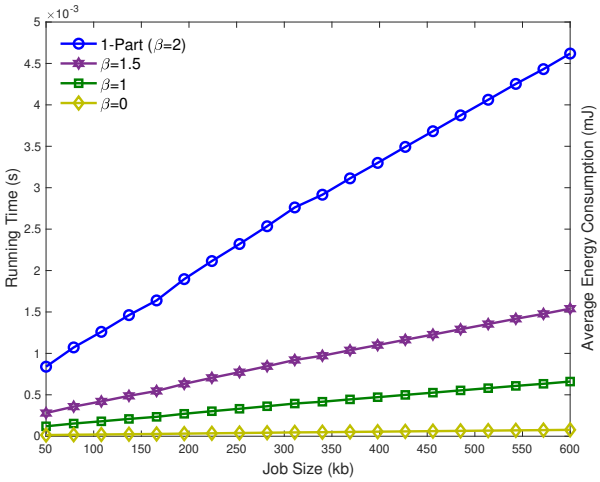
(b) t_D=60ms



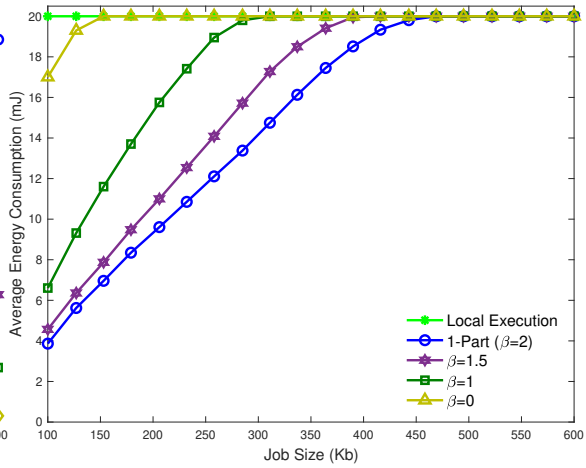
(c) $S = 200$kb

Figure 6.2: Approximation of 1-Part offloading using 1-Part-MC for different values of α

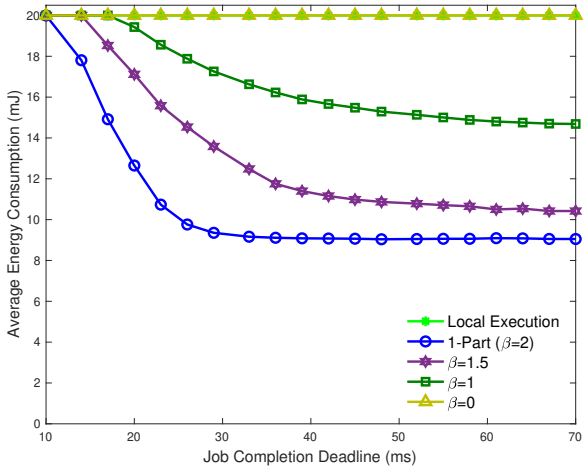
We call this algorithm 1-Part-MC. The compression factor is set as $l_{MC} = (\frac{S}{B_b})^{\frac{2-\alpha}{2}}$, where $\alpha \in [0, 2]$ with $\alpha = 0$ corresponding to the maximum compression and $\alpha = 2$ corresponding to no compression. Figure 6.2a compares the running time of 1-Part-MC versus task size S with different α values. The reported values for running time in this figure are the average amount of time needed by the mobile device to make the decisions to either wait or start offloading for all available time slots. Therefore, the average amount of time per



(a) t_D=60ms



(b) t_D=60ms



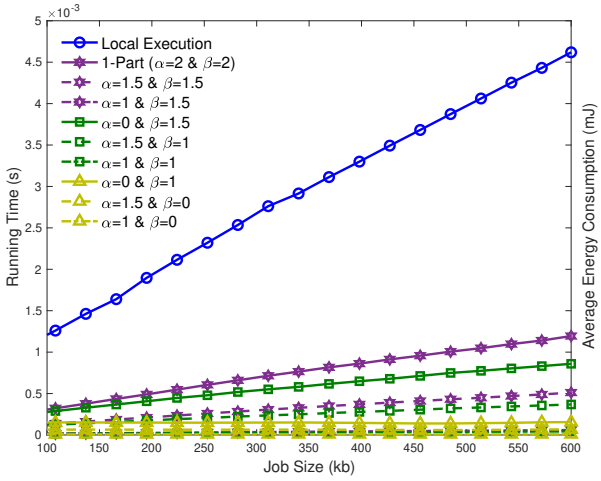
(c) $S = 200$kb

Figure 6.3: Approximation of 1-Part offloading using 1-Part-TC for different values of β

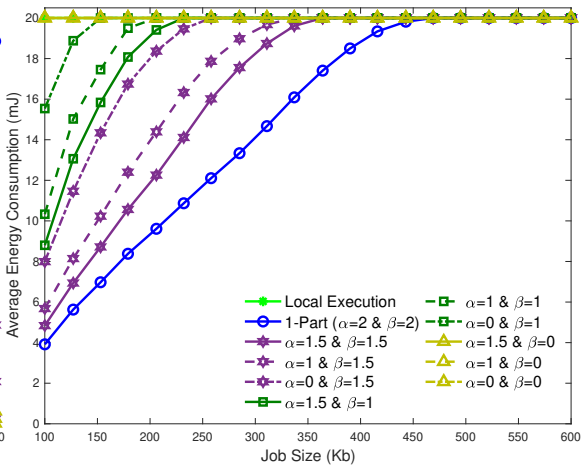
decision making at each time slot can be found by dividing these numbers by the number of the total available time slots (t_D). When $\alpha = 0$, the whole TDAMC is compressed into one node, and the algorithm can be run in a constant amount of time for any values of S. When $\alpha > 0$, the running time increases with both α and S, since the size of the approximated TDAMC increases with both.

