Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften

(Dr. rer. nat)

Analysis of Offloading Decision Making in Mobile Cloud Computing

eingereicht
am Institut für Informatik
des Fachbereichs Mathematik und Informatik
der Freien Universität Berlin
von

Huaming Wu

Berlin, 2015

Gutachter:

Prof. Dr. Katinka Wolter
Department of Computer Science
Freie Universität Berlin, Germany

Prof. Dr. William J. Knottenbelt
Department of Computing
Imperial College London, United Kingdom

Tag der Disputation: 23. November 2015

Eidesstattliche Erklärung

Ich versichere, dass ich die Doktorarbeit selbständig verfasst, und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit hat keiner anderen Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass bei Verwendung von Inhalten aus dem Internet ich diese zu kennzeichnen und einen Ausdruck mit Angabe des Datums sowie der Internet-Adresse als Anhang der Doktorarbeit anzugeben habe.

Author's Declaration

I hereby declare to have written this thesis on my own. I have used no other literature and resources than the ones referenced. All text passages that are literal or logical copies from other publications have been marked accordingly. All figures and pictures have been created by me or their sources are referenced accordingly. This thesis has not been submitted in the same or a similar version to any other examination board.

Huaming Wu

Berlin, den 2, September 2015

Abstract

Besides lightweight Internet applications, there is an increasing demand from mobile users for computation-heavy and energy-hungry applications that are being deployed to mobile devices. Running complex applications on such devices is however challenging due to the strict constraints on their resources, e.g. limited computational capacity, battery lifetime and network connectivity. Offloading computation-intensive parts of mobile applications to a capable cloud server is an effective way to alleviate a tussle between resource-constrained mobile devices and resource-hungry mobile applications, and thus boosting the device's performance.

Offloading decisions may involve multiple factors such as resource and component availability, connectivity intermittence and network capacity. Potential benefits obtained from mobile cloud offloading include time and energy saving, which can be achieved by deciding *what*, *where*, *how* and *when* to offload correctly. Unlike previous works that only focus on some specific issues in the offloading process, our time- and energy-aware offloading decision process is based on multiple perspectives. For instance, *what* defines the name of the candidate tasks to be offloaded through application partitioning; *where* describes the type of surrogate and choosing the appropriate offloading target (e.g. local, cloudlet and cloud) in which the application has to be offloaded; *how* introduces offloading plans that enable the device to schedule offloading operations; *when* considers the offloading conditions and dynamic changes of context, since sometimes offloading may not be worthwhile at all.

This work covers both the theoretical and practical sides of offloading decision making. Mathematical and queueing models are applied and evaluated by numerical simulations and real world experiments. Specifically, the contributions of this thesis can be summarized as follows:

- Studying how to effectively and dynamically partition a given application into local and remote parts while keeping the total cost as small as possible.
- Proposing static and dynamic methods based on multi-criteria decision making, in order to find out what resource is the most appropriate one for offloading.

- Comparing different offloading operations of a mobile terminal equipped with multiple radio access technologies (e.g. 3G, LTE, Bluetooth, WLAN) and determining how to leverage the complementary strength of WLAN and cellular networks by choosing heterogeneous wireless interfaces for offloading.
- Exploring the energy-delay tradeoffs for different types of applications (e.g. delay-tolerant and delay-sensitive applications) based on some combined metrics.

Zusammenfassung

Neben leichtgewichtigen Internetanwendungen verwenden Besitzer mobiler Geräte immer stärker berechnungsintensive und energiehungrige Programme, die dafür nutzbar gemacht werden. Solche Anwendungen auf mobilen Geräten laufen zu lassen, erweist sich allerdings als schwierig, da sie in Bezug auf ihre Ressourcen stark limitiert sind: Sie verfügen über eine geringere Rechenleistung, eine begrenzte Akkulaufzeit und über eine schlechtere Netzanbindung. Um trotzdem komplexe Anwendungen auf mobilen Geräten laufen zu lassen, können deren berechnungsintensive Anteile auf einen performanten Cloud-Server ausgelagert werden (Computation Offloading). Dadurch können die Probleme durch ressourcenfordernde Anwendungen auf ressourcenbeschränkten mobilen Geräten drastisch gemildert und somit die Leistung dieser Geräte gesteigert werden.

Offloading-Entscheidungen werden aufgrund diverser Faktoren getroffen: So werden unter anderem die Verfügbarkeit von Ressourcen und Komponenten, Verbindungsschwankungen und die Netzwerkauslastung berücksichtigt. Vorteile der Verwendung von Offloading sind reduzierte Anwendungslaufzeiten und ein geringerer Energieverbrauch des mobilen Gerätes. Um dies zu erreichen, werden korrekte Antworten für folgende Fragestellungen benötigt: Was wird *was*, *wohin*, *wie* und *wann* ausgelagert. Im Gegensatz zu vorherigen Arbeiten, die sich nur auf eine dieser Fragen des Offloading-Prozesses konzentriert haben, beruht unser Zeit und Strom berücksichtigender Offloading-Entscheidungsprozess auf mehreren dieser Aspekte. *Was* bestimmt mithilfe von Anwendungspartitionierung die auszulagernden Anwendungsteile; *wohin* beschreibt den Ort, wohin ein Anwendungsteil ausgelagert und ausgeführt wird (z.B.: mobiles Gerät, Cloudlet oder Cloud); *wie* erlaubt es dem mobilen Gerät, auszuführende Offloading-Operationen in ihrer Reihenfolge zu bestimmen (Offloading-Pläne); *wann* berücksichtigt die kontinuierlichen Veränderungen der Umgebung, da Offloading sich unter bestimmten Bedingungen nicht lohnt.

Diese Arbeit beschäftigt sich sowohl mit der theoretischen als auch mit der praktischen Seite der Offloading-Entscheidungsfindung. Zur Anwendung kommen mathematische Modelle und Warteschlangenmodelle, die durch numerische Simulationen und reale Experimente evaluiert werden.

Genauer können die Beiträge dieser Arbeit wie folgt zusammengefasst werden:

- Erforschung einer Methode, die eine gegebene Anwendung effektiv und dynamisch in lokal und remote berechnete Anteile partitioniert, während die totalen Kosten dabei so gering wie möglich bleiben.
- Entwicklung statischer und dynamischer Methoden, die mithilfe von Multikriterien-Entscheidungsfindung diejenige Ressource bestimmen, die sich am besten für Offloading eignet.
- Vergleich verschiedener Offloading-Operationen bei einem mobilen Gerät mit Zugriff auf mehrere Funk-Technologien (z.B.: 3G, LTE, Bluetooth, WLAN) und Ermittlung, wie die ergänzenden Stärken von WLAN und Mobilfunknetzen durch Nutzung heterogener Funk-Schnittstellen für Offloading nutzbar gemacht werden können.
- Untersuchung, inwiefern sich der Konflikt zwischen Energieeinsparung und Reduzierung Offloading-bedingter Verzögerung bei verschiedenen Anwendungsarten (z.B.: verzögerungstolerante und -sensitive Anwendungen) auswirkt. Dazu werden kombinierte Metriken verwendet.

Acknowledgements

My study is under the financial support by China Scholarship Council (CSC) and has been carried out at Computer Systems & Telematics (CST) group at Freie Universität Berlin, Germany. I am very thankful to CSC and CST for providing me with such a valuable learning opportunity. I would like to express my sincere gratitude to everyone who contributed to the completion of this thesis. All of you made my study in Berlin so wonderful!

First and foremost, I would like to thank my supervisor Prof. Dr. Katinka Wolter for her invaluable advice and encouragement. She has been more than a perfect supervisor during my life at FUB, she has been an advisor, mentor, and a good friend. During my early days at FUB, she made it much easier for me to adjust to a new culture. She provided me with much guidance and valuable feedback in the research leading to this dissertation. Without her advice and patience, this thesis would have never been possible. I really appreciate everything she has done for me.

I would like to take this opportunity to thank my colleagues for creating an ever nice and friendly working atmosphere in the institute. I enjoyed working with them very much. I wish to thank Yubin Zhao and Yuan Yang, who were an inexhaustible source of inspiration and helped whenever I was in need. Without their countless discussions I had certainly been stuck in blind alleys several times.

Thanks to my Chinese colleagues Yi Sun, Qiushi Wang, Tianhui Meng, Jialu Hu, Chenxu Pan and Zhihao Shang for their help and memorable time. I earned a solid friendship. I also would like to thank Prof. Dr.-Ing. Jochen Schiller, Prof. Dr. Marcel Kyas, Prof. Dr. Mesut Güneş, Dr. Philipp Reinecke, Dr. Matthew Orlinski, Norman Dziengel, Stephanie Bahe, Matthias Wählisch, Alexandra Danilkina, Dr. Emmanuel Baccelli, Oliver Hahm, Martin Seiffert, Stephan Adler and all the members of CST for their kindly daily help and for sharing with me their PhD study experience, which makes me quickly get involved into the life of German research.

Last but not least, I am very grateful for the love, support and patience of my parents, who have dedicated their entire life for my education.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Mobile devices, such as smartphones, smartwatches, tablets and notebooks, are restricted by their battery lifetime, computational and storage capacity [44]. These constraints prevent mobile devices from performing complicated tasks and widely running content-rich or computation-intensive mobile applications. This is not just a temporary limitation of current mobile hardware technology, but is intrinsic to mobility [105].

Battery life is the top concern of mobile users. An investigation[1] engaged by thousands of users around the world indicated that "over 75 percent of respondents said better battery life is the main feature they want from a future converged device". Longer battery life is more important than all other features, including cameras or storage. Mobile devices are getting advanced in terms of processing speed, sharper screen and more sensors which lead to higher energy consumption. Smartphones are no longer used only for voice communication but are more and more frequently used for watching videos, web surfing, interactive gaming, augmented reality and other purposes which consume huge power and seriously shorten the smartphones' battery life as a result. Further, these applications are too computation intensive to be executed on a mobile system. Even though battery technology has been steadily improving, it is not able to keep up with the rapid growth of power consumption of the mobile systems [106].

Response time is another primary constraint for mobile systems. Mobile applications (e.g. face recognition, speech and object recognition, interactive games and mobile augmented reality) are becoming increasingly intensive and sophisticated that require ever increasing amounts of compu-

[1] Battery life concerns mobile users. http://edition.cnn.com/2005/TECH/ptech/09/22/phone.study/

tational capabilities [57]. Response time is important because most of these mobile applications are usually operated in real-time and user-interactive [70]. It may take a long time to obtain the results due to the limited processing speed of the mobile systems.

Cloud computing is becoming increasingly popular these days due to its features like elasticity, scalability, low cost and so on [35]. Mobile cloud computing (MCC), which combines the strength of cloud computing with the convenience of mobile terminals, is emerging as a new computing paradigm that aims to augment computational capabilities of mobile devices, taking advantage of the abundant resources hosted by clouds. It refers to an infrastructure where both the data storage and processing happen outside of mobile devices [28]. Mobile users can access the applications, data, and cloud services from anywhere at any time through the internet using a thin mobile client or a web browser, etc.

Along with the maturity of mobile cloud computing, mobile cloud offloading (MCO) is becoming a promising method to reduce execution time and extend battery life of mobile devices [71]. Its main idea is to augment execution through migrating heavy computation from mobile devices to resourceful cloud servers and then receive the results from them via wireless networks. Offloading is an effective way to overcome the resources and functionalities constraints of the mobile devices since it can release them from intensive processing and increase performance of the mobile applications, in terms of response time [31]. Offloading brings many potential benefits, such as energy saving, performance improvement, reliability improvement, ease for the software developers and better exploitation of contextual information [17].

1.2 Main Challenges and Research Questions

While offloading has been widely considered for saving energy and time, it still faces many challenges due to heterogeneous wireless environments, different applications and resources, which may significantly impede the improvement of service quality [31]. According to Fig. 1.1, offloading decisions in mobile cloud computing may involve multiple factors as follows:

- ***Resource***: resource heterogeneity between mobile devices and available cloud services is a challenge for offloading. A mobile client can be thin (no computation on the offloading task) or thick (doing computation on tasks before offloading). It can opt for offloading when its available resource is not adequate either to execute the tasks or to achieve the desired performance (e.g. short execution time and/or low battery energy requirements). There are a variety of cloud resources that can be selected, a mobile device can offload its application either to a remote cloud or a nearby cloudlet [105], which is a cluster of multi-core computers

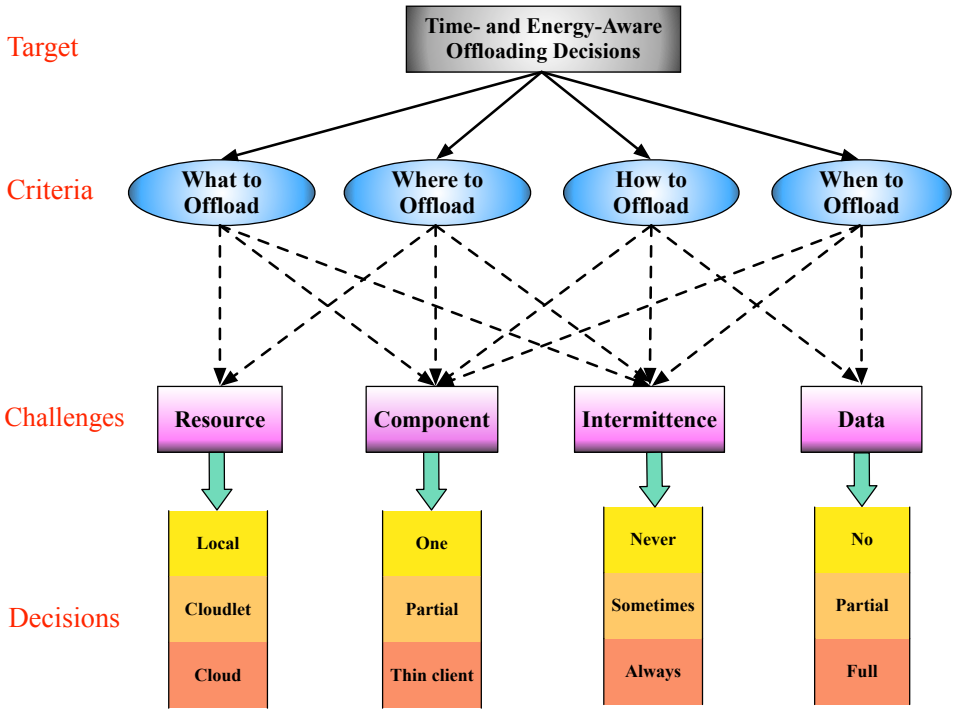


Figure 1.1: Structure of this dissertation

located at one-hop latency accessible through a WLAN hotspot. Mobile devices can discover unknown surrogates and automatically establish service connection using a service discovery technique [84].

- ***Component***: an application can consist of several components, and some should be unconditionally processed locally on the mobile device while others are flexible that can be processed either locally or remotely in a cloud. Since offloading a whole application to the cloud is not always possible or effective, a decision of which portion of the application should be offloaded and where to place the execution (locally or remotely) should be made based on either the minimum response time or the minimum energy consumption.
- ***Intermittence***: mobile users use heterogeneous wireless interfaces to access the cloud service and offload tasks. Different types of networks usually have different bandwidth and network latency. For example, tasks that require low response time should not be offloaded using high latency wireless networks; data-intensive tasks should not be offloaded using low-rate networks. Intermittent connectivity issues may exist due to heterogeneous wireless environments, device mobility and cloud resource availability [56]. Unstable connectivity of mobile networks can have a great impact on the offloading decisions [25]. High latency and energy

consumption can be caused by the intermittent nature of wireless networks, which makes executing applications locally more advantageous in certain circumstances [12]. If a mobile device moves outside the network coverage area, then offloading will be interrupted until the device arrives at a covered area again.

- ***Data***: the amount of data for an offloading task includes: (i) the size of the task's code and input data when initially the task is offloaded for remote execution, (ii) the amount of code offloading when native and remote modules communicate during remote execution, and (iii) the amount of output data generated from remote execution of the task [84]. Data transfers will incur additional communication costs, which could be critical for data intensive tasks that might not benefit from offloading. For example, remote execution may reduce the execution time of a task, however, data transfer during remote execution may consume more energy than during local execution. Therefore, offloading decisions must include: how to effectively transmit data and when to offload data from mobile devices to cloud servers in order to save time and energy.

1.3 Contribution of this Dissertation

Response time and energy consumption are two primary aspects for mobile systems that must be considered when making offloading decisions. The performance of an offloaded task is judged based on the goals set by the user. Accordingly, we consider three objectives as follows:

- **Shorten response time**: from the perspective of a mobile device, response time is defined as the duration between sending the application to the cloud and receiving the results back from it. Reducing the response time is becoming increasingly important, especially for computation-intensive mobile applications. When the amount of computation is very large, it will take such a long time to get the result that it fails to meet the user's need, and thus the application should be offloaded to the cloud, in order to save time and improve performance. Therefore, we take decisions to offload only if response time will reduce no matter the impact on energy consumption.
- **Reduce energy consumption**: the energy spent on a mobile device during the offloading period, is another primary aspect that must be considered. We aim to optimize the energy consumption of a mobile device by estimating and evaluating the tradeoff between the energy consumed by local processing versus offloading the application for remote execution [46]. In this situation, offloading is applied only if energy consumption is expected to reduce no matter the expected impact on response time.

- **Achieve combination of the above**: both energy and time saving are crucial design targets. We do offloading only if both the response time and energy consumption are expected to improve [58]. It is possible that achieving one offloading goal may affect the other goal, e.g. executing a task on a service node might decrease the response time of the task, however, it might not be energy-efficient. We study the tradeoff between the mean energy consumption and mean response time, which is a non-trivial multi-objective optimization problem. For example, sometimes a short remote execution time for a task is more important even though more energy will be spent due to offloading than for local execution, and vice-versa [84].

According to Fig. 1.1, the major contributions of this work are fourfold:

1) **What to offload**: by effectively and dynamically partitioning an application into local and remote parts while keeping the total cost as small as possible, we determine which portions of the application to run on mobile devices and which portions on cloud servers [119, 124, 128, 134]. We propose an application partitioning algorithm [124], which aims at finding the optimal partitioning scheme under various cost models in dynamic mobile environments. It provides a stable low time complexity method and can significantly reduce execution time and energy consumption by optimally distributing tasks between mobile devices and cloud servers, and in the meantime, it can well adapt to wireless environment changes.
2) **Where to offload**: we try to find an optimal cloud service for offloading when considering multiple criteria simultaneously [127, 129]. An energy-efficient decision making algorithm based on *Lyapunov optimization* is proposed in [126] (i.e. to determine whether each application component should be run locally, or remotely on a cloud infrastructure, directly or via a cloudlet under available wireless networks). The algorithm has low complexity and can significantly reduce the energy consumption while leveraging delay tolerance. Further, we would like to know how effective and efficient offloading systems are and what factors influence their performance. With this purpose, we introduce a mathematical model and analyze offloading systems with and without failures [135].
3) **How to offload**: we explore the methods of how to deploy offloadable applications in a more optimal way, by dynamically and automatically determining which application modules should be deployed on the cloud server and which should execute on the mobile device to achieve a particular performance target. A queueing model has been proposed to capture the tradeoff between the energy consumption and the response time based on an additive energy-performance metric, where static and dynamic offloading polices are analyzed. We propose two types of offloading models: the uninterrupted and interrupted strategies [132, 133], which are compared based on several metrics [121]. Further, we generalize the offloading models to

the scenario with more than two types of networks.

4) **When to offload**: this is done by exploring the decision whether to offload or not and by means of which communication interface to use [130] in its effect on different tradeoff metrics for response time and energy consumption. Through dynamic scheduling and link selection between the mobile device and the cloud, energy consumption can be minimized by leveraging delay tolerance. Different transmission schedulers with energy-efficient link selection policies are proposed and compared [131]. Further, we develop an analytical framework for delayed offloading systems with reneging (not offload) and service interruptions [125]. Some tasks are abandoned from the offloading process and executed locally when the deadline expires, which can be used to predict the average performance and energy consumption.

1.4 Thesis Structure

According to Fig. 1.1, this thesis is structured mainly in four chapters, each one addressing one objective of offloading decision making. The chapters are organized as follows:

Chapter 2 describes characteristics of mobile cloud offloading, general offloading model and offloading process. We survey existing research efforts regarding mobile offloading on time saving, energy saving, and time and energy combined saving.

Chapter 3 illustrates the issue of what to offload. We construct weighted consumption graphs according to the estimated computational and communication costs, and further design a partitioning algorithm for offloading.

In Chapter 4 we focus on where to offload. First we use multi-criteria decision analysis to find the most suitable resource for offloading. Then a dynamic offloading decision algorithm is proposed to determine where to offload to. Finally we analyze the performance of offloading systems.

In Chapter 5 we analyze how to perform offloading via wireless local area network (WLAN) or cellular networks. First we present a general queueing model for offloading where both static and dynamic offloading polices are analyzed. Then two types of offloading assignment strategies are proposed and compared. Finally, energy-delay tradeoffs are evaluated by using different metrics.

Chapter 6 shows our efforts on when to offload in heterogeneous wireless environments. First we analyze the tradeoff (time vs. energy or computation vs. communication) in offloading decision making. Then transmission schedulers are proposed to minimize the energy consumption. Finally an analytical framework for delayed offloading with reneging and service interruption is developed.

Chapter 7 concludes this thesis by integrating the current efforts into offloading decision making in mobile cloud computing. We describe the future directions opened by our work.

Chapter 2

Aspects of Mobile Cloud Offloading

In this chapter, we provide a brief introduction to mobile cloud offloading. A general offloading process is described, which covers multiple aspects such as profiling, metrics, application partitioning, and offloading decisions. We focus on time- and energy-aware offloading decision making by dividing into: *what*, *where*, *how* and *when* to offload. Existing work on energy and/or time saving is investigated.

2.1 The Context of Mobile Cloud Offloading

In this section we provide an elementary material on offloading by describing a generic offloading system. Throughout the section, we refer to the most important aspects that lead to offloading for mobile devices: different types of mobile applications, mobile characteristics like heterogeneous wireless environments.

2.1.1 Generic Offloading System

The emergence of cloud computing allows to solve a number of problems affecting mobile computing, since the cloud can be seen as a system characterized by unlimited resources that can be accessed anytime and anywhere in the world. A large number of cloud infrastructures are appearing these days for data storage and processing, e.g. Amazon EC2 [5], Apple iCloud [8], Microsoft Windows Azure [81], and Google App Engine [41]. Such systems use proprietary cloud platforms to provide different kinds of services.

The convergence of mobile and cloud computing has been studied for a number of years and is still a hot topic, because of the dynamics in mobile computing and the challenges that continue to

arise [3]. Cloud offloading is one of the emerging trends in distributed computing involving mobile devices. Figure 2.1 illustrates a generic mobile cloud offloading system which is organized as a two- or three-level hierarchy:

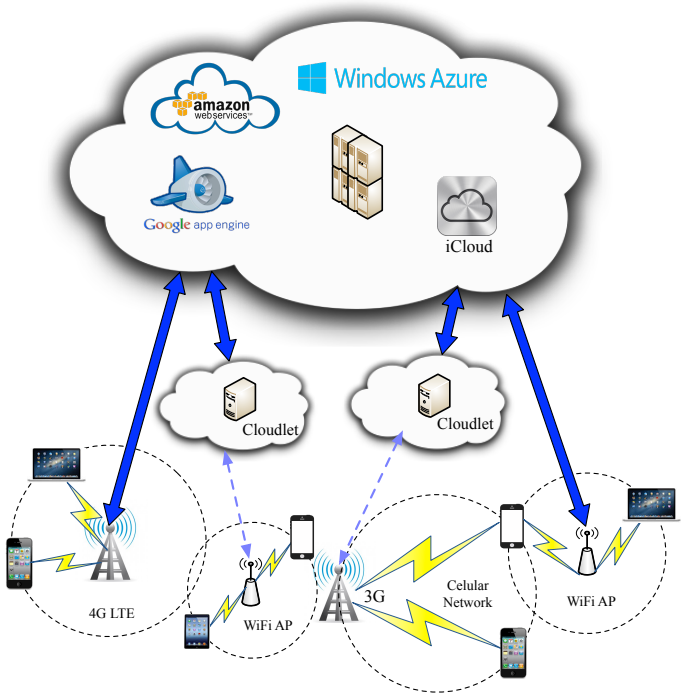


Figure 2.1: A generic mobile cloud offloading system

- **Two-Level Offloading**: mobile devices have limited computational capacity and battery life, heavy multimedia and signal processing are unable to run on them [87]. Rather than running applications locally and directly requesting data from content providers, a mobile device can offload parts of its workload to a powerful cloud server via one or more communication networks, taking advantage of the abundant cloud resources to help gather, store, and process data. This offloading scheme critically depends on a reliable end-to-end communication and on the availability of the cloud [107]. In addition, it suffers from high network access latency and low network bandwidth. Access to the cloud is usually influenced by uncontrollable factors, such as the instability and intermittency of wireless networks.
- **Three-Level Offloading**: rather than relying on a remote cloud, the resource poverty of a mobile device can be addressed by using a nearby resource-rich cloudlet via a WLAN hotspot to decrease latency and lower battery consumption. A cloudlet is viewed as a trusted, resource-rich computer or cluster of computers that is well-connected to the internet and is available

for use by nearby mobile devices [105]. Therefore, the application task is first offloaded to the cloudlet, and then onto the remote cloud through a stable internet connection. This architecture reduces latency by using a single-hop network and potentially saves battery by using WiFi or short-range radio instead of broadband wireless which typically consumes more energy [23]. Besides, loss or destruction of a cloudlet is not catastrophic since it only contains soft state such as cache copies of data or code that is also available elsewhere.

Mobile users are easily subject to dynamically changing network conditions due to their mobility, which makes it hard to make good offloading decisions in mobile environments [65]. Mobile network environments have a great influence on the performance of offloading systems. For example, if a mobile device has a stable network connectivity and plenty of network bandwidth, then offloading will result in better performance in terms of response time. Thus, taking a high-quality offloading decision requires a good understanding of network conditions. Also, near-future network conditions should be taken into account [65].

2.1.2 Classification of Applications

Different types of applications usually give different relative importance to both factors of response time and energy consumption. There exists a fundamental tradeoff between the mean energy consumption and mean response time for different applications [103, 114].

- *Delay-Tolerant Applications*: many mobile applications (e.g. iCloud, Dropbox and participatory sensing) deal with video, audio, sensor data, or access large databases on the Internet, which are less sensitive to network delays. Participatory sensing applications are a good example of data-intensive yet delay-tolerant applications. The collective sampling of sensor data acquired by a number of sensor nodes creates a body of knowledge on parameters such as personal resource consumption, dietary habits and urban documentation [103]. Data is uploaded from a smartphone to a back-end cloud server either through the cellular network or any available WiFi network. Some of the sensor information is not time-critical and its submission to the server may be delayed until the device enters an energy-efficient network [19]. Users can browse or search the obtained archives through a website at the server side. For delay-tolerant applications, response time is less critical and optimizing energy usage is more relevant.
- *Delay-Sensitive Applications*: when running delay-sensitive applications (e.g. language translator, face recognition, video conferencing, vehicular communications) on mobile devices, mobile users desire a fast response which is comparable to their normal cognitive capabilities. Thus, for better user experience the response time of cognitive applications should be low. Task offloading can be exploited to use cognitive applications ubiquitously by executing them

remotely on computing nodes. For delay-sensitive applications, fast response is a primary concern. The offloading scheme in which cloud services are available with short network latencies (e.g. WiFi networks) can serve in a better way by providing low response time.

2.1.3 Heterogeneous Wireless Environments

Mobile devices often have multiple wireless interfaces with varying availability, delay and energy cost, such as 2G, EDGE, 3G, 4G, LTE and WiFi for data transfer. The difference between WiFi and cellular networks is as shown in Fig. 2.2.

	Cellular	WiFi
Delay	High	Low
Availability	High	Low
Energy-Efficiency	Low	High

Figure 2.2: Comparison of WiFi and cellular networks

- *Data Rate*: the achievable data rates for different radio transmissions depend on the environment and can vary significantly: EDGE supports tens of Kbps to a few hundred Kbps, 3G supports 2 Mbps peak stationary and 384 Kbps peak mobile bandwidth are becoming widely used [21], WiFi provides faster connection to an infrastructure network with several Mbps being common, at orders of magnitude faster speeds than 3G and LTE supports 1 Gbps peak stationary and 100 Mbps peak mobile bandwidth.
- *Availability*: cellular networks such as EDGE and 3G, usually have much higher availability than WiFi, in particular EDGE has very high coverage. There are WiFi hotspots at home, office, and public places like universities and cafes, however their coverage is limited. LTE networks can achieve much faster speed than common WiFi routers, but are still not widely available.
- *Energy-Efficiency*: the energy usage for transferring a fixed amount of data can differ by an order of magnitude or more [103]. In general, the WiFi interface is more energy-efficient than the cellular interface. WiFi consumes less power compared to EDGE which in turn consumes

less power than 3G. While in most situations LTE uses most energy and WiFi the least.

Not only the availability and quality of access points (APs) may vary from place to place, but also the uplink and downlink bandwidths fluctuate frequently due to multiple factors such as weather, mobility and building shield [72]. Data transmission in good connectivity consumes much less energy than that under bad conditions.

2.2 Offloading Process

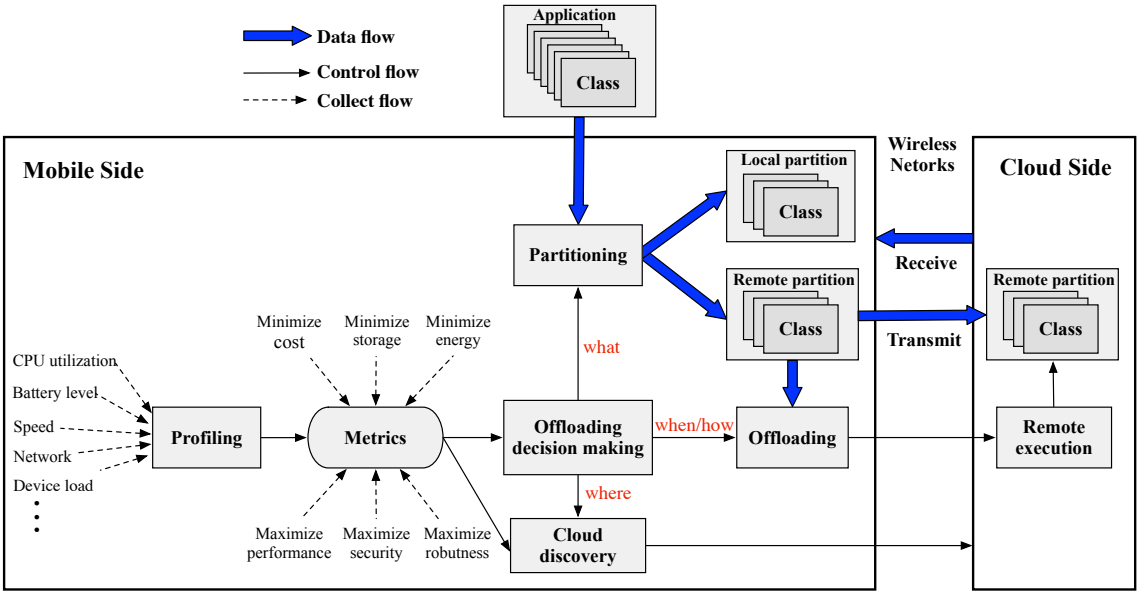


Figure 2.3: System architecture of the offloading service

Figure 2.3 describes a general offloading process [137]. On the mobile side, upon receipt of an offloading request, resource information such as CPU utilization, battery level, speed and network bandwidth, are collected by the *profiling module*. On the basis of the collected information and the *metrics module*, the *offloading decision module* invokes the *partitioning module* to cut the composing classes of the application into local partition and remote partition, where the former is executed locally on the mobile device and the latter will be offloaded to a dedicated cloud server (*what to offload*). Then the *cloud discovery module* is invoked to find an appropriate cloud service for offloading (*where to offload*). The remote partition classes are migrated to the cloud side via wireless networks by the *offloading module* for remote execution (*how* and *when to offload*). The offloaded classes can interact with the classes in the local partition. Once completed, the results are sent back to the mobile side. The main modules are further analyzed in the following subsections.

2.2.1 Profiling

Profiling is the process of gathering the information required to make offloading decisions. Such information may consist of the computation and communication costs of the execution units (program profiler), the network status (network profiler), and the mobile device specific characteristics such as energy consumption (energy profiler). Profilers are needed to collect information about the device and network characteristics, which is a critical part of the partitioning algorithm: the more accurate and lightweight they are, the more correct decisions can be made, and the lower overhead is introduced [58]. We will in the following introduce all types of profilers.

Program Profiler

A program profiler (static or dynamic) collects characteristics of applications, such as the execution time, the memory usage and the size of data.

Static analysis obtains the control flow graph of an application by analyzing the bytecode with nodes representing objects and edges representing relations between objects. We can get all the objects and the relations between them based on method invocations by traversing the graph. Constructing call graphs by hand and without the help of analysis tools would have cost far more time and resources. Many tools and frameworks have been developed to generate the call graph, e.g. Spark [67], Cgc [4], and Soot[2].

Dynamic profiling is also adopted to obtain weights of the nodes and edges. Since there is a certain ratio of execution time to the total bytecode instruction count for Java programs, execution time of objects can be evaluated by the corresponding bytecode instruction count [14]. Data transmission between objects include parameters and return values of method invocations. With Java bytecode rewriting and combined with pretreatment information, we can obtain the execution time for each object (node weight) and the transmission time for each invocation (edge weight), respectively. These weights can be dynamically assigned according to the different processing capabilities of the cloud server and the wireless bandwidth.

Network Profiler

A network profiler collects information about wireless connection status and available bandwidth. It measures the network characteristics at initialization, and it continuously monitors environmental changes. Network throughput can be obtained by measuring the time duration when sending a

[2]Fully implemented in Java http://sable.github.io/soot/, is a framework for analyzing and transforming Java and Android applications

certain amount of data as in [20]. Due to the mobile nature, the status of a wireless connection could change frequently (e.g. user moves to other location). Fresh information about a wireless connection is critical for the optimizer to make correct offloading decisions.

The profiler tracks several parameters for the WiFi and 3G interfaces, including the number of packets transmitted and received per second, and receiving and transmitting data rate [58]. These measurements enable better estimation of the current network performance being achieved.

We use Speedtest[3] to measure the mobile network bandwidth. Actual devices (see Table 2.1) are applied in mobile cloud environments with various mobile communication networks. Here, we measure wireless bandwidth statistics under representative scenarios as shown in Table 2.2. Specifically, during one week in May, 2015, I stayed inside some buildings or randomly walked around our campus, carrying two smartphones (Xiaomi Redmi 2 and Samsung Galaxy S6) equipped with WiFi and cellular interfaces.

Table 2.1: Mobile Device Specifications

Device	CPU	Memory	Communication Method	Technology
Xiaomi Red 2	Quad-core 2.1 GHz Cortex-A57	1GB RAM	WiFi	IEEE 802.11g
			3G	HSPAP/HSUPA
			4G	LTE
Samsung Galaxy S6	Quad-core 1.2 GHz Snapdragon 410	3GB RAM	WiFi	IEEE 802.11g
			3G	HSPAP/HSUPA
			4G	LTE

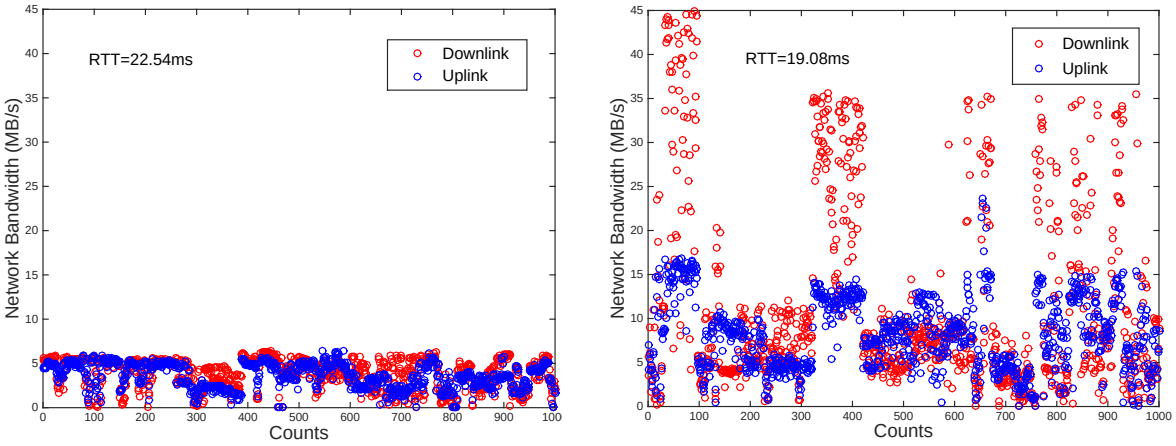
Table 2.2: Network Trace Data Acquisition

Scene	Place	Mobility
Indoor (Static)	Office, Library, Classroom, Lecture Hall, Kitchen, Washing Room, Meeting Room, Student Cafeteria, Laboratory	Low
Outdoor (Dynamic)	Walking around the campus	Medium

The measured mobile network traces are depicted in Figs. 2.4-2.6. It is found that the bandwidths of both WiFi and cellular networks (3G and LTE) vary considerably over time and are highly unpredictable. Indoor WiFi, which has a good coverage, is stable and fast. Even in the same scenario, different mobile devices may record different levels of transmission speeds for example, the

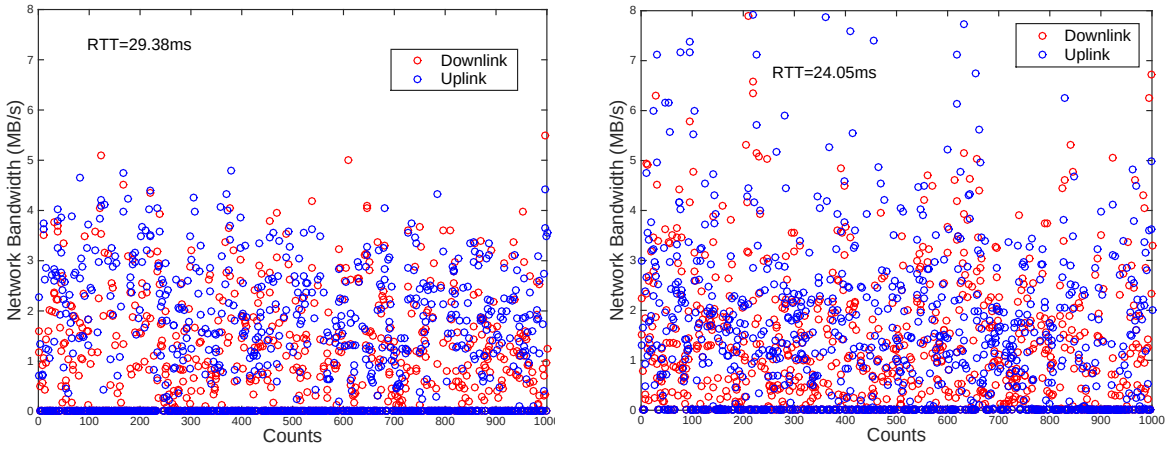
[3]A free connection analysis tool, which shows real-time download and upload graphs, stores results both locally and on the Internet for sharing, http://www.speedtest.net/

Samsung S6 has much higher bandwidth than the Xiaomi Redmi 2 in indoor environments. This is because the two devices consist of different hardware and software. The mobility of users has a significant impact on the network connection bandwidth quality. Outdoor WiFi experience varying signal strength and frequent intermittent connectivity make it unavailable from time to time. On the contrary, cellular networks are much more stable and also provide near-ubiquitous connectivity. Further, cellular connections can suffer from high latencies or round-trip time (RTT) and slow data transfers when compared with WiFi.