Figures 6.2b and 6.2c, respectively, show the average energy consumption of the 1-Part-MC algorithm versus S and t_D with different α values. As α decreases (i.e., l_{MC} increases), more compression is done, resulting in less accurate offloading decisions that increase the average energy consumption of the mobile device. When α is small, the 1-Part-MC approaches the Local Execution faster as S increases (Figure 6.2b), and it requires larger t_D in order to achieve lower average energy consumption than the Local Execution (Figure 6.2c). In general, for each given value of α, when S is small or t_D is large, the delay constraint is less stringent, and it is more likely for offloading (without local execution) to meet the delay constraint due to a shorter channel uploading time. In this case, the energy consumption is less than that of Local Execution. On the other hand, when S is large or t_D is tight, jobs are most likely processed locally, and the average energy consumption of the offloading methods is close to or the same as Local Execution.

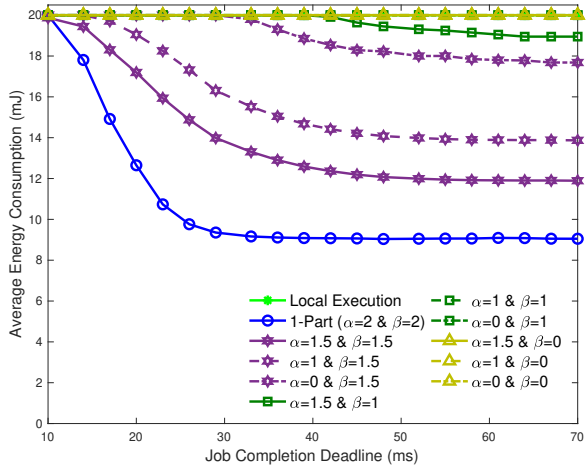
1-Part-TC: This is done by applying the TC method to 1-Part Offloading. The compression factor is $l_{TC} = {t_D}^{\frac{2-\beta}{2}}$, where $\beta \in [0, 2]$ with $\beta = 0$ corresponding to the maximum compression and $\beta = 2$ corresponding to no compression. Figure 6.3a shows the running time of the 1-Part-TC algorithm as S increases for different β values. By increasing l_{TC} (using smaller β) the running time of the algorithm can be reduced significantly. The compression factor is the largest when $\beta = 0$, in which case only one offloading decision is



(a) t_D=60ms



(b) t_D=60ms



(c) $S = 200$kb

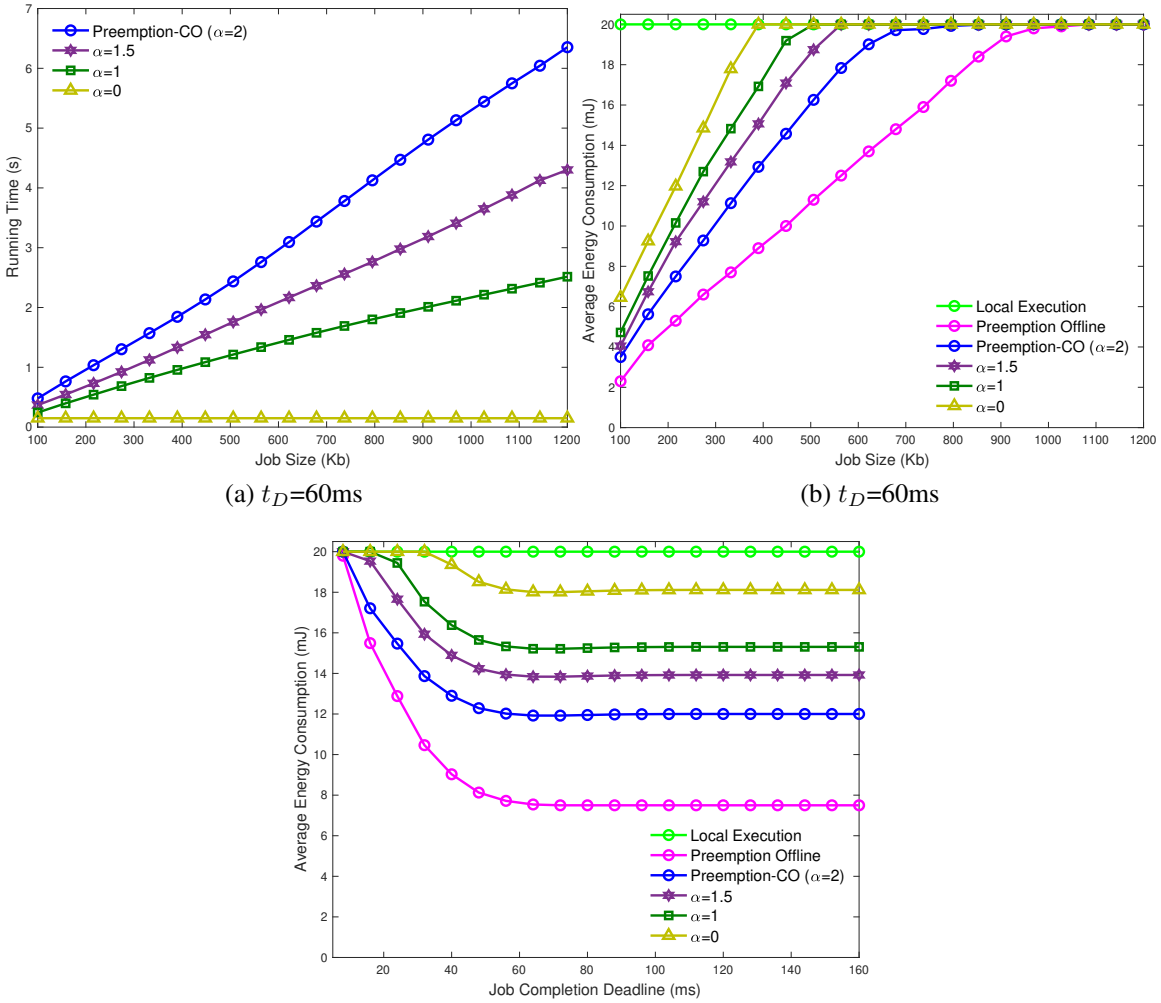
Figure 6.4: Approximation of 1-Part offloading using 1-Part-MC-TC for different combinations of α and β