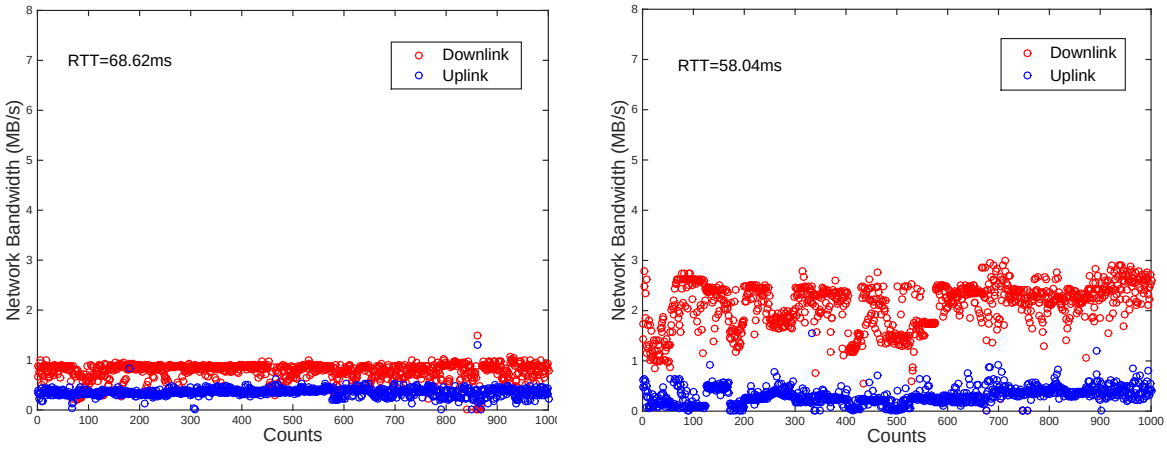
(a) Xiaomi Redmi 2 (b) Samsung Galaxy S6

Figure 2.4: The downlink and uplink bandwidths of WiFi in indoor environments



(a) Xiaomi Redmi 2 (b) Samsung Galaxy S6

Figure 2.5: The downlink and uplink bandwidths of WiFi in mobile environments



(a) 3G with Samsung Galaxy S6 (b) LTE with Samsung Galaxy S6

Figure 2.6: The downlink and uplink bandwidths of cellular networks in mobile environments

Energy Profiler

There are two ways to estimate the energy consumption, namely, software and hardware monitors. For example, some works [23, 47] used a power meter attached to the smartphone's battery to build an energy profile. Power Monitor (e.g. Monsoon monitor) is a device that measures energy consumption when data is transmitted from the mobile device to the cloud server by supplying a certain level of power to the mobile device. We use PowerTutor [4] to measure the power consumption of the applications. Although PowerTutor does not give as accurate results as a hardware power monitor does, the result is still reasonable and does provide some value because it gives the detailed energy consumption information for each hardware component.

In Fig. 2.7 both energy consumption and transmission time increase in proportion to transferred file sizes. When the same volume of data was transferred, WiFi has relatively lower energy consumption than 3G. Moreover, the device's energy consumption via each communication network is proportional to its data transmission time.

2.2.2 Metrics

The execution cost for a task is a user-defined metric, which could be a task's execution time, usage of CPU power, and consumption of battery energy by the task, etc. In this thesis, we investigate modeling and optimality by considering two parameters: the energy consumption on the mobile

[4]PowerTutor is an application for Android phones that provides accurate, real-time power consumption estimates for power-intensive hardware components, http://ziyang.eecs.umich.edu/projects/powertutor/

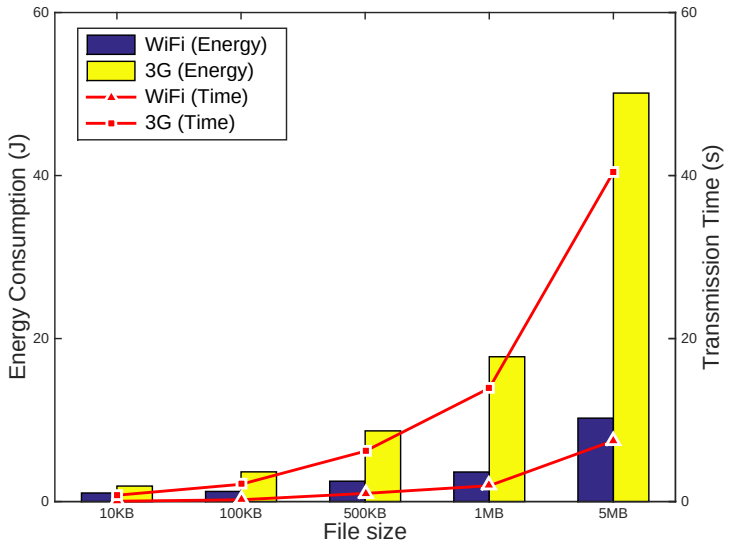


Figure 2.7: The energy consumption and transmission time when using the Xiaomi Redmi 2

device and the application response time, via various derived metrics.

Energy-Response time Weighted Sum (ERWS)

The ERWS metric is addressed by setting the cost function as the weighted sum of the average values:

$$ERWS = \omega \mathbb{E}[\mathcal{E}] + (1 - \omega)\mathbb{E}[T], \tag{2.1}$$

where $\mathbb{E}$ is used for expectation of a random variable, $\mathbb{E}[T]$ and $\mathbb{E}[\mathcal{E}]$ are mean response time and mean energy consumption, respectively, and ω (ranging between 0 and 1) is a weighting parameter that represents the relative significance of energy consumption and response time for the mobile device. To focus on performance, ω should be less than 0.5; to focus on power consumption, ω should be greater than 0.5. When ω is exactly 0.5, the focus is on both increasing performance and reducing power consumption.

The ERWS metric has the advantage of being analytically well tractable since the expectation is additive over time and thus can be optimized via a Markov decision process (MDP) [37]. From the view of minimization, this metric allows comparing arbitrary offloading policies to the optimal offloading policy in our work. However, it has the disadvantage of a linear combination of two metrics on different scales.

To obtain deeper insights, we compare it to the scheme without offloading:

$$ERWS = \omega \cdot \frac{\mathbb{E}[\mathcal{E}]}{\mathbb{E}[\mathcal{E}_{\text{local}}]} + (1-\omega) \cdot \frac{\mathbb{E}[T]}{\mathbb{E}[T_{\text{local}}]}, \tag{2.2}$$

where $\mathbb{E}[T_{\text{local}}]$ and $\mathbb{E}[\mathcal{E}_{\text{local}}]$ are average local response time and average energy consumption, respectively. If $\mathbb{E}[\mathcal{E}]/\mathbb{E}[\mathcal{E}_{\text{local}}]$ is less than 1, offloading operations will reduce the energy consumption. Similarly, if $\mathbb{E}[T]/\mathbb{E}[T_{\text{local}}]$ is less than 1, offloading operations will improve the application's performance [63]. In some special cases performance can be traded for power consumption and vice versa, and thus the ω parameter can be used to express preferences for different types of applications.

Energy-Response time Product (ERP)

The ERP metric is also widely accepted as a suitable metric to capture energy-performance tradeoffs, which is defined as:

$$ERP = \mathbb{E}[\mathcal{E}] \cdot \mathbb{E}[T]. \tag{2.3}$$

Minimizing the ERP metric can be seen as maximizing the 'performance-per-joule', with performance being defined as jobs per time unit [37].

The ERP metric does not suffer from comparison of different scales. While the ERWS metric implies that a reduction in mean response time from 1000 to 999 second is of the same value as a reduction from 2 to 1 second the ERP implies that a reduction in mean response time from 2 to 1 second is much better than a reduction from 1000 to 999 second, which is indeed a more realistic view on the actual system [37]. However, since ERP is the product of two mean values, it is a difficult metric to address analytically.

Energy-Response time Weighted Product (ERWP)

To overcome the disadvantages as aforementioned, we propose a new metric named ERWP, which combines the strengths of ERWS and ERP. It is defined as:

$$ERWP = \mathbb{E}[\mathcal{E}]^{\omega} \cdot \mathbb{E}[T]^{1-\omega}. \tag{2.4}$$

We can rewrite (2.4) as: $ERWP = e^{\omega \cdot \ln(\mathbb{E}[\mathcal{E}]) + (1-w) \cdot \ln(\mathbb{E}[T])}$, which inherits the characteristics of the ERWS metric that assigns different importance weights to energy consumption and response time, and has the advantage of being analytically tractable since the logarithmic expectation is additive over time. Meanwhile, the mean energy consumption and mean response time have equal

importance when $\omega = 0.5$, also in (2.4): $ERWP = \sqrt{\mathbb{E}[\mathcal{E}] * \mathbb{E}[T]}$, which indicates that the ERWP metric has the advantages of the ERP metric that is insensitive to difference of scales.

To the best of our knowledge, the ERWP metric has not been treated analytically before. We obtain tight optimality results by deriving explicit expressions in mobile cloud offloading systems to capture energy-performance tradeoffs.

2.2.3 Partitioning

Application partitioning plays a critical role in high-performance offloading systems. It involves splitting the execution of the application between the mobile side and the cloud side so that the total execution cost is minimized [91]. Through partitioning, a mobile device can benefit most from offloading. Applications can be partitioned statically during development or dynamically during execution.

- *Static Partitioning*: determined beforehand which parts of the application should run locally and which parts should be offloaded, depending on contextual parameters, such as computational intensity of each module, size of data and state to exchange, battery level, and delay constraints [26]. Optimal partitioning for offloading is calculated based on the estimation of communication and computational costs before the program execution. Static partitioning applies to a fixed number of partitions. It has the advantage of requiring a low overhead during execution, but it works well only if the parameters related to the offloading decisions are accurately known in advance or predicted well [25].
- *Dynamic Partitioning*: the requirement of resources for a task may change in its input data and the user-defined goals (e.g. response time, battery consumption). Also, the availability of resources may change at the service nodes (available CPU power, memory, file cache, etc.) and at the wireless network (bandwidth, network latency, etc.) [84]. Thus, optimal partitioning decisions must be made dynamically at runtime to adapt to different operating conditions. Given the variability of the wireless channel, dynamic partitioning seems more appropriate, but it has an associated higher signaling overhead, which must be taken under control. A dynamic approach to computation offloading consists of establishing a rule to decide which parts of the computations may be advantageously offloaded, depending on channel conditions, server state, delay constraints, and so on [26].

2.2.4 Offloading Decision Making

Benefits obtained from offloading greatly depend on whether it is applied at the right time in the right way. Offloading decisions can be made in different ways (e.g. *what*, *where*, *how* and *when*).

What to offload?

Except for typical applications (e.g. chess games, face recognition) that are suitable for offloading, there are many computation tasks that are hard to be classified as “suitable for offloading" or “suitable for local processing”. Offloading decisions should be made to determine what kinds of applications are suitable for offloading. A mobile application can be decomposed into a set of fine-grained tasks, however it is not always necessary or possible to offload all components to the cloud mainly due to the high communication delay that may be generated or because some tasks must access local features (sensors, user interface, etc.). Among the sets of partitions offered by the partitioning result, we should judiciously determine what portion of an application is worth offloading to the cloud and what should be executed locally. A partial offloading strategy selects a subset of tasks to be offloaded, considering the balance between how much the offloading saves and how much extra cost is incurred.

Where to offload?

Applications can deploy their components on multiple application processing nodes such as mobile device, cloudlet and cloud, i.e. there could be multiple offloading destinations and targets [120]. Based on the computational requirements and constraints, offloading decisions should be made on where to offload:

1) Computation is performed locally on the mobile device.
2) Computation is performed on a nearby cloudlet with data transferred between the mobile device and the cloudlet, e.g. via Bluetooth.
3) Computation is performed on a remote cloud server with data transferred between a mobile device and the cloud, e.g. via a cellular network.

How to offload?

Mobile cloud offloading migrates heavy computation from mobile devices to powerful cloud servers using one or more of possibly several available communication networks. There are several ways to offload tasks to a dedicated resource, either using a cellular connection (e.g. 2G, 3G or LTE) or via an intermittently available WLAN hotspot [50]. The wireless bandwidth plays an important role in

the offloading decision process. A weak wireless link can slow down the communication and raise power consumption. Therefore, how to offload tasks through different wireless channels to achieve an overall optimal objective is worth studying.

When to offload?

Offloading is not an always advantageous strategy in that under some circumstances the overhead in time and energy may turn out to be greater than the offloading saving. One may consider sometimes executing a program locally and to offload only when available networks seem favorable to the desired metric. Offloading is worthwhile when the estimated execution time of the task on the mobile device is greater than the sum of its estimated execution time on the cloud sever plus the predicted costs of transferring the related data. Offloading is beneficial when large amounts of computation are needed with relatively small amounts of communication. Therefore, we should determine when offloading a computational task to the dedicated server is optimal and when, on the contrary, local execution is more advisable.

2.3 Related Work

The issues of time and energy saving on mobile devices are becoming increasingly critical. For ease of reference, all related works are summarized in Table 2.3.

2.3.1 Time Saving

Offloading becomes an attractive solution for meeting response time requirements on mobile systems as applications become increasingly complex [60]. Many research efforts have been devoted to offloading computation to remote servers in order to shorten execution time.

The offloading inference engine proposed in [44] can adaptively make decisions at runtime, dynamically partition an application and offload part of the application execution to a powerful nearby surrogate. The partitioning algorithm introduced in [144] aims at reducing the response time of tasks on mobile devices. It finds the offloading and integrating points on a sequence of calls by depth-first search and a linear time searching scheme, and can achieve low user-perceived latency while largely reduce the partitioning computation on cloud. Some application partitioning solutions [40,44,59,80] heavily depend upon programmers and middleware to partition the applications, which limits their uses.

Satyanarayanan *et al*. [105] proposed a virtual machine (VM)-based cloudlet in mobile comput-

Table 2.3: Comparison of Current Offloading Works

Year	Paper	Contribution	Decision	Partition	Time Saving	Energy Saving	Time &Energy Saving
2001	[68]	Partition scheme	Static	√		√	
2002	[80]	Use a distributed platform to offload program	Static	√	√		
2004	[44]	Adaptive offloading for pervasive computing	Dynamic	√	√		
2007	[92]	Performance analysis of offloading systems	Dynamic		√		
2007	[104]	Context-sensitive energy-efficient offloading	Static				√
2008	[118]	Use bandwidth to make offloading decision	Dynamic		√		
2009	[40]	Enable mobile phones as interfaces to cloud	Static		√		
2009	[105]	Cloudlet-based resource-rich mobile computing	Dynamic		√		
2010	[21]	Dynamic partition between devices and clouds	Dynamic	√			√
2010	[23]	Fine grained energy-aware code offload	Dynamic	√		√	
2010	[61]	Study energy tradeoffs	Static			√	
2010	[103]	Stable and adaptive link selection algorithm	Dynamic				√
2011	[20]	Elastic execution between devices and clouds	Static	√			√
2012	[58]	Dynamic resource allocation and parallel execution	Dynamic				√
2012	[59]	Mobile augmentation cloud services (MACS)	Dynamic	√	√		
2012	[48]	Present a dynamic offloading algorithm	Dynamic	√		√	
2012	[144]	Propose an efficient code partition algorithm	Static	√	√		
2013	[86]	Efficient multisite offloading (EMSO) algorithm	Dynamic	√		√	
2013	[107]	Energy-efficient data transmission strategy	Dynamic				√
2013	[70]	Develop an offloading framework (TDM)	Dynamic				√
2013	[12]	Feasibility of mobile cloud systems in a real setting	Dynamic				√
2013	[30]	Measure the energy consumption	Dynamic			√	
2013	[36]	Power control for a multi-tier wireless network	Dynamic			√	
2013	[143]	Energy-efficient scheduling policy for offloading	Dynamic	√		√	
2014	[85]	Partition scheme taking the bandwidth as a variable	Dynamic	√			√
2014	[69]	Energy and performance-aware task scheduling	Dynamic	√			√
2014	[78]	A queueing analytic model for delayed offloading	Dynamic	√			√
2015	[50]	A stochastic model for dynamic offloading	Dynamic		√		
2015	[101]	An energy-aware computation offloading system	Dynamic	√		√	

ing, to which a smartphone connects over a WLAN, with the argument against the use of the cloud due to higher latency and lower bandwidth available when connecting. In essence, cloudlets make use of smartphones simply as a thin-client to access local resources, rather than using the smartphone's capabilities directly and offloading only when required.

A stochastic model for dynamic offloading has been developed in [50] using various performance metrics and also intermittently available access links. The mobile nature of mobile devices and the unstable connectivity of wireless links affect the predictability of the performance of a pervasive service running under the control of offloading systems. Ou *et al*. [92] analyzed the performance of offloading systems in mobile wireless environments when considering system failure and recovery. However, they did not consider how to make offloading decisions. Further, a framework using estimated bandwidth to make offloading decisions was investigated in [118], which formulated decision

problems for computational offloading systems according to the bandwidth prediction between the local and remote systems. It assumed that the network reliability was not an issue, while in a realistic scenario, the network may even not be available due to mobility or other reasons.

2.3.2 Energy Saving

Extending battery lifetime is one of the most crucial design objectives of mobile devices because they are usually equipped with limited battery capacity.

Partitioning technologies were adopted to identify offloaded parts for energy saving [23, 48, 68]. The energy cost of each function of the application was profiled. According to the profiling result, they constructed a cost graph, in which each node represented a function to be performed, and each edge indicated the data to be transmitted. Finally, the server parts were executed on remote servers for reducing energy consumption.

MAUI [23] is a system that enables energy-aware offloading of mobile code to the infrastructure. Its main aim is to optimize energy consumption of a mobile device, by estimating and trading off the energy consumed by local processing vs. transmission of code and data for remote execution. It decides at runtime which methods should be remotely executed, and achieves the best energy savings possible under the mobile device's current connectivity constraints.

Jade [97–102] is a system that adds sophisticated energy-aware computation offloading capabilities to Android applications. It monitors device and application status by adapting to workload variation, communication costs, and energy status, and then automatically decides where code should be executed. Jade can effectively reduce up to 39% of average power consumption for mobile application while improving application performance.

Some works like [30, 61, 143] built energy models to approximate the energy consumption of offloading. The energy models can be used to construct the aforementioned cost graph or make offloading decisions. However, they did not provide an effective method to obtain optimal offloading decisions.

Karthik *et al*. [61] argued that offloading could potentially save energy for mobile users, but not all applications were energy-efficient when migrated to the cloud. It depends on whether the computational cost saved due to offloading outperforms the extra communication cost. A large amount of communication combined with a small amount of computation should preferably be performed locally on the mobile device, while a small amount of communication with a large amount of computation should preferably be executed remotely.

Energy consumption in mobile devices has become an important issue for network selection. Gribaudo *et al*. [42] developed a framework based on the Markovian agent formalism, which could

model the dynamics of user traffic and the allocation of the network radio resources. A power control scheme suitable for a multi-tier wireless network was presented in [36]. It maximized the energy-efficiency of a mobile device transmitting on several communication channels while at the same time ensuring the required minimum quality of service (QoS).

Some works consider a response time constraint for the application when partitioning application tasks for execution on mobile devices and cloud servers. This deadline is an important issue for many interactive applications [122]. To achieve energy saving while satisfying given application deadline, dynamic offloading algorithms were presented in [48, 143]. They showed low complexity to solve the offloading decision making problem (i.e. to determine which software components to execute remotely under mobile network environments).

2.3.3 Time and Energy Combined Saving

Both time and energy saving are crucial for mobile devices. Many research efforts have been devoted to optimizing the two objectives simultaneously.

CloneCloud [20] used a combination of static analysis and dynamic profiling to partition applications automatically at a fine granularity while optimizing execution time and energy usage for a target computation and communication environment. However, this approach only considers limited input/environmental conditions in the offline pre-processing and needs to be bootstrapped for every new application built. Dynamic partitioning of applications between weak devices and clouds was presented in [21, 85], to better support applications running on diverse devices in different environments. They addressed how dynamic partitioning can address these heterogeneity problems by taking the bandwidth as a variable.

ThinkAir [58] exploited the concept of smartphone virtualization in the cloud and provided method-level computation offloading. Advancing on previous work, it focused on the elasticity and scalability of the cloud and enhanced the power of MCC by parallelizing method execution using multiple VM images.

Beraldi *et al*. [12] showed that rather than always offloading the whole application remotely, running partial components locally can be more advantageous. They proposed a novel generic architecture that can be integrated in any MCC application in order to automate the offloading decision and improve the application response time while minimizing the overall energy consumed by the mobile device.

Recently, several groups have worked on optimizing the tradeoff between the energy consumption and response time. Rahmati *et al*. [104] suggested seamless offloading operation by switching between several transmission technologies, and examined the tradeoff between energy consumption

for WiFi search and transmission efficiency when the WiFi network was intermittently available. Energy-efficient delayed network selection has been suggested in [103, 107] to optimize the tradeoff between energy usage and delay in data transmission by intentionally deferring data transmission until the device meets an energy-efficient network. Researchers have further suggested the use of "delayed offloading": if no WiFi connection is available, (some) traffic can be delayed up to a chosen deadline, or until WiFi becomes available [78].

Chapter 3

Offloading Decision Making: What to Offload

Since offloading all computation components of an application to the remote cloud is not always necessary or effective, we should judiciously determine which part of the application should be deployed on the cloud server and what should be left on the mobile device to achieve a particular performance target (low response time and/or energy consumption, etc.).

Calculations can naturally be described as graphs in which vertices represent computational costs and edges reflect communication costs [45]. By partitioning the vertices of a graph, the calculation can be divided among processors of local mobile devices and remote cloud servers. Graph partitioning algorithms traditionally applied in the context of mobile applications (e.g. [4, 15, 109, 139]) cannot be applied directly to the mobile offloading systems, because they only consider the weights on the edges of the graph, neglecting the weight of each node. Our research is situated in the context of resource-constrained mobile devices, in which there are often multi-objective cost functions, such as minimizing the total response time or energy consumption by offloading partial workloads to a cloud server. We explore the methods of how to deploy application tasks in a more optimal way, by dynamically and automatically determining which portion of the application should be offloaded to the cloud and what should be performed on the mobile device.

3.1 Partitioning Problems

Application partitioning is very important for designing an adaptive, cost-effective, and efficient offloading system. Some critical issues concerning the partitioning problem include:

- *Weighting*: when choosing an application task to offload, we need to scale the weights of each task regarding its resource utilization, such as memory, processing time, and bandwidth utilization [91]. The weights can vary for different mobile devices and in different running environments. Communication overhead is introduced by the remote communication between a mobile device and a cloud server.
- *Real-Time Adaptability*: since available network bandwidth varies in wireless environments, static partitioning algorithms proposed by previous works with a fixed bandwidth assumption are unsuitable for mobile platforms [66]. The partitioning algorithms should be adaptive to network and device changes. For example, an optimal partition for a high-bandwidth network and low-capacity client might not be a good partition for a high-capacity client with a bad network connection. Since the network condition is only measurable at runtime, the partitioning algorithm should be a real-time online process [144].
- *Partitioning Efficiency*: making partitioning decisions for simple applications (e.g. an alarm clock) in real-time is not difficult, but for some complex applications (e.g. speech/face recognition) that contain a large number of methods [144], a highly efficient algorithm is required to perform real-time partitioning.

3.1.1 Partitioning Process

To solve the above challenges, the workflow of an environment-adaptive application partitioning process is proposed in Fig. 3.1. It starts with profiling an application that can be split into multiple tasks, through static analysis and dynamic profiling technology [85]. We then construct a weighted consumption (refer to response time or energy consumption) graph of the mobile application as shown in Fig. 3.3(b). Based on partitioning cost models, an elastic partitioning algorithm is proposed to make a proper application partitioning. By calling such an algorithm, we can get preliminary partitioning results for response time or energy optimization. During the application execution process, if a mobile environment changes, and these changes meet or exceed a certain threshold, the application graph will be re-partitioned according to the new parameters. Therefore, it can ultimately realize the condition-aware and environment-adaptive elastic partitioning. Here in the context of the mobile environment, it includes mobile computing resources inside the device, such as battery level, CPU speed and memory, but also involves an external mobile environment, such as network connection status and speed of the cloud server. After partitioning, it then automatically offloads the distributed application components that require remote execution to the cloud server and performs the rest locally on the mobile device according to the partitioning results.

The problem of whether or not to offload certain parts of an application to the cloud depends on

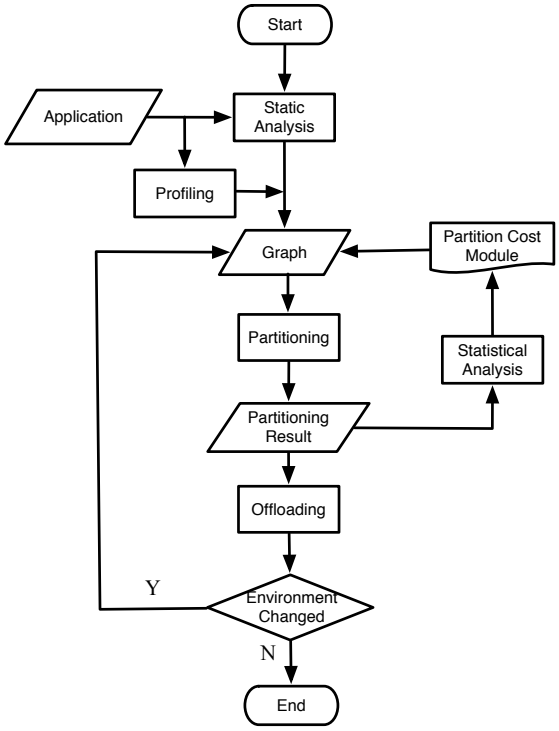


Figure 3.1: Flowchart of an application partitioning process

the following factors: CPU speed of the mobile device, network bandwidth, transmission data size, and the speed of the cloud server [74]. When considering such factors, we construct a weighted consumption graph according to the estimated computational and communication cost, and further derive a new partitioning algorithm designed especially for mobile offloading systems.

3.1.2 Classification of Application Tasks

Different applications emerge in a mobile device according to some process and each consists of several tasks. Since not all the application tasks are suitable for remote execution, they need to be weighted and distinguished as:

- **Unoffloadable Tasks**: some should be unconditionally executed locally on the mobile device, either because transferring relevant information would take tremendous time and energy or because these tasks must access local components (e.g. camera, GPS, user interface, accelerometer or other sensors) [23]. Tasks that might cause security issues when executed on a different place should also not be offloaded (such as e-commerce). Local processing consumes the battery power of the device, but there are no communication costs or delays.
- **Offloadable Tasks**: some application components are flexible tasks that can be processed ei-

ther locally on the processor of the mobile device, or remotely in a cloud infrastructure. Many tasks fall into this category, and the offloading decision depends on whether the communication costs outweigh the difference between local and remote costs or not [60].

We do not need to take offloading decisions for unoffloadable components. However, as for offloadable ones, since offloading all the application tasks to the remote cloud is not necessary or effective under all circumstances, it is worth considering what should be executed locally on the mobile device and what should be offloaded onto the remote cloud for execution based on available networks, response time or energy consumption. The mobile device has to take an offloading decision based on the result of a dynamic optimization problem.

3.2 Partitioning Models

In this section, we will illustrate which assumptions are made, how weighted consumption graphs for different types of applications are constructed and how the optimization problem is defined.

3.2.1 Classification of Topologies

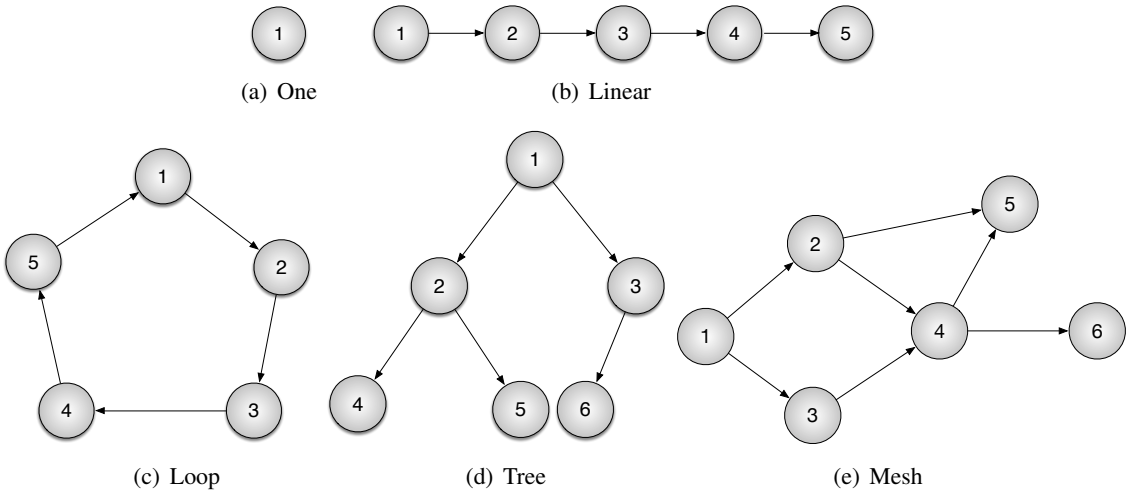
The granularity levels for partitioning computational-intensive mobile application are not limited to a specific form [73]. Previous work considers application partitioning at different levels of granularity: classes [2], objects [85], methods [23], components [112, 139], and threads [20]. Without loss of generality, we refer to application tasks in this thesis. Application developers can choose the appropriate partition granularity according to different applications.

Construction of weighted consumption graphs is critical for application partitioning. A mobile application can be represented as a list of fine-grained tasks, formulating different topologies as depicted in Fig. 3.2, where each node reflects an application task, executed either at the mobile side or offloaded onto the cloud side for further execution.

(a) *Only one active node*: representing an entire application (without partitioning). Such a topology is often adopted by previous full offloading schemes [20, 105, 130], which can also be viewed as an example of software as a service. In this case, the whole application is migrated to a remote server involving complete transfer of code and program state to the server [93]. The main drawback of this solution includes inflexibility and coarse granularity.

(b) *Linear topology*: representing a sequential list of fine-grained tasks. Each task is sequentially executed, with the output data generated by one task as the input of next one [52].

(c) *Loop-based topology*: a loop-based application is one in which most of the functionality is given by iterating an execution loop, such as all the online social applications, in which we



(a) One (b) Linear

(c) Loop (d) Tree (e) Mesh

Figure 3.2: Task-flow graphs in different topologies

model their processing with a graph that consists of a cycle [89].

(d) *Tree-based topology*: representing a tree-based hierarchy of tasks [93]. The node at the top of the tree is the application entry node (i.e. the main module).

(e) *Mesh-based topology*: representing a lattice-based topology of tasks, e.g. a Java example of face recognition as depicted in [85].

When compared with the scheme that offloads the whole application (i.e. Fig. 3.2(a)) into the cloud, an application partitioning scheme is able to achieve a fine granularity for computation offloading when partitioning a topological consumption graph between local and remote executions. Different partitions can lead to different costs, and the total cost incurred due to offloading depends on multiple factors, such as device platforms, networks, clouds, and workloads. Therefore, the application may have different optimal partitions for different mobile environments and workloads.

3.2.2 Construction of Weighted Consumption Graph

There are two types of costs in the offloading systems: one is computational cost for running the application tasks locally or remotely (including memory cost, processing time cost and so on) and the other is communication cost for the application tasks' interaction (associated with movement of data and requisite messages). Even the same task can have different costs on the mobile device and the cloud in terms of execution time and energy consumption. As cloud servers usually execute much faster than mobile devices having a powerful configuration, it can save energy and improve performance when offloading part of the computation to cloud servers [86]. However, when vertices

are assigned to different sides, the interaction between them leads to extra communication costs. Therefore, we try to find the optimal assignment of vertices for graph partitioning and computation offloading by trading off the computational costs with the communication costs.

Call graphs are widely used to describe data dependencies within a computation, where each vertex represents a task and each edge represents the calling relationship from the caller to the callee. Figure 3.3(a) shows a consumption graph example consisting of six tasks [40]. The computational costs are represented by vertices, while the communication costs are expressed by edges. We denote the dependency of an application's tasks and their corresponding costs as a directed acyclic graph $G = (V, E)$, where the set of vertices $V = (v_1, v_2, \cdots, v_N)$ denotes N application tasks and an edge $e(v_i, v_j) \in E$ represents the frequency of invocation and data access between nodes v_i and v_j, where vertices v_i and v_j are neighbors. Each task v_i is characterized by five parameters:

- *type*: offloadable or unoffloadable task.
- m_i: the memory consumption of node v_i on a mobile device platform,
- c_i: the size of the compiled code of node v_i,
- in_{ij}: the data size of input from node v_i to node v_j,
- out_{ji}: the data size of output from node v_j to node v_i.

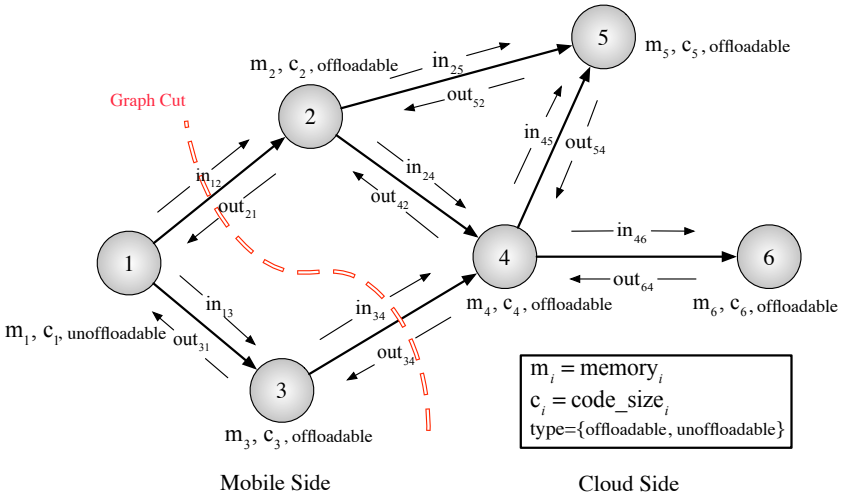
We further construct a weighted consumption graph as depicted in Fig. 3.3(b). Each vertex $v \in V$ is annotated with two-cost weights: $< w^{\text{local}}(v), w^{\text{cloud}}(v) >$, where $w^{\text{local}}(v)$ and $w^{\text{cloud}}(v)$ represent the computational cost of executing the task v locally on the mobile device and remotely on the cloud, respectively. Each vertex is assigned one of the weights depending on the partitioning result of the application graph it finally ends up in or the label it is assigned [108]. The edge set $E \subset V \times V$ represents the communication cost amongst tasks. The weight of an edge $w(e(v_i, v_j))$ is denoted as:

$$w(e(v_i, v_j)) = \frac{in_{ij}}{B_{\text{upload}}} + \frac{out_{ij}}{B_{\text{download}}}, \tag{3.1}$$

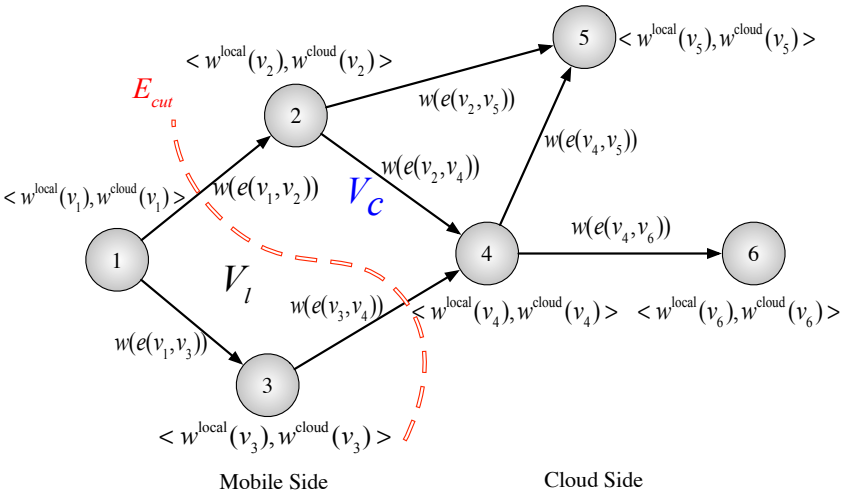
which is the communication cost of transferring the input and return states when the tasks v_i and v_j are executed on different sides, and it closely depend on the network bandwidths (upload bandwidth B_{upload} and download bandwidth B_{download}) and the transferred data.