made for the whole task at the release time, which means either the uploading starts immediately after the release time of the task or the task will be executed locally on the mobile device. In this case, the running time stays almost constant as S increases.

Figures 6.3b and 6.3c show the average energy consumption versus the task size S and deadline t_D, respectively. As the compression factor increases, the less accurate decisions result in higher energy consumption of the mobile device. When the compression factor is the largest ($\beta = 0$), the energy consumption of the mobile device is close to that of Local Execution. However, when $\beta = 1$ and 1.5, the compression causes a moderate increase in energy consumption but leads to a significant reduction in the running time.

1-Part-MC-TC: In this set of simulations, the MC and TC methods are applied simultaneously to approximate 1-Part Offloading. Figure 6.4a shows that the running time can be more significantly reduced by using both the compression methods, compared to 1-Part-MC and 1-Part-TC. However, by applying both methods at the same time, 1-Part-MC-TC suffers higher approximation errors and results in higher energy consumption as shown in Figures 6.4b and 6.4c.

Preemption-CO-MC: This is obtained by applying MC to the Preemption-CO 1-Part estimate (cf. Section 6.4.3). The MC method is used to approximate the optimal expected offloading cost at each time slot. Figure 6.5a shows the running time of Preemption-CO-MC as the task size changes for different compression factors, where α is defined the same as in 1-Part-MC. By increasing the compression factor the running time can be reduced considerably. Figures 6.5b and 6.5c show the average energy consumption of the mobile device versus the task size and deadline, respectively. By decreasing α (increasing compression factor), the approximation error becomes larger, and the energy consumption increases. As a reference, we also plot Preemption Offline, which finds the optimal

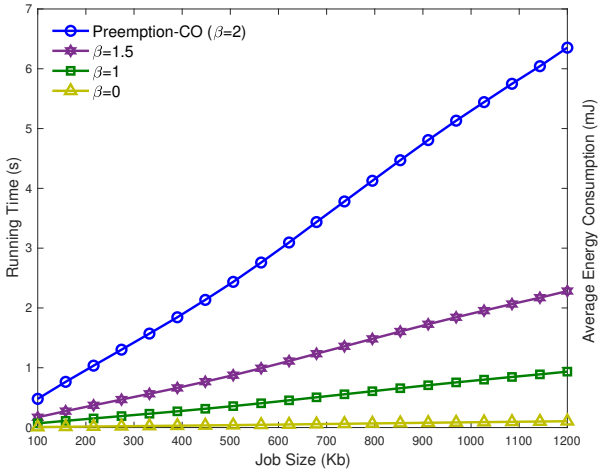


(a) t_D=60ms (b) t_D=60ms

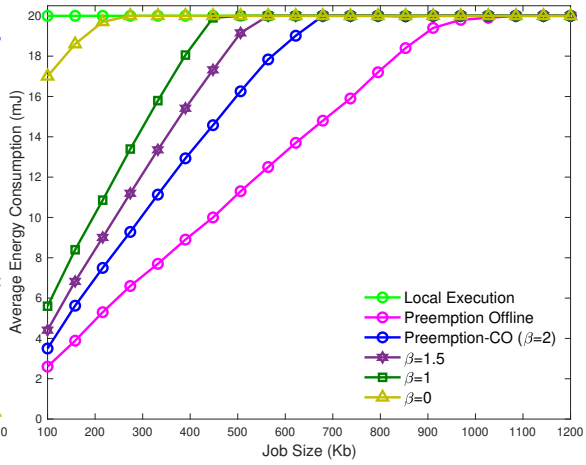
(c) $S = 350$ kb

Figure 6.5: Approximation of Preemption offloading using Preemption-CO-MC for different values of α