A candidate offloading decision is described by one cut in the weighted consumption graph, which separates the vertices into two disjoint sets, one representing tasks that are executed on the mobile device and the other implying tasks that are offloaded to the remote server [54]. Hence, taking the optimal offloading decision is equivalent to partitioning the weighted consumption graph such that the cost evaluation function is minimized [113].

The red dotted line in Fig. 3.3(b) is one possible partitioning cut, indicating the partitioning of



(a) Consumption graph



(b) Weighted consumption graph

Figure 3.3: Construction of consumption graph and weighted consumption graph.

computational workload in the application between the mobile device and the cloud. V_l and V_c are sets of vertices, where V_l is the local set in which tasks are executed locally and V_c is the cloud set in which tasks are directly offloaded to the cloud. We have $V_l \cap V_c = \emptyset$ and $V_l \cup V_c = V$. Further, E_{cut} is the edge set in which the graph is cut into two parts.

3.2.3 Cost Models

Mobile application partitioning aims at finding the optimal partitioning solution that leads to the minimum execution cost, in order to make the best tradeoff between time/energy savings and transmission costs/delay.

The optimal partitioning decision depends on user requirements/expectations, device information, network bandwidth, and the application itself. Device information includes the execution speed of the device and the workloads on it when the application is launched. If the device computes very slowly and the aim is to reduce execution time, it is better to offload more computation to the cloud [138]. Network bandwidth affects data transmission for remote execution. If the bandwidth is very high, the cost in terms of data transmission will be low. In this case, it is better to offload more computation to the cloud.

The partitioning decision is made based on the cost estimation (computational and communication costs) before the program execution. On the basis of Fig. 3.3(b), we have:

$$C_{\text{total}} = \underbrace{\sum_{v \in V} I_v \cdot w^{\text{local}}(v)}_{\text{local}} + \underbrace{\sum_{v \in V} (1 - I_v) \cdot w^{\text{cloud}}(v)}_{\text{remote}} + \underbrace{\sum_{e(v_i, v_j) \in E} I_e \cdot w(e(v_i, v_j))}_{\text{communication}}, \tag{3.2}$$

where the total cost is the sum of computational costs (local and remote) and communication costs of cut affected edges.

The cloud server node and the mobile device node must belong to different partitions. One possible solution for this partitioning problem will give us an arbitrary tuple of partitions from the vertices set $< V_l, V_c >$ and the cut of edge set E_{cut} in the following way:

$$I_v = \begin{cases} 1, & \text{if } v \in V_l \\ 0, & \text{if } v \in V_c \end{cases} \quad \text{and} \quad I_e = \begin{cases} 1, & \text{if } e \in E_{\text{cut}} \\ 0, & \text{if } e \notin E_{\text{cut}} \end{cases}. \tag{3.3}$$

We seek to find an optimal cut in the weighted consumption graph such that some application tasks are executed on the mobile side and the remaining ones on the cloud side. The optimal cut maximizes or minimizes an objective function and meanwhile satisfies a mobile device's resource constraints. The objective function expresses the general goal of a partition, this may be, for instance, minimize the energy consumption, minimize the amount of exchanged data, or complete the execution in less than a predefined time. We only actually perform partitioning when it is beneficial. Not all applications can benefit from partitioning because of application-specific properties. The cost estimation of running each application task on the mobile device and cloud server is needed.

Offloading makes sense only if the speedup of the cloud server outweighs the extra communication cost.

The communication time and energy costs for the mobile device will vary according to the amount of data to be transferred and the wireless network conditions. According to (3.2), the dynamic execution configuration of an elastic application can be decided based on some different saving objectives with respect to response time and energy consumption. A task's offloading goals may change due to a change in environmental conditions.

Minimum Response Time

The communication cost depends on the size of data transfer and the network bandwidth, while the computational cost depends on the computation time. If the minimum response time is selected as the objective function, we can calculate the total time spent due to offloading as:

$$T_{\text{total}}(\boldsymbol{I}) = \underbrace{\sum_{v \in V} I_v \cdot T_v^l}_{\text{local}} + \underbrace{\sum_{v \in V} (1 - I_v) \cdot T_v^c}_{\text{remote}} + \underbrace{\sum_{e \in E} I_e \cdot T_e^{tr}}_{\text{communication}}, \tag{3.4}$$

where $T_v^l = F \cdot T_v^c$ is the computation time of task v on the mobile device when it is executed locally; F is the speedup factor, the ratio of the cloud server's execution speed compared to that of the mobile device, since the computation capacity of cloud infrastructure is stronger than that of the mobile device, we have $F > 1$; T_v^c is the computation time of task v on the cloud server when it is offloaded; $T_e^{tr} = D_e^{tr}/B$ is the communication time between the mobile device and the cloud; D_e^{tr} is the amount of data that is transmitted and received; B is the current wireless bandwidth.

In this scenario, the offloading decision engine then selects the best partitioning candidate that minimizes the total response time. The aim of this cost model is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \left\{ I_v, I_e | I_v, I_e \in \{0, 1\} \right\}$, which satisfies $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} T_{\text{total}}(\boldsymbol{I})$. For a given application and a mobile device, the optimal partitioning result also changes according to different wireless network bandwidth and speedup factor of the cloud server.

The saved response time in the partitioning scheme compared to the scheme without offloading is calculated as:

$$T_{\text{save}}(\boldsymbol{I}) = \frac{T_{\text{local}} - T_{\text{total}}(\boldsymbol{I})}{T_{\text{local}}} \cdot 100\%, \tag{3.5}$$

where $T_{\text{local}} = \sum_{v \in V} T_v^l$ is the local time cost when all the application tasks are executed locally on the mobile device.

Minimum Energy Consumption

If the minimum energy consumption is chosen as the objective function, we can calculate the total energy consumed due to offloading as:

$$E_{\text{total}}(\boldsymbol{I}) = \underbrace{\sum_{v \in V} I_v \cdot E_v^l}_{\text{local}} + \underbrace{\sum_{v \in V} (1 - I_v) \cdot E_v^i}_{\text{idle}} + \underbrace{\sum_{e \in E} I_e \cdot E_e^{tr}}_{\text{communication}}, \tag{3.6}$$

where $E_v^l = p_m \cdot T_v^l$ is the energy consumed of task v on the mobile device when it is executed locally; $E_v^i = p_i \cdot T_v^c$ is the energy consumed of task v on the mobile device when it is offloaded to the cloud; $E_e = p_{tr} \cdot T_e^{tr}$ is the energy spent on the communication between the mobile device and the cloud; p_m, p_i and p_{tr} are the powers of the mobile device for computing, while being idle and for data transfer, respectively.

In this scenario, the offloading decision engine then selects the best partitioning plan that minimizes the partitioning cost of energy. The aim is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \left\{ I_v, I_e | I_v, I_e \in \{0, 1\} \right\}$, which satisfies: $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} \ E_{\text{total}}(\boldsymbol{I})$.

The saved energy when compared to the scheme without offloading is:

$$E_{\text{save}}(\boldsymbol{I}) = \frac{E_{\text{local}} - E_{\text{total}}(\boldsymbol{I})}{E_{\text{local}}} \cdot 100\%, \tag{3.7}$$

where $E_{\text{local}} = \sum_{v \in V} E_v^l$ is the local energy cost when all tasks are executed locally.

Minimum of the Weighted Sum of Time and Energy

If we combine both the response time and energy consumption, by using the ERWS metric in (2.2), we have the cost model for partitioning as follows:

$$W_{\text{total}}(\boldsymbol{I}) = \omega \cdot \frac{T_{\text{total}}(\boldsymbol{I})}{T_{\text{local}}} + (1 - \omega) \cdot \frac{E_{\text{total}}(\boldsymbol{I})}{E_{\text{local}}}, \tag{3.8}$$

and the saved weighted sum of time and energy in the partitioning scheme compared to the scheme without offloading is calculated as:

$$W_{\text{save}}(\boldsymbol{I}) = \left[\omega \cdot \frac{T_{\text{local}} - T_{\text{total}}(\boldsymbol{I})}{T_{\text{local}}} + (1 - \omega) \cdot \frac{E_{\text{local}} - E_{\text{total}}(\boldsymbol{I})}{E_{\text{local}}} \right] \cdot 100\%, \tag{3.9}$$

where $T_{\text{total}}(\boldsymbol{I})$ and $E_{\text{total}}(\boldsymbol{I})$ are the response time and energy consumption with a partitioning solution $\boldsymbol{I}$, respectively. Since we have already normalization, the units of energy and time chosen will not affect the tradeoff.

In this scenario, we choose the best partitioning plan that minimizes the partitioning cost of the weighted sum of time and energy. Its aim is to find the optimal application partitioning: $\boldsymbol{I}_{\text{min}} = \left\{ I_v, I_e | I_v, I_e \in \{0,1\} \right\}$, while satisfying: $\boldsymbol{I}_{\text{min}} = \arg\min_{\boldsymbol{I}} \ W_{\text{total}}(\boldsymbol{I})$.

3.3 Partitioning Algorithm for Offloading

In this section, we propose a new partitioning algorithm for arbitrary topology. It takes a weighted consumption graph as input which represents an application's operations/calculations as the nodes and the communication between them as the edges. Each node has two costs: first is the cost of performing the operation locally (e.g. on the mobile phone) and second is the cost of performing it elsewhere (e.g. on the cloud). The weight of the edges is the communication cost to the offloaded computation. It is assumed that the communication cost between operations in the same location are negligible. The result contains information about the cost and reports which operations should be performed locally and which should be offloaded.

3.3.1 Steps

The partitoning algorithm can be divided into two steps as follows:

1) *Unoffloadable Vertices Merging*: An unoffloadable vertex is the one that has special features making it unable to be migrated outside of the mobile device and thus is located only in the unoffloadable partition. Apart from this, we can choose any task to be executed locally according to our preferences or other reasons. Then all vertices that are not going to be migrated to the cloud are merged into one that is selected as the source vertex. By 'merging', we mean that these nodes are coalesced into one, whose weight is the sum of the weights of all merged nodes. Let G represent the original graph after all the unoffloadable vertices are merged.
2) *Coarse Partitioning*: The target of this step is to coarsen G to the coarsest graph $G_{|V|}$. To coarsen means to merge two nodes and reduce the node count by one. Therefore, the algorithm has $|V| - 1$ phases. In each phase i (for $1 \leq i \leq |V| - 1$), the cut value, i.e. the partitioning cost in a graph $G_i = (V_i, E_i)$ is calculated. G_{i+1} arises from G_i by merging "suitable nodes", where $G_1 = G$. The partitioning results are the minimum cut among all the cuts in an individual phase i and the corresponding group lists for local and cloud execution.

Furthermore, in each phase i of the coarse partitioning, we still have five steps:

(a) Start with A={a}, where a is usually an unoffloadable node in G_i.

(b) Iteratively add the vertex to A that is the most tightly connected to A.

(c) Let s, t be the last two vertices (in order) added to A.

(d) The graph cut of the phase i is $(V_i \backslash \{t\}, \{t\})$.

(e) G_{i+1} arises from G_i by merging vertices s and t.

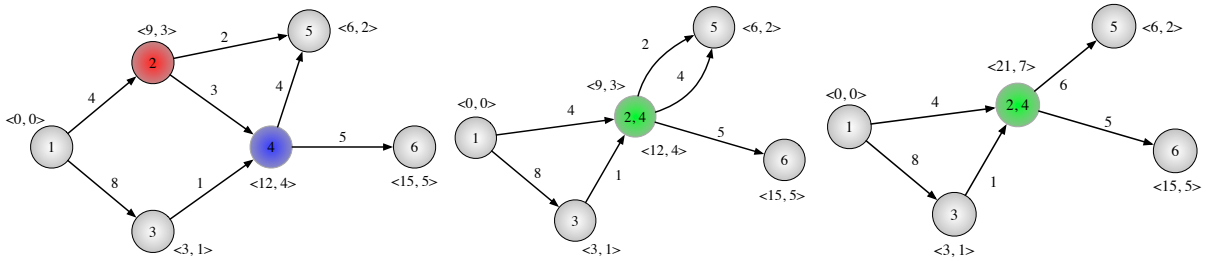
3.3.2 Algorithmic Process

The algorithmic process is illustrated as the *MinCut* function in Algorithm 2, and in each phase i, it calls the *MinCutPhase* function as described in Algorithm 3. Since some tasks have to be executed locally, we need to merge them into one node.

The *merging* function is used to merge two vertices into one new vertex, which is implemented as in Algorithm 1. If nodes $s, t \in V$ ($s \neq t$), then node s and node t can be merged as follows:

1) Nodes s and t are chosen.
2) Nodes s and t are replaced by a new node $x_{s,t}$. All edges that were previously incident to s or t are now incident to $x_{s,t}$ (except the edge between nodes s and t when they are connected).
3) Multiple edges are resolved by adding edge weights. The weights of the node $x_{s,t}$ are resolved by adding the weights of s and t.

For example, we can merge nodes 2 and 4 as shown in Fig. 3.4.



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3.4: An example of merging two nodes

The core of this partitioning algorithm is to make it easy to select the next vertex to be added to the local set A, that is *Most Tightly Connected Vertex (MTCV)*, which is defined as the vertex whose $\Delta(v)$ into A is maximized, when $\Delta(v) = w(e(A, v)) - [w^{\text{local}}(v) - w^{\text{cloud}}(v)]$. In other words, the potential benefit from offloading, i.e. $[w^{\text{local}}(v) - w^{\text{cloud}}(v)] - w(e(A, v))$, is the minimum when v is offloaded to the cloud, thus task v has the most chance to be executed locally.

Algorithm 1 The *Merging* function

//This function takes s and t as vertices in the given graph and merges them into one
Function: G'=$Merge(G, w, s, t)$

Input: G: the given graph, $G = (V, E)$
w: the weights of edges and vertices
s, t: two vertices in previous graph that are to be merged
Output: G': the new graph after merging two vertices

1: $x_{s,t} \Leftarrow s \cup t$
2: **for** all nodes $v \in V$ **do**
3: **if** $v \neq \{s, t\}$ **then**
4: $w(e(x_{s,t}, v)) = w(e(s, v)) + w(e(t, v))$ //adding weights of edges
5: //adding weights of nodes
6: $\left[w^{\text{local}}(x_{s,t}), w^{\text{cloud}}(x_{s,t})\right] = \left[w^{\text{local}}(s) + w^{\text{local}}(t), w^{\text{cloud}}(s) + w^{\text{cloud}}(t)\right]$
7: $E \Leftarrow E \cup e(x_{s,t}, v)$ //adding edges
8: **end if**
9: $E' \Leftarrow E \backslash \{e(s, v), e(t, v)\}$ //deleting edges
10: **end for**
11: $V' \Leftarrow V \backslash \{s, t\} \cup x_{s,t}$
12: **return** $G' = (V', E')$

Algorithm 2 The $MinCut$ function

//This function performs an optimal offloading partitioning algorithm
Function: $[minCut, MinCutGroupsList] = MinCut(G, w, SourceVertices)$

Input: G: the given graph, $G = (V, E)$
w: the weights of edges and vertices
SourceVertices: a list of vertices that are forced to be kept in one side of the cut
Output: *minCut*: the minimum sum of weights of edges and vertices among the cut
MinCutGroupsList: two lists of vertices, one local list and one remote list

1: $w(minCut) \Leftarrow \infty$
2: **for** $i = 1 : length(SourceVertices)$ **do**
3: //Merge all the source vertices (unoffloadable) into one
4: $(G, w) = Merge(G, w, SourceVertices(1), SourceVertices(i))$
5: **end for**
6: **while** $|V| > 1$ **do**
7: $[cut(A - t, t), s, t] = MinCutPhase(G, w)$
8: **if** $w(cut(A - t, t)) < w(minCut)$ **then**
9: $minCut \Leftarrow cut(A - t, t)$
10: **end if**
11: $Merge(G, w, s, t)$ //Merge the last two vertices (in order) into one
12: **end while**
13: **return** $minCut$ and $MinCutGroupsList$

Further, we have the total cost from partitioning:

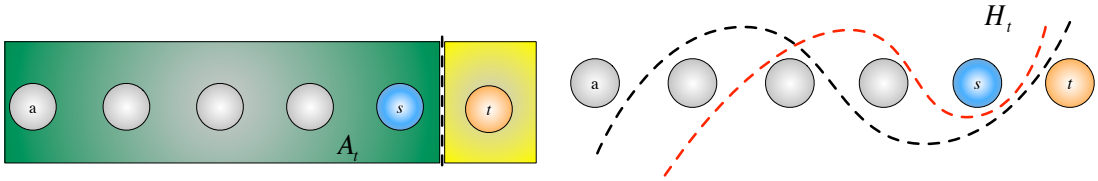
$$C_{cut(A-t,t)} = C^{\text{local}} - \left[w^{\text{local}}(t) - w^{\text{cloud}}(t)\right] + \sum_{v \in A \setminus t} w(e(t,v)), \tag{3.10}$$

where $C^{\text{local}} = \sum_{v \in V} w^{\text{local}}(v)$ is the total of local costs and the cut value $C_{cut(A-t,t)}$ is the partitioning cost, $w^{\text{local}}(t) - w^{\text{cloud}}(t)$ is the gain of node t from offloading, and $\sum_{v \in A \setminus t} w(e(t,v))$ is the total of extra communication costs due to offloading.

Theorem 1. *$cut(A-t,t)$ is always a minimum $s-t$ cut in the current graph, where s and t are the last two vertices added in the phase, the $s-t$ cut separates nodes s and t on two different sides.*

The run of each *MinCutPhase* function orders the vertices of the current graph linearly, starting with a and ending with s and t, according to the order of addition into A. We want to show that $C_{cut(A-t,t)} \leq C_{cut(H)}$ for any arbitrary $s-t$ cut H.

Lemma 1. *We define H as an arbitrary $s-t$ cut, A_v as a set of vertices added to A before v, and H_v as a cut of $A_v \cup \{v\}$ induced by H. For all active vertices v, we have $C_{cut}(A_v, v) \leq C_{cut}(H_v)$.*



(a) The minimum $s-t$ cut (b) An arbitrary $s-t$ cut

Figure 3.5: Illustration for the proof of Lemma 1

Proof. As shown in Fig. 3.5, we use induction on the number of active vertices, k.