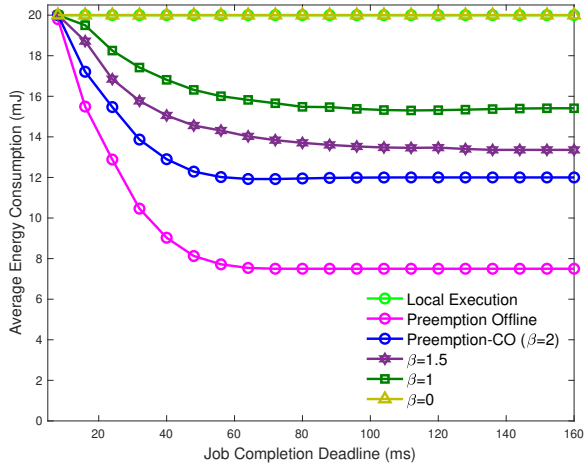
uploading times to offload the task preemptively by assuming complete knowledge of all future channel states. The gap between Preemption-CO-MC with $\alpha = 2$ (no compression) and Preemption Offline is due to the fact that the former is ignorant of future channel states.



(a) t_D=60ms



(b) t_D=60ms

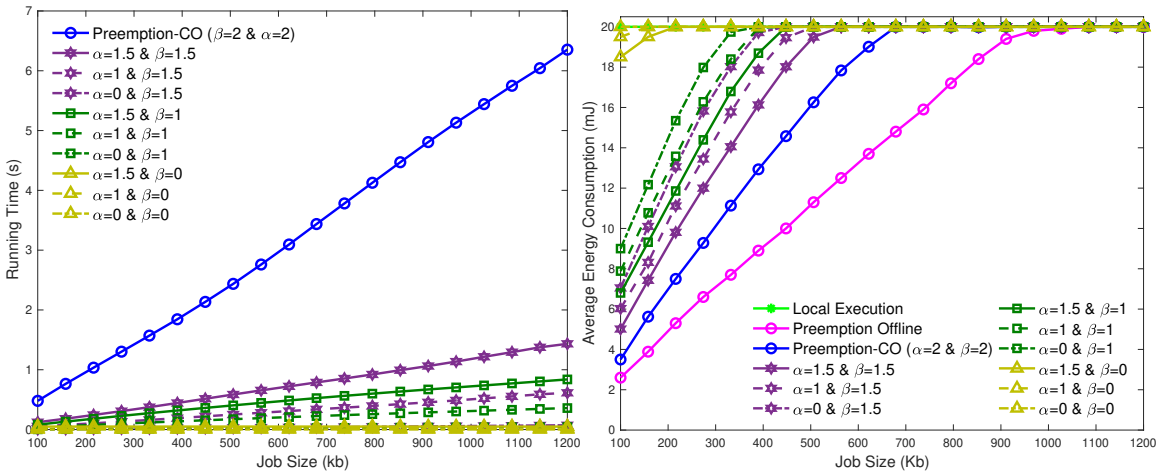


(c) $S = 350$ kb

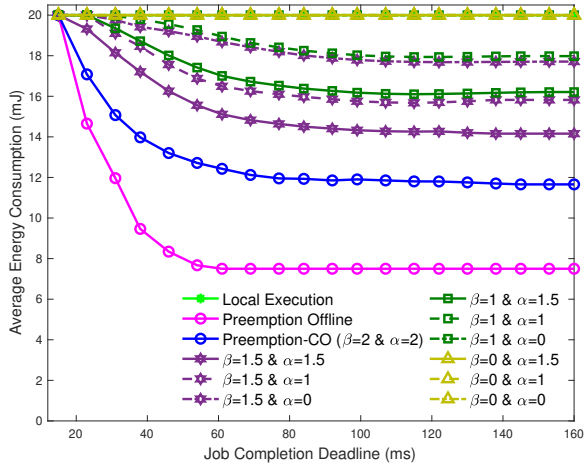
Figure 6.6: Approximation of Preemption offloading using Preemption-CO-TC for different combinations of β

Preemption-CO-TC: By applying TC approximation on Preemption-CO Offloading, we have Preemption-CO-TC. The compression factor β is defined as in 1-Part-TC. Figure 6.6a represents the running time of the Preemption-CO-TC algorithm as S increases for different values of β. As can be seen from this figure, increasing l_{TC} (using smaller β) decreases the running time of the algorithm considerably. Figures 6.6b and 6.6c show the average energy consumption versus the task size S and deadline t_D, respectively. Increasing the compression factor results in less accurate offloading decisions, which causes higher energy consumption on the mobile device. When the compression factor is the largest ($\beta = 0$), the mobile device's energy consumption is close to that of Local Execution. However, when $\beta = 1$ and 1.5, the compression causes a moderate increase in energy consumption but leads to a significant reduction in the running time.

Preemption-CO-MC-TC: All three approximation algorithms in Section 6.4 are applied simultaneously to implement Preemption Offloading. Figure 6.7a shows that more significant reductions are achieved in running time, compared to Preemption-CO-MC. Figure 6.7b shows that the average energy consumption approaches Local Execution energy faster as S increases, and Figure 6.7c shows that Preemption-CO-MC-TC requires a larger t_D to achieve lower average energy consumption than Local Execution.