1) When $k = 1$, the claim is true,
2) Assume the inequality holds true up to u, i.e. $C_{cut}(A_u, u) \leq C_{cut}(H_u)$,
3) Suppose v is the first active vertex after u, and then we have:

$$\begin{aligned} C_{cut}(A_v, v) &= C_{cut}(A_u, v) + C_{cut}(A_v - A_u, v) \\ &\leq C_{cut}(A_u, u) + C_{cut}(A_v - A_u, v) \quad (u \text{ is } \textit{MTCV}) \\ &\leq C_{cut}(H_u) + C_{cut}(A_v - A_u, v) \quad (C_{cut}(A_u, u) \leq C_{cut}(H_u)) \\ &\leq C_{cut}(H_v). \end{aligned}$$

□

Since t is always an active vertex with respect to H, according to Lemma 1, we can conclude that

Algorithm 3 The *MinCutPhase* function

//This function perform one phase of the partitioning algorithm
Function: $[cut(A-t,t), s, t]$=$MinCutPhase(G_i, w)$

Input: G_i: the graph in Phase i, i.e. $G_i = (V_i, E_i)$
w: the weights of edges and vertices
SourceVertices: a list of vertices that are forced to be kept in one side of the cut

Output: s, t: the lasted two vertices that are added to A
$cut(A-t,t)$: the cut between $\{A-t\}$ and $\{t\}$ in Phase i

$a \Leftarrow$ arbitrary vertex of G_i
$A \Leftarrow \{a\}$
while $A \neq V_i$ **do**
    $max = -\infty$
    $v_{\max} = null$
    for $v \in V_i$ **do**
        if $v \notin A$ **then**
            //Performance gain through offloading the task v to the cloud
            $\Delta(v) \Leftarrow w(e(A,v)) - [w^{\text{local}}(v) - w^{\text{cloud}}(v)]$
            //Find the vertex that is the most tightly connected to A
            if $max < \Delta(v)$ **then**
                $max = \Delta(v)$
                $v_{max} = v$
            end if
        end if
    end for
    $A \Leftarrow A \cup (v_{max})$
    $a \Leftarrow Merge(G, w, a, v_{max})$
end while
$t \Leftarrow$ the last vertex (in order) added to A
$s \Leftarrow$ the last second vertex (in order) added to A
return $cut(A-t,t)$

$C_{cut(A-t,t)} \leq C_{cut(H)}$ which says exactly that the cost of $cut(A-t,t)$ is at most as heavy as the cost of $cut(H)$. This proves Theorem 1.

3.3.3 Computational Complexity

As the running time of the algorithm *MinCut* is essentially equal to the accumulated running time of the $|V|-1$ runs of *MinCutPhase*, which is called on graphs with decreasing number of vertices and edges, it suffices to show that a single *MinCutPhase* needs at most $O(|V|\log|V|+|E|)$ time. The computation complexity of the min-cost offloading partitioning (MCOP) algorithm can be noted as $O(|V|^2\log|V|+|V||E|)$.

As a comparison, a linear-programming (LP) solver is widely used [20, 23]. The LP solver is based on branch and bound, which is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as general real valued problems [117]. The number of its optional solutions grows exponentially with the number of tasks, which means higher time complexity $O(2^{|V|})$.

Therefore, the MCOP algorithm has much lower time complexity when compared to the existing algorithms, which is proportional to the square of the number of tasks and hence can achieve an optimal offloading strategy in minimal time.

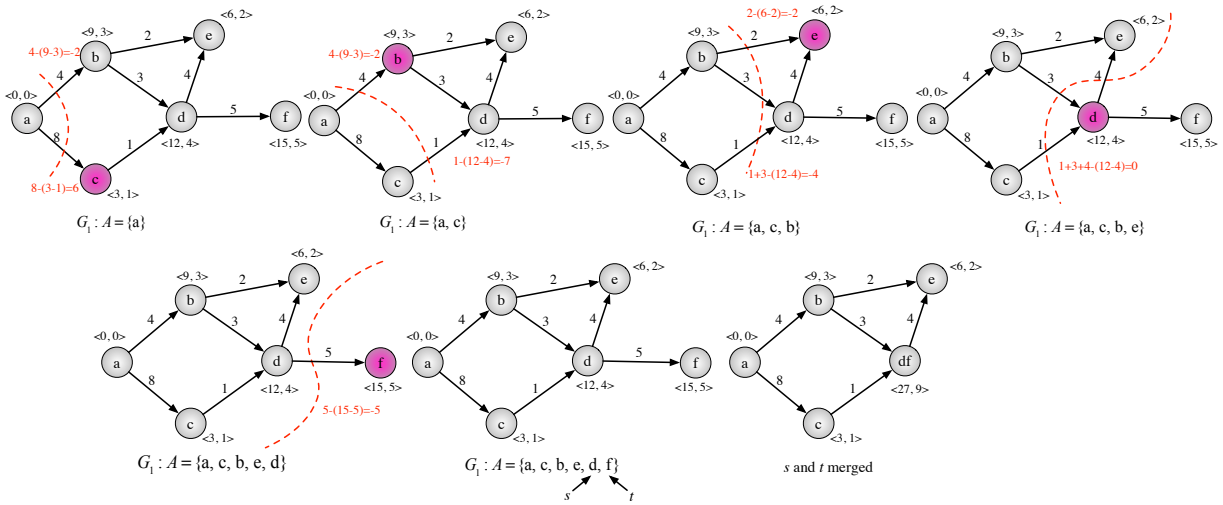


Figure 3.6: The 1st phase of *MinCutPhase* function. The induced ordering a, c, b, e, s, t of the vertices, where s = d and t = f. The 1st cut-of-the-phase corresponds to the partitions {a, c, b, e, d} and {f} with the cut value: $C_{cut(A-\mathrm{f},\mathrm{f})} = 45-(15-5)+5 = 40$.

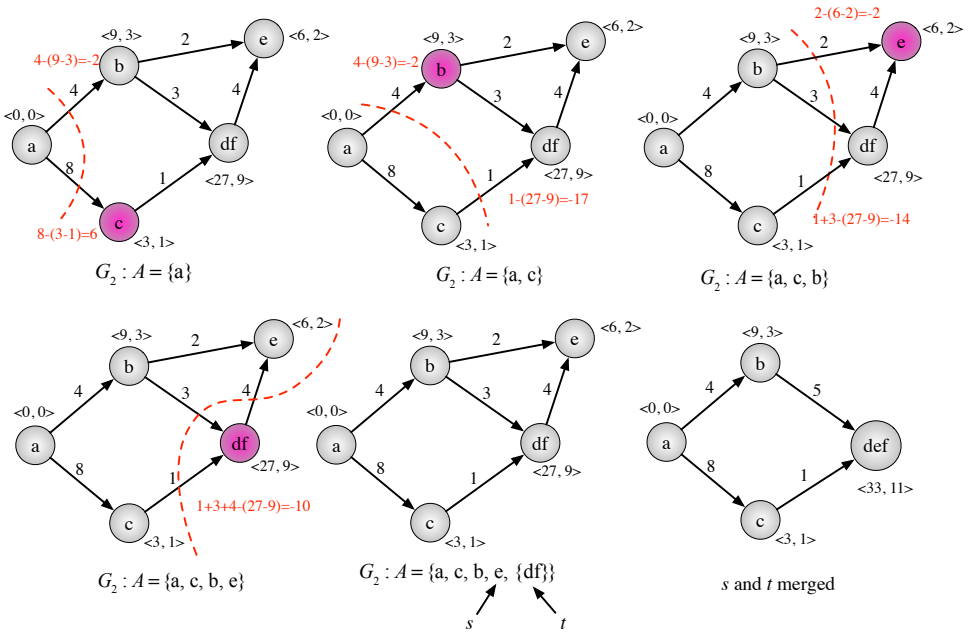


Figure 3.7: The 2nd phase of *MinCutPhase* function. The induced ordering of the vertices is a, c, b, s, t, where s = e and t = {df}. The 2nd cut-of-the-phase corresponds to the partitions {a, c, b, e} and {d, f} with the cut value: $C_{cut(A-\{\text{d, f}\},\ \{\text{d, f}\})} = 45 - (27 - 9) + (1 + 3 + 4) = 35$.

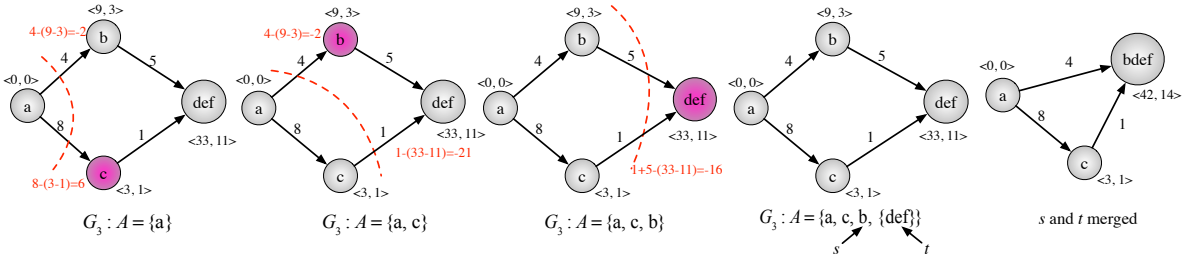


Figure 3.8: The 3rd phase of *MinCutPhase* function. The induced ordering of the vertices is a, c, s, t, where s = b and t = {def}. The 3rd cut-of-the-phase corresponds to the partitions {a, b, c} and {d, e, f} with the cut value: $C_{cut(\{\text{a, b, c}\},\ \{\text{d, e, f}\})} = 45 - (33 - 11) + (1 + 5) = 29$.

3.3.4 Case Study

Figure 3.6 shows that node a is defined as the starting point where the corresponding task will always be computed by the mobile device. We have s = d and t = f, and the induced ordering a, c, b, e, d, f of the vertices. Node f is cut off from the graph. The first *cut-of-the-phase* corresponds to the partitions {a, c, b, e, d} and {f}. Since the overall local cost is $C^{\text{local}} = \sum_{v \in V} w^{\text{local}}(v) = 45$, we can calculate the cut cost by using (3.10) as: $C_{cut(A-\text{f},\ \text{f})} = 45 - (15 - 5) + 5 = 40$. At the end, we merge nodes s = d and t = f into one.

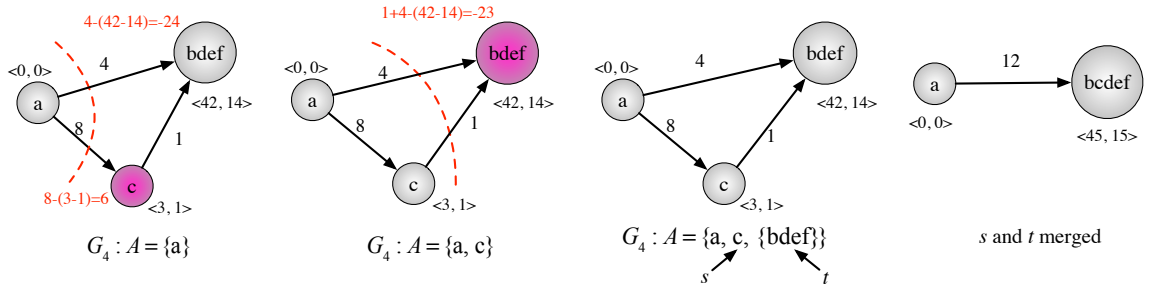


Figure 3.9: The 4th phase of *MinCutPhase* function. The induced ordering of the vertices is a, s, t, where s = c and t = {bdef}. The 4th cut-of-the-phase corresponds to the partition {a, c} and {b, d, e, f} with the cut value: $C_{cut(\{a,\,c\},\,\{b,\,d,\,e,\,f\})} = 45 - (42 - 14) + (1 + 4) = 22$.

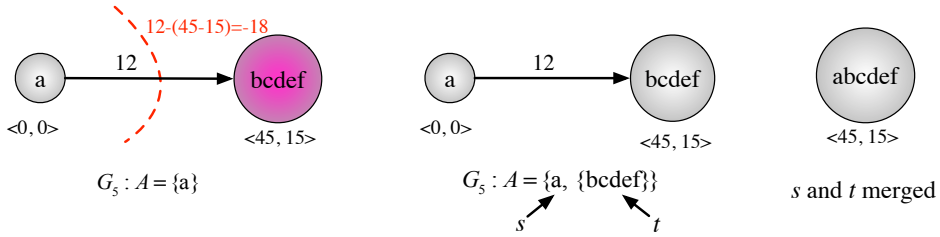


Figure 3.10: The 5th phase of *MinCutPhase* function. The induced ordering of the vertices is s, t, where s = a and t = {bcdef}. The 5th cut-of-the-phase corresponds to the partition {a} and {b, c, d, e, f} with cut value $C_{cut(\{a\},\,\{b,\,c,\,d,\,e,\,f\})} = 45 - (45 - 15) + 12 = 27$.

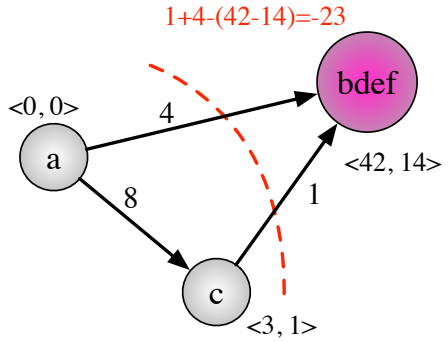


Figure 3.11: The optimal cut in phase 4

From Figs. 3.7-3.10, we repeat the same process of the *MinCutPhase* function as the first phase in Fig. 3.6. There are $|V| - 1 = 5$ phases, and at the end, all nodes are merged into one. Then, we compare all the cut values, the minimum value refers to the phase which has the optimal partitioning cut. In this scenario, the minimum cut of the graph G is the fourth *cut-of-the-phase*. The optimal cut is between {a, c} and {b, d, e, f} as depicted in Fig. 3.11 with the minimum cost of $C_{cut(\{a,\,c\},\,\{b,\,d,\,e,\,f\})} = 45 - (42 - 14) + (4 + 1) = 22$. Here, tasks b, d, e, f are offloaded to the remote cloud server while tasks a and c are executed locally.

3.4 Evaluation of the Partitioning Algorithm

We combine static analysis and dynamic profiling to construct the weighted consumption graph of an application. Using the Soot analysis tool, we can get all the tasks and the relations between tasks based on method invocations by traversing the graph. Combining Java bytecode rewriting with pretreatment information like speedup factor F and wireless bandwidth B (for convenient, we assume $B_{\text{upload}} = B_{\text{download}}$), we can obtain the execution time for each task (node weight) and the transmission time for each invocation (edge weight). These weights can be dynamically updated according to the varying processing capabilities of the cloud server and the wireless bandwidth.

3.4.1 Setup

We take a face recognition application[5] as an example. By analyzing the application with Soot, the call graph could be constructed as a tree-based topology in Fig. 3.12. Further, from the local estimated execution time, we can get the remote estimated execution time dividing by the speedup factor F. When offloading a task to the cloud, the communication cost incurred between the mobile device and the cloud is the data transfer divided by the bandwidth. Finally, with remote execution and transmission costs, we now have all information to get the weighted consumption graph.

To evaluate the partitioning algorithm, we need to know three different kinds of values:

- *Fixed Values*: they are set by the mobile application developer, determined based on a large number of experiments. For example, the computing power p_m, the power consumption while being idle p_i, and the transmission power p_{tr} are parameters specific to the mobile system. We use an HP iPAQ PDA with a 400-MHz Intel XScale processor that has the following values: $p_m \approx 0.9$ W, $p_i \approx 0.3$ W, and $p_{tr} \approx 1.3$ W [61].
- *Specific Values*: such parameters represent some state of mobile devices, e.g. the size of transferred data, the value of current wireless bandwidth B and the speedup factor F that depends on the speed of the current cloud server and the mobile device.
- *Calculated Values*: these values cannot be determined by application developers. For a given application, the computational cost is affected by input parameters and device characteristics, which can be measured using a program profiler. The communication cost is related to transmitting codes/data via wireless interfaces such as WiFi or 3G, which can be tracked by a network profiler.

Performance evaluation results encompass comparisons with other existing schemes, in contrast

[5]The face recognition application is built upon an open source code http://darnok.org/programming/face-recognition/, which implements the Eigenface face recognition algorithm

to the energy conservation efficiency and execution time. We compare the partitioning results with two other intuitive strategies without partitioning [6] and, for ease of reference, we list all three kinds of offloading techniques:

- *No Offloading* (*Local Execution*): all computation tasks of an application are running locally on the mobile device and there is no communication cost. This may be costly since as compared to the powerful computing capability at the cloud side, the mobile device is limited in processing speed and battery life.
- *Full Offloading*: all computation tasks of mobile applications (except the unoffloadable tasks) are moved from the local mobile device to the remote cloud for execution. This may significantly reduce the implementation complexity, which makes the mobile devices lighter and smaller. However, full offloading is not always the optimal choice since different application tasks may have different characteristics that make them more or less suitable for offloading [66].
- *Partial Offloading* (*With Partitioning*): with the help of the MCOP algorithm, all tasks including unoffloadable and offloadable ones are partitioned into two sets, one for local execution on the mobile device and the other for remote execution on the cloud server. Before a task is executed, it may require certain amount of data from other tasks. Thus, data migration via wireless networks is needed between tasks that are executed at different sides.

We define the saved cost in the partial offloading scheme compared to that in the no offloading scheme as *Offloading Gain*, which can be formulated as:

$$\textit{Offloading Gain} = 1 - \frac{\textit{Partial Offloading Cost}}{\textit{No Offloading Cost}} \cdot 100\%. \tag{3.11}$$

The offloading gains in terms of time, energy and the weighted sum of time and energy are described in (3.5), (3.7) and (3.9), respectively.

3.4.2 Evaluation in Computational Complexity

We implement the MCOP algorithm in Java[6] that can serve as a comparison to the theoretic results. As an example, we partition the constructed weighted consumption graph in Fig. 3.12 under the condition of the speedup factor $F = 2$ and the bandwidth $B = 1$ MB/s, where the *main* and *checkAgainst* methods are assumed as unoffloadable nodes. The optimal partitioning result is depicted in Fig. 3.13. The red nodes represent the application tasks that should be offloaded to the remote cloud

[6]An optimal partitioning algorithm, the code can be found in https://github.com/carlosmn/work-offload

and the blue nodes are the tasks that are supposed to be executed locally on the mobile device. The partitioning results will change as B or F varies.