(a) t_D=60ms (b) t_D=60ms



(c) $S = 350$ kb

Figure 6.7: Approximation of Preemption offloading using Preemption-CO-MC-TC for different combinations of α and β

Chapter 7

Conclusions and Future Work

In this work, we considered the problem of MCO over stochastic wireless transmission channels, where task execution times are subjected to hard deadline constraints. To ensure that hard job completion deadlines are met in the face of any random channel conditions, a CLE model was proposed, which allows for the overlapping of offloading and local execution, as opposed to the conventional computation offloading model where job execution is either local or remote. This mechanism ensures that deadlines are met even in worst-case situations, such as where the wireless channel suffers a complete outage during the offload.

The wireless communication channel was modeled as a Markov Chain, which transitions amongst states of different bit-rates in every time slot according to a transition probability matrix which is known to the mobile terminal. In this thesis, MCO was studied for various job offloading schemes.

In the first scenario, which was proposed in Chapter 3, we considered a continuous MCO, referred to as *1-Part Offloading*, which means once the offload starts, it occurs contiguously in one piece without interruption. An online provably energy-optimal algorithm, OnOpt, which uses CLE as a method of enforcing hard task execution time constraints, was

developed to find the optimal upload initiation time. This is done by first constructing a TDAMC from the underlying Markov channel description. The theory of optimal stopping time for Markov chains and DP results were then used with the TDAMC to show the optimality of OnOpt in terms of the expected energy consumption of the mobile device. This resulted in a simple test that can be performed to determine if the current time is best for initiating a computation offload. Then a Gilbert-Elliott channel model was considered and closed-form results were derived that were used to find optimal offload initiation times. The job completion time probabilities were computed recursively, which led to a large reduction in the computational complexity.

The performance of the proposed algorithm was 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 algorithm can significantly improve mobile device energy consumption compared to the other approaches while guaranteeing hard task execution deadlines.

Mobile device energy can sometimes be further improved by splitting the task upload into multiple parts. For this purpose, in the second scenario, which was introduced in Chapter 4, multi-part mobile computation offloading was assumed. In this offloading scheme, the computational job was partitioned into multiple (K) parts of known bit-sizes, and each part was uploaded separately. We called this method *K-Part offloading*. By segmenting a task upload into multiple parts, rather than doing a conventional contiguous task upload, the mobile device has a higher chance for adaption to the varying wireless channel conditions at the end of each upload part. This may decrease the energy consumption of the mobile

device in exchange for higher computational complexity and running time. An online mobile computation offloading algorithm, *MultiOpt*, that was shown to be energy-optimal, and satisfies hard deadline constraints, was proposed for this problem.

Since the computational complexity of MultiOpt can be significant, simpler and more computationally efficient heuristics, that always satisfy hard task execution deadlines, may be used. In Chapter 4, we used two such heuristics, the Immediate Offloading, and Multi Threshold algorithms. The mobile energy use of MultiOpt was compared to these heuristics, as well as to local execution without offloading. Simulation results showed that MultiOpt performs significantly better when compared to the proposed heuristics, as well as when K increases.

The third scenario, which was introduced in Chapter 5, considered preemptive mobile computation offloading, referred to as *preemptive offloading*, where a piece of job can be uploaded at each time slot. In this case, at the start of each time slot, a decision is made to either continue offloading or to temporarily interrupt the offload. Preemptive offloading can be seen as a generalization of K-Part offloading where the number and the size of upload parts are not known in advance. An energy optimal preemption offloading (*OPO*) algorithm, which also respects hard deadline constraints, is introduced for this problem. The computational complexity of OPO is prohibitive, even for simple Markovian channels, and, therefore, we introduced three computationally efficient techniques: Water-Filling, Water-Filling with Scheduling, and Generalized Water-Filling. For each, two variations were considered. The first (Equ) uses the equilibrium channel state probabilities to determine its offloading decisions, and the second (Exp) uses Markovian transition matrix exponentiation. The six resulting algorithms have a wide variety of energy performance

and computational complexity. The performance of the proposed algorithms was compared on Markovian channels with different characteristics, in order to show the tradeoffs between complexity and mobile energy saving performance.

The proposed online algorithms, which were proved to be energy optimal, were developed using the theory of optimal stopping for Markov decision processes. In general, they make the offloading decision at each time slot by constructing a TDAMC from the given Markov channel model and applying DP. The large size of TDAMC and running DP at each time slot increases the computational complexity of these algorithms, especially in the case of preemptive offloading where we need to consider all future optimal offloading time slots in our DP calculations. To increase the efficiency of these algorithms and make them practical in online mobile implementations, three approximation methods were introduced in Chapter 6 to reduce this complexity, namely, Markovian Compression (MC), Time Compression (TC) and Preemption Using Continuous Offloading (Preemption-CO). MC reduces the state space of the offloading Markovian process, by aggregating TDAMC's states using a novel notion of geometric similarity. TC reduces the algorithm computation by traversing the TDAMC in aggregated time steps. In Preemption-CO, while a task is offloaded preemptively, the offloading decision at every time slot is based on non-preemptive calculations. These methods were used (alone or in combination) to construct efficient offloading algorithms, whose running times are significantly reduced, without suffering considerable performance degradation.