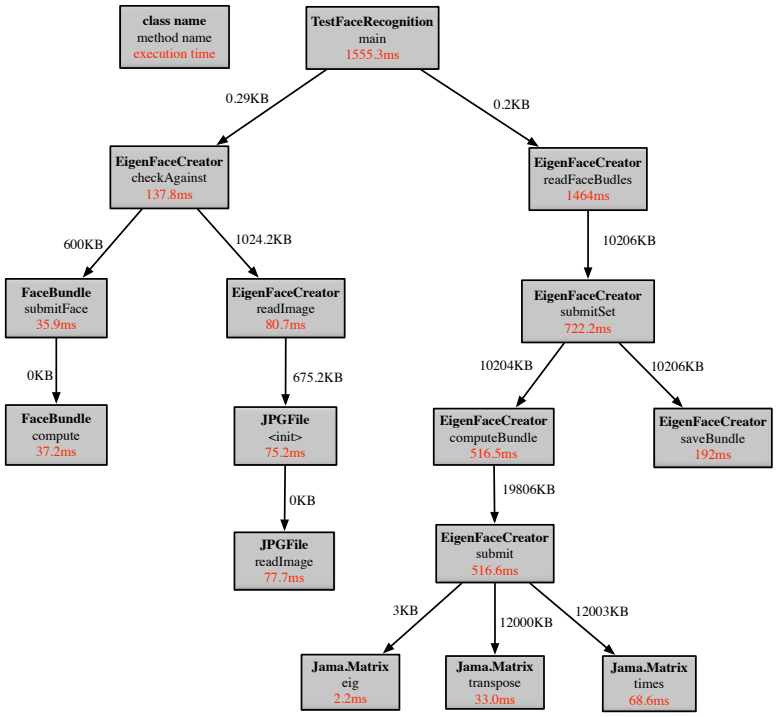


Figure 3.12: Call graph of a face recognition application

The running time of the Java implementation under a different number of application tasks is depicted in Fig. 3.14. We compare it with the theoretic computational complexity, which denoted as $O(|V|^2 \log |V| + |V||E|)$ in Section 3.3.3. We find they match each other well, which further proves that our partitioning algorithm has much lower time complexity than the LP solver which has exponential time complexity.

3.4.3 Evaluation in Dynamic Conditions

We build a graphical user interface (GUI) in MATLAB as shown in Fig. 3.15. The GUI is responsible for user interaction such as receiving input parameters and displaying the application partitioning results.

The user first inputs or selects the relative parameters, such as *Application Graph*, *Unoffloadable Nodes* and *Optimization Model*. We can either use the predefined application graphs of "linear", "loop", "tree" and "mesh" or just choose "user" to input any arbitrary consumption graph. Then,

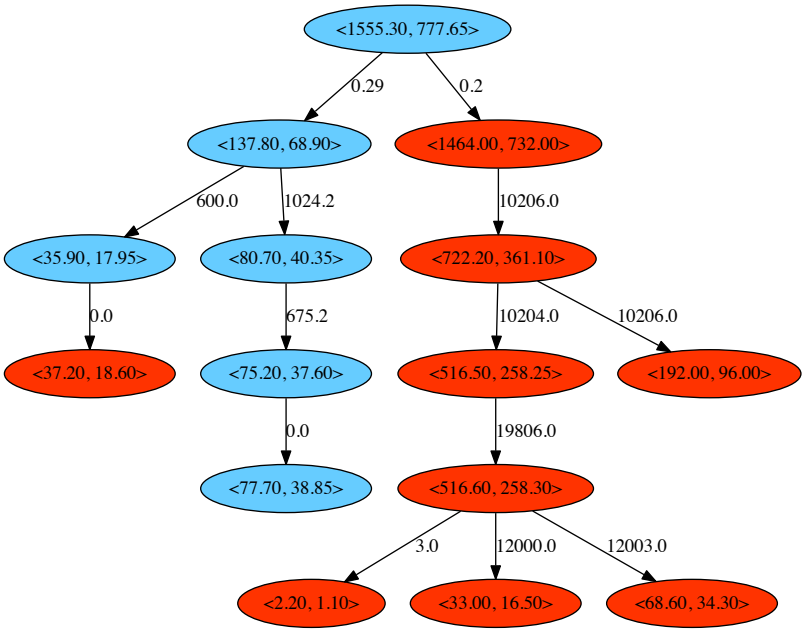


Figure 3.13: Optimal partitioning result of the face recognition application

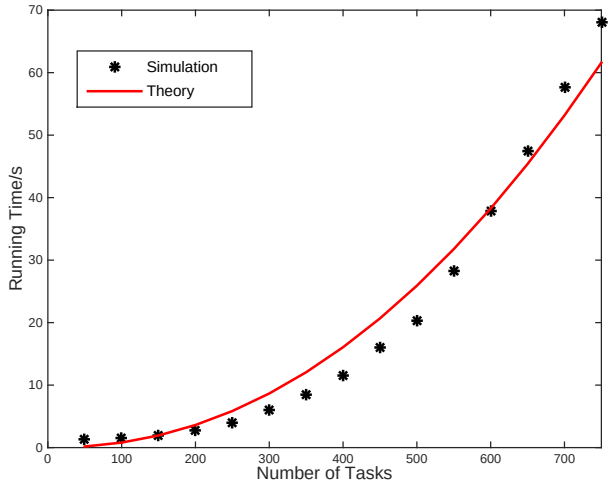


Figure 3.14: Running time of the MCOP algorithm under different number of tasks

by clicking the "Graph" button, a weighted consumption graph will be constructed based on the above parameters. Further, by clicking the "Start Partitioning" button, the partitioning process will begin, by calling the partitioning algorithm of MCOP. We can get the partitioning results such as *Partial Offloading Cost*, *No offloading Cost*, *Full Offloading Cost* and *Offloading Gain*. In addition, the optimal partitioning graph will appear like Fig. 3.16, which further proves the correctness of the

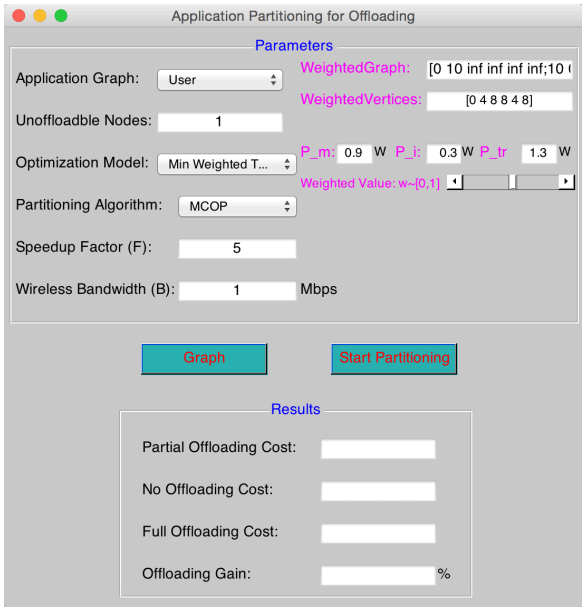


Figure 3.15: The GUI for demonstration

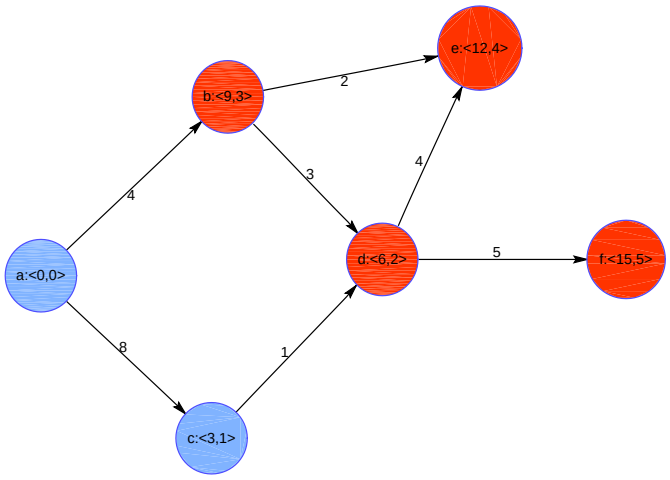
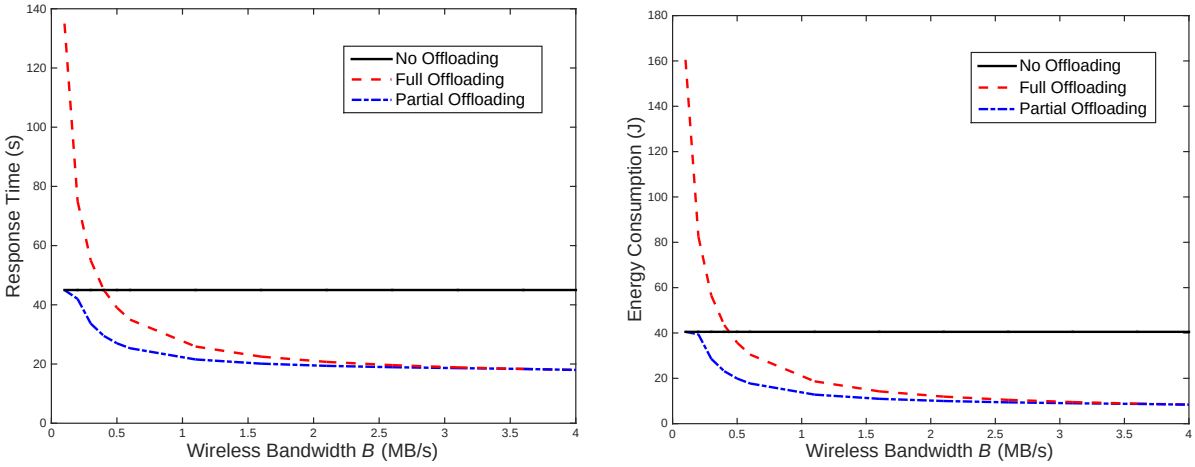


Figure 3.16: An optimal partitioning result of using the MCOP algorithm

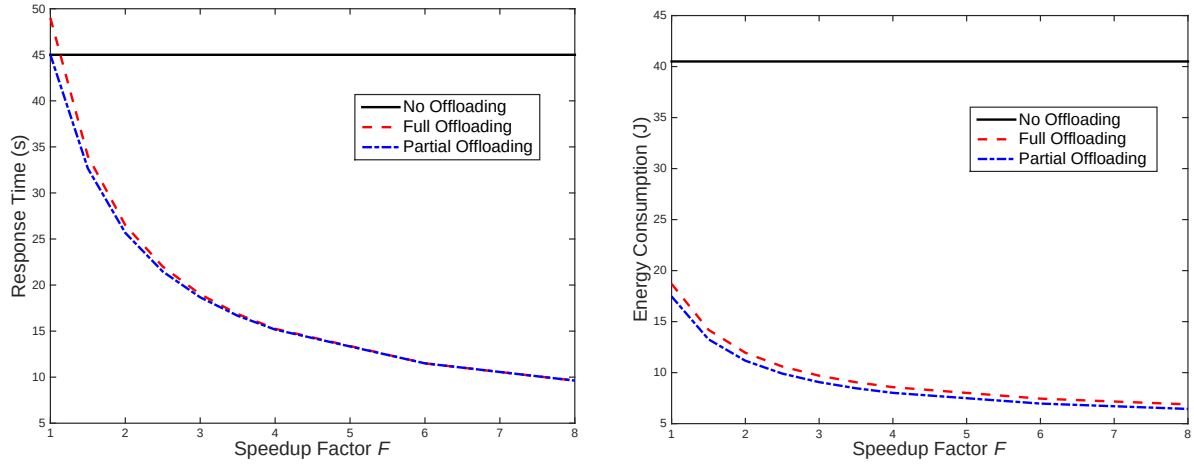
partitioning result in Fig. 3.11 with the minimum cost of 22. We can get different results under the different parameters of speedup factor F and wireless bandwidth B.

In Fig. 3.17 the speedup factor is set to $F = 3$. Since the low bandwidth results in much higher cost for data transmission, the full offloading scheme can not benefit from offloading. Given a relatively large bandwidth, the response time or energy consumption obtained by the full offloading scheme slowly approaches to the partial offloading scheme because the optimal partition includes



(a) Response Time (b) Energy Consumption

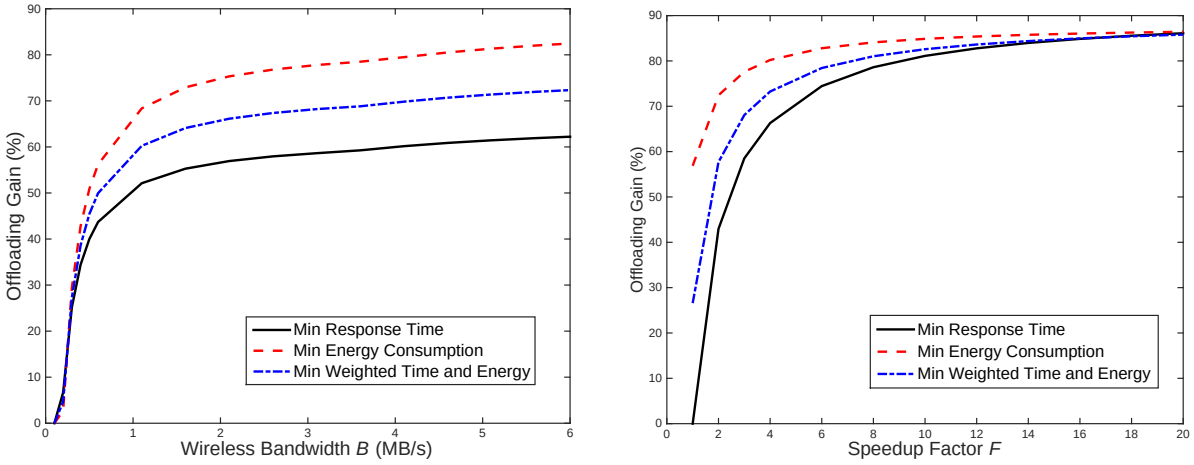
Figure 3.17: Comparisons of different schemes under different wireless bandwidths when the speedup factor $F = 3$



(a) Response Time (b) Energy Consumption

Figure 3.18: Comparisons of different schemes under different speedup factors when the bandwidth $B = 3$ MB/s

more and more tasks running on the cloud side until all offloadable tasks are offloaded to the cloud. With the higher bandwidth, they begin to coincide with each other and only decrease because all possible nodes are offloaded and the transmissions become faster. Both response time and energy consumption have the same trend as the wireless bandwidth increases. Therefore, bandwidth is a crucial element for offloading since the mobile system could benefit a lot from offloading in high



(a) Wireless bandwidth B ($F = 3$) (b) Speedup factor F ($B = 3$ MB/s)

Figure 3.19: Offloading gains under different environment conditions when $\omega = 0.5$

bandwidth environments, while with low bandwidth, the no offloading scheme is preferred.

In Fig. 3.18 the bandwidth is fixed as $B = 3$ MB/s. It can be seen that offloading benefits from higher speedup factors. When F is very small, the full offloading scheme can reduce energy consumption of the mobile device, however it takes much more response time than the no offloading scheme. The partial offloading scheme that adopts the MCOP algorithm can effectively reduce execution time and energy consumption, while adapting to environmental changes.

From Figs. 3.17-3.18, we can tell that the full offloading scheme performs much better than the no offloading scheme under certain adequate wireless network conditions, because the execution cost of running methods on the cloud server is significantly lower than on the mobile devices when the speedup factor F is large. The partial offloading scheme outperforms the no offloading and full offloading schemes and significantly improves the application performance, since it effectively avoids offloading tasks in the case of large communication cost between consecutive tasks compared to the full offloading scheme, and offloads more appropriate tasks to the cloud server. In other words, neither running all tasks locally on the mobile terminal nor always offloading their execution to a remote server, can offer an efficient solution, but rather our partial offloading scheme can do.

In Fig. 3.19(a) when the bandwidth is low, the offloading gain for all three cost models is very small and almost identical. That is because more time/energy will be spent in transferring the same data due to the low network bandwidth, resulting in increased execution cost. As the bandwidth increases, the offloading gain first rises drastically and then the increase becomes slower. It can be concluded that the optimal partitioning plan includes more and more tasks running on the cloud side

until all the tasks are offloaded to the cloud when the bandwidth increases. In Fig. 3.19(b) when F is small, the offloading gain for all three cost models is very low since a small value means very little computational cost reduction from remote execution. As F increases, the offloading gain first rises drastically and then approaches to the same value. That is because the benefits from offloading cannot neglect the extra communication cost. From Fig. 3.19, the proposed MCOP algorithm is able to effectively reduce the application's energy consumption as well as its execution time. Further, it can adapt to environmental changes to some extent and avoids a sharp decline in application performance once the bandwidth decreases.

3.5 Summary

We have studied how to disintegrate and distribute modules of an application between a mobile device and a dedicated cloud server, and effectively utilize the cloud resources. For applications under different scenarios, we construct them into weighted consumption graphs of arbitrary topology. To tackle the problem of dynamic partitioning in mobile environments, the MCOP algorithm is proposed to find the optimal partitioning plan from many possible partitioning candidates according to different cost models. Contrary to the traditional graph partitioning problem, our algorithm is not restricted to balanced partitions but takes the heterogeneity of mobile devices and cloud servers into account.

The MCOP algorithm has a stable quadratic runtime complexity for determining which parts of application tasks should be offloaded to the cloud server and what should be executed locally, in order to save energy of the mobile devices or to reduce application's execution time. We compared it with two other approaches, i.e. *No Offloading* and *Full Offloading*. Experimental results show that according to environmental changes (e.g. network bandwidth and cloud server performance), the proposed algorithm can effectively achieve the optimal partitioning result in terms of time and energy saving. Offloading benefits a lot from high bandwidth and large speedup factors, while low bandwidth favors the no offloading scheme.

Chapter 4

Offloading Decision Making: Where to Offload

A variety of clouds with different characteristics are emerging these days, and as a result several similar cloud services (from different cloud vendors) can be provided to a mobile device. Nearby cloudlets are also alternative destinations for offloading. Therefore, offloading decisions should be made to determine which resource to use. Where to offload is becoming a crucial issue due to the development of mobile cloud computing.

The goal of cloud service selection is to find an optimal cloud among a certain class of clouds that provide the same service, in order to carry out the offloaded tasks best. Unlike previous work that only uses one criterion, we combine the methods of analytic hierarchy process (AHP) and fuzzy technique for order preference by similarity to ideal solution (TOPSIS) based on multiple criteria to find the optimal cloud service for offloading.

Energy saving from mobile offloading is not guaranteed if the evoked data transfers via wireless networks consume an unpredictable amount of energy. Therefore, running a certain part of the application locally on the mobile device can be more advantageous and may save both energy and response time, especially in the presence of intermittent wireless connectivity. We derive an adaptive offloading decision algorithm based on Lyapunov optimization, which determines where to perform each application task (locally, cloud or cloudlet) such that energy consumption is minimized with a low delay penalty.

4.1 Multi-Criteria Decision Making in Cloud Selection

Different cloud providers usually offer different types of cloud services that shall not be comparable. Here, we consider a certain class of clouds that provide the same service. A more suitable example will be the peer cloud storage services such as Dropbox, OneDrive, iCloud, MS Skydrive and Google Drive. Offloading the same program to different clouds may perform different amounts of computing within the same duration due to the different speeds of cloud servers, and may cost different communication time due to the wireless network and cloud's availability. Therefore, a method for optimal cloud service selection is needed [115].

4.1.1 Problem Formulation

There are many criteria that need to be considered simultaneously in selecting the optimal cloud service. Here, we employ some of the QoS criteria from the Cloud Services Measurement Initiative Consortium (CSMIC) [22].