Future directions: There are several possible directions for future extensions of our work. The high computational complexity incurred by the exact DP solutions leads us to the development of approximate heuristics; rollout techniques (cf. Bertsekas (2005), and the references therein) can be studied as an efficient mechanism to tradeoff complexity and

optimality. Reinforcement learning, which has already been used for designing offloading policies (e.g., (Mao *et al.* (2016a); Li *et al.* (2018)), can be used to effectively learn the parameters of the system (e.g., channel transition probabilities).

Bibliography

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A., and Buyya, R. (2013). Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *IEEE Communications Surveys & Tutorials*, **16**(1), 337–368.

Akherfi, K., Gerndt, M., and Harroud, H. (2018). Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, **14**(1), 1–16.

Al-Ali, A., Aji, Y., Othman, H., and Fakhreddin, F. (2005). Wireless smart sensors networks overview. In *Proceedings of IEEE 2nd IFIP International Conference on Wireless and Optical Communications Networks, 2005. WOCN 2005.*, pages 536–540. IEEE.

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

Bangui, H., Rakrak, S., and Raghay, S. (2015). External sources for mobile computing: The state-of-the-art, challenges, and future research. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, pages 1–8. IEEE.

Barbu, V. S. and Limnios, N. (2009). *Semi-Markov chains and hidden semi-Markov models*

toward applications: their use in reliability and DNA analysis, volume 191. Springer Science & Business Media.

Barga, R., Gannon, D., and Reed, D. (2010). The client and the cloud: Democratizing research computing. *IEEE Internet Computing*, **15**(1), 72–75.

Berg, F., Dürr, F., and Rothermel, K. (2014). Optimal predictive code offloading. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 1–10.

Bertsekas, D. P. (2005). Dynamic programming and suboptimal control: A survey from ADP to MPC. *Eur. J. Control*, **11**(4-5), 310–334.

Bildea, A., Alphand, O., Rousseau, F., and Duda, A. (2015). Link quality estimation with the Gilbert-Elliot model for wireless sensor networks. In *Proceeding of 2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2049–2054.

Blazek, T. and Mecklenbräuker, C. F. (2018). Measurement-based burst-error performance modeling for cooperative intelligent transport systems. *IEEE Transactions on Intelligent Transportation Systems*, **20**(1), 162–171.

Botta, A. and Pescapé, A. (2015). IP packet interleaving for UDP bursty losses. *Journal of Systems and Software*, **109**, 177–191.

Buyya, R., Vecchiola, C., and Selvi, S. T. (2013). *Mastering cloud computing: foundations and applications programming*. Newnes.

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.

Chen, X., Pu, L., Gao, L., Wu, W., and Wu, D. (2017). Exploiting massive d2d collaboration for energy-efficient mobile edge computing. *IEEE Wireless Communications*, **24**(4), 64–71.

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 the 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 6th Conference on Computer Systems*, pages 301–314.

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 the 8th International Conference on Mobile Systems, Applications, and Services*, pages 49–62.

De, D. (2016). *Mobile cloud computing: architectures, algorithms and applications*. CRC Press.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, **13**(18), 1587–1611.

Elliott, E. O. (1963). Estimates of error rates for codes on burst-noise channels. *The Bell System Technical Journal*, **42**(5), 1977–1997.

Ericsson (2020). Ericsson mobility report 2020. `https://www.ericsson.com/en/mobility-report`.

Forman, G. H. and Zahorjan, J. (1994). The challenges of mobile computing. *Computer*, **27**(4), 38–47.

Gao, W., Li, Y., Lu, H., Wang, T., and Liu, C. (2014). On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Proceedings of 2014 IEEE 22nd International Conference on Network Protocols*, pages 1–12.

Geng, Y., Yang, Y., and Cao, G. (2018). Energy-efficient computation offloading for multicore-based mobile devices. In *Proceedings of 2018 IEEE INFOCOM Conference on Computer Communications*, pages 46–54.

Gilbert, E. N. (1960). Capacity of a burst-noise channel. *Bell System Technical Journal*, **39**(5), 1253–1265.

Goudarzi, M., Zamani, M., and Haghighat, A. T. (2017). A fast hybrid multi-site computation offloading for mobile cloud computing. *Journal of Network and Computer Applications*, **80**, 219–231.

Grinstead, C. M. and Snell, J. L. (2006). Markov chains - chapter 11. In *Grinstead and Snell's 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 *Proceedings of 2011 IEEE/ACIS 10th International Conference on Computer and Information Science*, pages 387–392.

He, P., Zhao, L., Zhou, S., and Niu, Z. (2013). Recursive waterfilling for wireless links with energy harvesting transmitters. *IEEE Transactions on Vehicular Technology*, **63**(3), 1232–1241.

Hu, M., Wu, D., Wu, W., Cheng, J., and Chen, M. (2019). Quantifying the influence of intermittent connectivity on mobile edge computing. *IEEE Transactions on Cloud Computing*, **4**(1), 1–1.

Huang, D., Wang, P., and Niyato, D. (2012). A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, **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*, pages 1–5.

Huth, A. and Cebula, J. (2011). The basics of cloud computing. *United States Computer*.

Hyytiä, E., Spyropoulos, T., and Ott, J. (2015). Offload (only) the right jobs: Robust offloading using the Markov decision processes. In *Proceedings of 2015 IEEE 16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–9.

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.

Josilo, S. and Dan, G. (2017). A Game Theoretic Analysis of Selfish Mobile Computation Offloading. In *Proceedings of 2017 IEEE INFOCOM Conference on Computer Communications*, pages 1–9.

Kamal, R. (2008). *Mobile computing*. Oxford University Press, Inc.

Kim, Y., Lee, J., Jeong, J., and Chong, S. (2016). Multi-flow rate control in delayed wi-fi offloading systems. In *Proceedings of 2016 International Conference on Information Networking (ICOIN)*, pages 274–279.

Ko, S.-W., Huang, K., Kim, S.-L., and Chae, H. (2017). Energy efficient mobile computation offloading via online prefetching. In *Proceedings of 2017 IEEE International Conference on Communications (ICC)*, 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.

Labidi, W., Sarkiss, M., and Kamoun, M. (2015). Energy-optimal resource scheduling and computation offloading in small cell networks. In *Proceedings of 2015 IEEE 22nd International Conference on Telecommunications (ICT)*, pages 313–318.

Lagar-Cavilla, H. A., Tolia, N., De Lara, E., Satyanarayanan, M., and O'Hallaron, D. (2007). Interactive resource-intensive applications made easy. In *Proceedings of ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer*, pages 143–163.

Li, J., Gao, H., Lv, T., and Lu, Y. (2018). Deep reinforcement learning based computation offloading and resource allocation for MEC. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6.

Ling, X., Wu, B., Ho, P.-H., Luo, F., and Pan, L. (2012). Fast water-filling for agile power allocation in multi-channel wireless communications. *IEEE communications letters*, **16**(8), 1212–1215.

Lioumpas, A. S., Karagiannidis, G. K., and Iossifides, A. C. (2007). Channel quality estimation index (CQEI): A long-term performance metric for fading channels and an application in egc receivers. *IEEE transactions on Wireless Communications*, **6**(9), 3315–3323.

Liu, D., Khoukhi, L., and Hafid, A. (2017). Data offloading in mobile cloud computing: A Markov decision process approach. In *Proceedings of 2017 IEEE International Conference on Communications (ICC)*, pages 1–6.

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.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016a). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets 2016*, pages 50–56.

Mao, Y., Zhang, J., and Letaief, K. B. (2016b). Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, **34**(12), 3590–3605.

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 & Tutorials*, **19**(4), 2322–2358.

Masdari, M. and Khezri, H. (2020). Efficient offloading schemes using Markovian models: a literature review. *Computing, Springer*, **102**, 1–44.

Mehmeti, F. and Spyropoulos, T. (2013). Performance analysis of "on-the-spot" mobile data offloading. In *Proceedings of 2013 IEEE Global Communications Conference (GLOBECOM)*, pages 1577–1583. IEEE.

Mehmeti, F. and Spyropoulos, T. (2016). Performance analysis of mobile data offloading in heterogeneous networks. *IEEE Transactions on Mobile Computing*, **16**(2), 482–497.

Meng, T., Wolter, K., Wu, H., and Wang, Q. (2018). A secure and cost-efficient offloading policy for mobile cloud computing against timing attacks. *Pervasive and Mobile Computing*, **45**, 4–18.

Meskar, E., Todd, T. D., Zhao, D., and Karakostas, G. (2015). Energy efficient offloading for competing users on a shared communication channel. In *Proceedings of 2015 IEEE International Conference on Communications (ICC)*, pages 3192–3197.

Meskar, E., Todd, T. D., Zhao, D., and Karakostas, G. (2017). Energy aware offloading for competing users on a shared communication channel. *IEEE Transactions on Mobile Computing*, **16**(1), 87–96.

Miao, G., Zander, J., Sung, K. W., and Slimane, S. B. (2016). *Fundamentals of mobile data networks*. Cambridge University Press.

Miettinen, A. P. and Nurminen, J. K. (2010). Energy efficiency of mobile clients in cloud computing. *HotCloud*, **10**(4), 19.

Mukherjee, A., Bhattacherjee, S., Pal, S., and De, D. (2013). Femtocell based green power

consumption methods for mobile network. *Computer Networks, Elsevier*, **57**(1), 162–178.

Nădăban, S., Dzitac, S., and Dzitac, I. (2016). Fuzzy topsis: A general view. *Procedia Computer Science, Elsevier*, **91**, 823–831.

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

Oo, T. Z., Tran, N. H., Saad, W., Niyato, D., Han, Z., and Hong, C. S. (2016a). Offloading in hetnet: A coordination of interference mitigation, user association, and resource allocation. *IEEE Transactions on Mobile Computing*, **16**(8), 2276–2291.

Oo, T. Z., Tran, N. H., Saad, W., Son, J., and Hong, C. S. (2016b). Traffic offloading via Markov approximation in heterogeneous cellular networks. In *Proceedings of 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 52–60.

Othman, M., Madani, S. A., Khan, S. U., *et al.* (2013). A survey of mobile cloud computing application models. *IEEE Communications Surveys & Tutorials*, **16**(1), 393–413.

Oyerinde, O. O. and Mneney, S. H. (2012). Review of channel estimation for wireless communication systems. *IETE Technical review*, **29**(4), 282–298.

Pan, Y., Pan, C., Zhu, H., Ahmed, Q. Z., Chen, M., and Wang, J. (2017). On consideration of content preference and sharing willingness in D2D assisted offloading. *IEEE Journal on Selected Areas in Communications*, **35**(4), 978–993.

Panigrahi, C. R., Sarkar, J. L., and Pati, B. (2018). Transmission in mobile cloudlet systems with intermittent connectivity in emergency areas. *Digital Communications and Networks*, **4**(1), 69–75.

Patel, M., Naughton, B., Chan, C., Sprecher, N., Abeta, S., Neal, A., *et al.* (2014). Mobile-edge computing introductory technical white paper. *In proceedings of Mobile Edge Computing (MEC) Industry Initiative*, pages 1089–7801.

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

Pu, L., Liu, J., Fang, Y., Li, W., and Wang, Z. (2010). Channel estimation in mobile wireless communication. In *Proceedings of 2010 IEEE International Conference on Communications and Mobile Computing*, pages 77–80.

Pu, L., Chen, X., Xu, J., and Fu, X. (2016). D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration. *IEEE Journal on Selected Areas in Communications*, **34**(12), 3887–3901.

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.

Sadeghi, P., Kennedy, R. A., Rapajic, P. B., and Shams, R. (2008). Finite-state Markov

modeling of fading channels-a survey of principles and applications. *IEEE Signal Processing Magazine*, **25**(5), 57–80.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2013). Heterogeneity in mobile cloud computing: taxonomy and open challenges. *IEEE Communications Surveys & Tutorials*, **16**(1), 369–392.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, **8**(4), 14–23.

Shakkottai, S. and Rappaport, T. S. (2002). Research challenges in wireless networks: a technical overview. In *Proceedings of IEEE 5th International Symposium on Wireless Personal Multimedia Communications*, volume 1, pages 12–18.

Shuja, J., Gani, A., Naveed, A., Ahmed, E., and Hsu, C.-H. (2017). Case of arm emulation optimization for offloading mechanisms in mobile cloud computing. *Future Generation Computer Systems*, **76**, 407–417.

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

Tao, X., Ota, K., Dong, M., Qi, H., and Li, K. (2017). Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wireless Communications Letters*, **6**(6), 774–777.

Terefe, M. B., Lee, H., Heo, N., Fox, G. C., and Oh, S. (2016). Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*, **27**, 75–89.

Tong, L. and Gao, W. (2016). Application-aware traffic scheduling for workload offloading in mobile clouds. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications*, pages 1–9.

Truong-Huu, T., Tham, C.-K., and Niyato, D. (2014). To offload or to wait: An opportunistic offloading algorithm for parallel tasks in a mobile cloud. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science*, pages 182–189.

Voas, J. and Zhang, J. (2009). Cloud computing: New wine or just a new bottle? *IT Professional*, **11**(2), 15–17.

Wu, H. and Wolter, K. (2017). Stochastic analysis of delayed mobile offloading in heterogeneous networks. *IEEE Transactions on Mobile Computing*, **17**(2), 461–474.

Wu, H., Knottenbelt, W., and Wolter, K. (2015). Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics. In *Proceedings of the 27th IEEE International Teletraffic Congress*, pages 134–142.

Wu, Q., Liu, H., Wang, R., Fan, P., Fan, Q., and Li, Z. (2019). Delay-sensitive task offloading in the 802.11 P-based vehicular fog computing systems. *IEEE Internet of Things Journal*, **7**(1), 773–785.

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.

Yu, F., Chen, H., and Xu, J. (2018). Dmpo: Dynamic mobility-aware partial offloading in mobile edge computing. *Future Generation Computer Systems*, **89**, 722–735.

Zafer, M. and Modiano, E. (2007). Minimum energy transmission over a wireless fading channel with packet deadlines. In *Proceedings of the 46th IEEE 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 *Proceedings of the 4th IEEE International Conference on Universal Personal Communications*, pages 211–215.

Zhang, C., Gu, B., Liu, Z., Yamori, K., and Tanaka, Y. (2016). A reinforcement learning approach for cost-and energy-aware mobile data offloading. In *Proceedings of the 18th IEEE Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–6.

Zhang, C., Gu, B., Liu, Z., Yamori, K., and Tanaka, Y. (2017a). Cost-and energy-aware multi-flow mobile data offloading under time dependent pricing. In *Proceedings of the 13th International Conference on Network and Service Management (CNSM)*, pages 1–6.

Zhang, J., Hu, X., Ning, Z., Ngai, E. C.-H., Zhou, L., Wei, J., Cheng, J., and Hu, B. (2017b). Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. *IEEE Internet of Things Journal*, **5**(4), 2633–2645.

Zhang, W., Wen, Y., and Wu, D. O. (2013a). Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In *Proceedings of IEEE INFOCOM*, pages 190–194.

Zhang, W., Wen, Y., Guan, K., Kilper, D., Luo, H., and Wu, D. O. (2013b). 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. (2014). Collaborative task execution in mobile cloud computing under a stochastic wireless channel. *IEEE Transactions on Wireless Communications*, **14**(1), 81–93.

Zhang, X. and Cao, Y. (2018). Mobile data offloading efficiency: a stochastic analytical view. In *Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6.

Zhang, Y., Niyato, D., and Wang, P. (2015). Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, **14**(12), 2516–2529.