- *Performance*: does it do what we need? Its sub-criteria, *speed*, *accuracy* and *service response time* should be considered. *Speed* means how fast a cloud server for computing is. *Accuracy* is the degree of closeness to user expected actual value or result generated by using the cloud service [62].
- *Bandwidth*: how fast is the data transferred? It depends on the wireless link between the mobile device and the cloud. When the wireless connection is excellent, a large amount of application execution and data should be offloaded to the cloud, but when it is poor, only a small amount can be offloaded during limited time [118]. Different network types and conditions have a large impact on the communication time and energy consumption [21]. While for stable and high-speed networks, the application should be executed on the cloud server, it should be better executed locally on the mobile device in the situations of unreliable and weak connectivity.
- *Security*: is the service safe and privacy well protected? First of all, shifting all data and computing resources to the cloud is dangerous, e.g. tracking individuals through location-based navigation data offloaded to the cloud. Besides, security and privacy settings depend on the cloud providers since the data is stored and managed in the cloud [61]. Safely offloading adds non-trivial latency and energy overhead. Thus, careful choices must be made in deciding whether to encrypt communication and whether specific compute resources should be used. However, the QoS value of security is difficult to measure, but specific criteria that are measurable should be used when possible. Furthermore, security is also multi-dimensional in

nature and it includes many attributes like *data integrity*, *data privacy* and *data loss*.

- *Availability*: is it able to connect or access the cloud service? It is related with link failure and cloud availability during the offloading process. The cloud service may not be available in some cases and the distance to the cloud also affects the performance. Offloading is difficult in locations such as the basement of a building, interior of a tunnel, or subway, where the wireless network bandwidth is so small that cloud computing is not possible [61]. Dependence on a remote cloud could lead to problems when service outages occur. Failures may occur due to the mobile nature and unstable connectivity of wireless links, which render a less predictability of the performance of offloading systems [92].
- *Cost*: what is the monetary cost for the same amount of computing? It varies in different cloud services. Comparison results of AWS, Azure and AppEngine are listed in Table 4.1. The price varies a lot from instance types and different cloud service providers.

Table 4.1: Price for Public Cloud Service

Cloud Provider	Instance Type	CPU/Number of Cores	Price/hr
Azure	Small	1	$0.091
	Medium	2	$0.182
	Large	4	$0.364
	Extra Large	8	$0.728
AWS	Small	1	$0.12
	Medium	2	$0.24
	Large	4	$0.48
AppEngine	Default	N/A	$0.08

Therefore, the decision hierarchy for the cloud service selection is formed as shown in Fig. 4.1, including all the above service measurement criteria and sub-criteria. There are three hierarchies listed. The first level is called target hierarchy, meaning what the target is. Here, it aims at finding the optimal cloud service amongst available cloud services which satisfy the essential requirements of the mobile device. The second level is called criteria hierarchy, and five criteria: *performance*, *security*, *bandwidth*, *availability* and *cost* are considered for cloud service selection. The criteria can be classified into two categories: subjective criteria and objective criteria. The former is defined in linguistic/qualitative terms while the latter has monetary/quantitative definition. Root criteria can be made up of sub-criteria. The bottom level is named decision hierarchy, in which we can make the final decision in choosing one of the alternative clouds based on the analysis in criteria hierarchy.

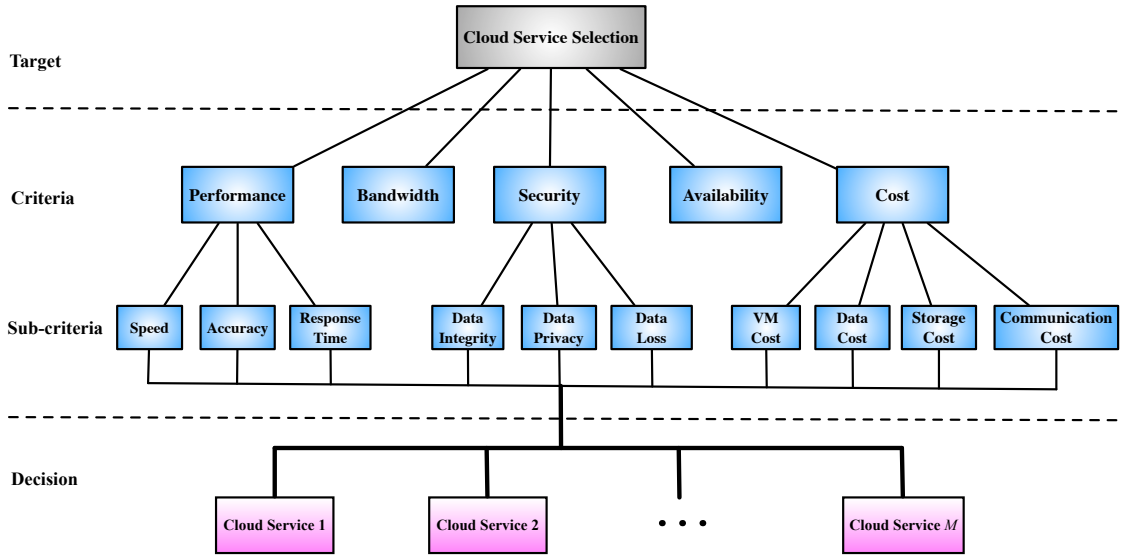


Figure 4.1: The decision hierarchy of cloud service selection

4.1.2 Steps of Cloud Service Selection

As shown in Fig. 4.2, there are three basic steps to be taken in the process of cloud service selection: *matching*, *ranking* and *selecting*.

- *Matching*: to find a list of available cloud services that are functionally matched with a service request by a mobile user. On the mobile device side, upon receipt of an offloading request, the *service request module* invokes the *cloud discovery module* to find an appropriate cloud service according to the task of *service level agreement (SLA) management* that keeps track of SLAs of customers with cloud providers and their fulfillment history. The candidate cloud services are registered based on the collected information in the *cloud registry module*.
- *Ranking*: to evaluate and rank the available cloud services according to QoS values and the results of criteria and sub-criteria calculation. The *criteria calculator module* depends on the tasks of qualitative and quantitative measurements. Qualitative criteria are those that cannot be quantified and are mostly inferred based on previous user's experiences, e.g. *security*. Quantitative criteria are those that be measured by using software and hardware monitoring tools [110], e.g. *bandwidth*, *VM cost* and *speed*.
- *Selecting*: the *decision maker module* is invoked to choose the optimal cloud service according to the ranked list of cloud services. And then the *offloading invoker module* is triggered to partition the application into local partition and remote partition, and the latter is then offloaded to the selected cloud.

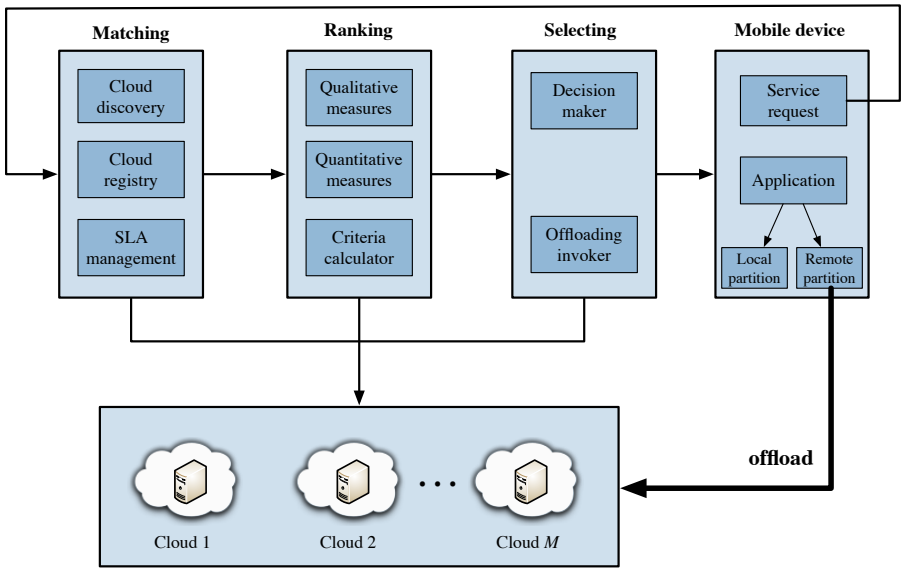


Figure 4.2: Steps of cloud service selection for offloading

4.1.3 Methods of AHP and Fuzzy TOPSIS

The selection process can be a hard task since a variety of data need to be analyzed and many factors need to be considered. We combine the methods of analytic hierarchy process (AHP) and fuzzy technique for order preference by similarity to ideal solution (TOPSIS), which are ideal ways to do multi-criteria decision making [88]. AHP is employed to obtain weights of the criteria for each cloud service and fuzzy TOPSIS is to determine the priorities of the alternative clouds in decision-making process [24].

The AHP Method

Analytic hierarchy process (AHP) is a process for determining the relative importance of a set of alternatives in a multi-criteria decision problem. It converts the evaluations to numerical values that can be processed and compared and derives a numerical weight or priority for each element of the hierarchy.

The results of the pairwise comparison on N criteria can be expressed in an evaluation matrix:

$$\mathbf{A} = (a_{ij})_{N\times N} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix}, a_{ii} = 1, a_{ji} = 1/a_{ij},$$

where element a_{ij} is based on a standardized comparison scale of nine levels as shown in Table 4.2 [88]. The relative weights are given by eigenvector ($\mathbf{w}$) corresponding to the largest eigenvalue (λ_{max}) as:

$$\mathbf{A}\mathbf{w} = \lambda_{\text{max}}\mathbf{w}. \tag{4.1}$$

Table 4.2: Importance Scale and Its Definition

Definition	Intensity of importance
Equally important	1
Moderately more important	3
Strongly more important	5
Very strongly more important	7
Extremely more important	9
Intermediate	2, 4, 6, 8

The output of AHP is strictly related to the consistency of the pairwise comparison. The consistency index (CI) is:

$$\text{CI} = \frac{\lambda_{\text{max}} - N}{N - 1}. \tag{4.2}$$

The consistency ratio (CR), usage of which let someone to conclude whether the evaluations are sufficiently consistent [24], is calculated as:

$$\text{CR} = \text{CI}/\text{RI}, \tag{4.3}$$

where the random index (RI) is only relevant with the matrix order. To meet the consistency, CR must be less than 0.1.

The Fuzzy TOPSIS Method

Technique for order preference by similarity to ideal solution (TOPSIS) is widely used to solve decision problems in real situation. The reason we adopt fuzzy TOPSIS here is that it is intuitively easy for the decision-makers to use and calculate through a triangular fuzzy number, which is proved to be an effective way for formulating decision problems [24]. The process steps of fuzzy TOPSIS can be outlined as follows [88]:

1) Establish a decision matrix for the ranking. The structure of the matrix is expressed by:

$$\begin{array}{c} \\ A_1 \\ A_2 \\ \vdots \\ A_i \\ \vdots \\ A_M \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & C_2 & \cdots & C_j & \cdots & C_N \end{array} \\ \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2N} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{iN} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_{M1} & x_{M2} & \cdots & x_{Mj} & \cdots & x_{MN} \end{pmatrix} \end{array}$$

where C_j is the j^{th} criterion, A_i is the i^{th} candidate cloud service. There is no need for normalization since the triangular fuzzy number $x_{ij} \in [0, 1]$. Figure 4.3 describes membership functions of linguistic values, which are used for evaluation of alternative weapons in this step, and the corresponding triangular fuzzy numbers are listed in Table. 4.3.

2) Calculate the weighted normalized decision matrix. The weighted normalized value v_{ij} is calculated as:

$$v_{ij} = x_{ij} \times w_j, \quad i = 1, 2, \cdots, M, \quad j = 1, 2, \cdots, N, \tag{4.4}$$

where w_j denotes the weight of the j^{th} criterion, which is obtained from the AHP method.

3) Determine the positive-ideal (A^+) and negative-ideal solutions (A^-), respectively:

$$A^+ = \{v_1^+, v_2^+, \cdots, v_N^+\} = \left\{ \left(\max_j v_{ij} | i \in I \right), \left(\min_j v_{ij} | i \in I' \right) \right\}, \tag{4.5}$$

$$A^- = \{v_1^-, v_2^-, \cdots, v_N^-\} = \left\{ \left(\min_j v_{ij} | i \in I \right), \left(\max_j v_{ij} | i \in I' \right) \right\}. \tag{4.6}$$

For normalized positive triangular numbers, we can define the fuzzy positive-ideal and negative-ideal solutions. As for benefit criterion, we have $v_j^+ = (1, 1, 1)$ and $v_j^- = (0, 0, 0)$, while for cost criterion, $v_j^+ = (0, 0, 0)$ and $v_j^- = (1, 1, 1)$.

4) Calculate the distance of each alternative from A^+ and A^- using the Euclidean distance:

$$D_i^+ = \sum_{j=1}^{N} d(v_{ij}, v_j^+), \quad i = 1, 2, \cdots, M, \tag{4.7}$$

$$D_i^- = \sum_{j=1}^{N} d(v_{ij}, v_j^-), \quad i = 1, 2, \cdots, M, \tag{4.8}$$

where $d(v_{ij}, v_j)$ calculates the Euclidean distance between v_{ij} and v_j .

5) Calculate the relative closeness to ideal solution, denoted as:

$$C_i^* = \frac{D_i^-}{D_i^+ + D_i^-}. \tag{4.9}$$

6) Rank the alternatives according to C_i^* in descending order. The nearer the value C_i^* close to 1 means the better the performance of the alternatives.

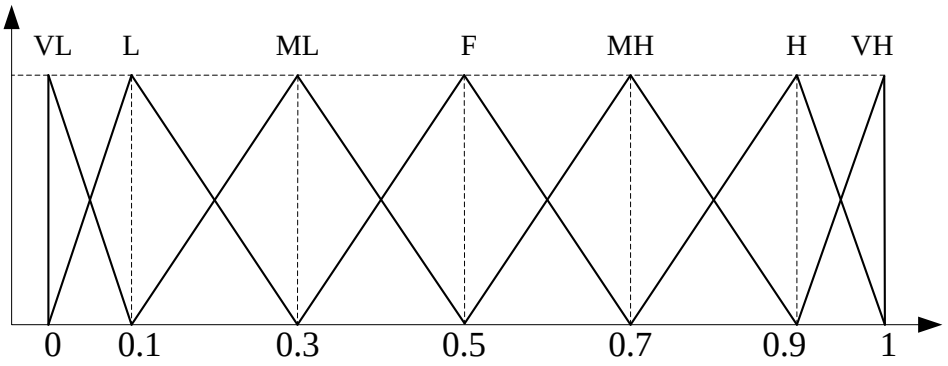


Figure 4.3: Membership functions of linguistic values

Table 4.3: Fuzzy Membership Functions

Linguistic values	Fuzzy ranges
Very low (VL)	(0, 0, 0.1)
Low (L)	(0, 0.1, 0.3)
Medium low (ML)	(0.1, 0.3, 0.5)
Fair (F)	(0.3, 0.5, 0.7)
Medium high (MH)	(0.5, 0.7, 0.9)
High (H)	(0.7, 0.9, 1)
Very high (VH)	(0.9, 1, 1)

Calculate the Weights of Criteria

The priority of importance depends on what we care about most. For instance, if the offload data is neither privacy nor confidential, in this case, *security* is the least important factor among the five criteria. For mobile offloading systems, *bandwidth* is considered the most significant because

it decides the extra communication cost between the mobile device and the cloud. Besides, *performance* is also important since it determines the application's execution time and affects battery consumption of the mobile device. We assume the priority of importance is ranked as: *bandwidth* > *performance* > *availability* > *security* > *cost*, although the priority of these five criteria can vary in other situations.

Employing the importance scale given in Table 4.2, we get the pairwise comparison matrix as shown in Table 4.4. By using the AHP method, we can calculate the weights of the criteria as shown in Table 4.5. It can be seen that *bandwidth* and *performance* are determined as the most important criteria. Besides, the consistency ratio (CR) is $0.020 < 0.1$ (criteria checking point). Thus, the weights are shown to be consistent which can be used in the decision-making process.

Table 4.4: Pairwise Comparison Matrix for Criteria

Criteria	*bandwidth*	*cost*	*performance*	*security*	*availability*
bandwidth	1	9	3	7	5
cost	1/9	1	1/6	1/2	1/3
performance	1/3	6	1	4	3
security	1/7	2	1/4	1	1/2
availability	1/5	3	1/3	2	1

Table 4.5: Results Obtained from AHP

Criteria	Weights	λ_{max}, CI, RI	CR
bandwidth	0.528		
cost	0.042	λ_{max}=5.089	
performance	0.252	CI=0.022	0.020
security	0.068	RI=1.12	
availability	0.110		

Select the Optimal Cloud Service

We assume that there are four candidate cloud services available. Besides, it can be seen that *monetary cost* is a cost criterion whereas the others are benefit criteria. The results of the fuzzy weighted decision matrix are given in Table 4.6.

The results of fuzzy TOPSIS analysis are summarized in Table 4.7. D_i^+ and D_i^- can be calculated by using (4.7) and (4.8). On the basis of C_i^* values, the ranking of the clouds in descending order

Table 4.6: Weighted Evaluation Matrix for the Alternative Clouds

Cloud	*bandwidth*	*cost*	*performance*	*security*	*availability*
cloud 1	VL	H	MH	L	ML
cloud 2	VH	H	VH	VL	L
cloud 3	F	ML	H	H	MH
cloud 4	MH	L	F	MH	VH
cloud 1	(0, 0, 0.1)	(0.7, 0.9, 1)	(0.5, 0.7, 0.9)	(0, 0.1, 0.3)	(0.1, 0.3, 0.5)
cloud 2	(0.9, 1, 1)	(0.7, 0.9, 1)	(0.9, 1, 1)	(0, 0, 0.1)	(0, 0.1, 0.3)
cloud 3	(0.3, 0.5, 0.7)	(0.1, 0.3, 0.5)	(0.7, 0.9, 1)	(0.7, 0.9, 1)	(0.5, 0.7, 0.9)
cloud 4	(0.5, 0.7, 0.9)	(0, 0.1, 0.3)	(0.3, 0.5, 0.7)	(0.5, 0.7, 0.9)	(0.9, 1, 1)
Weight	0.528	0.042	0.252	0.068	0.110
cloud 1	(0, 0, 0.053)	(0.029,0.038, 0.042)	(0.126, 0.176, 0.227)	(0, 0.007, 0.020)	(0.011, 0.033, 0.055)
cloud 2	(0.475, 0.528, 0.528)	(0.029,0.038, 0.042)	(0.227, 0.252, 0.252)	(0, 0, 0.007)	(0, 0.011, 0.033)
cloud 3	(0.158, 0.264, 0.370)	(0.004,0.013, 0.021)	(0.176, 0.227, 0.252)	(0.048, 0.061, 0.068)	(0.055, 0.077, 0.099)
cloud 4	(0.264, 0.370, 0.475)	(0, 0.004, 0.013)	(0.076, 0.126, 0.176)	(0.034, 0.048, 0.061)	(0.099, 0.110, 0.110)
A^+	$v_1^+ = (1,1,1)$	$v_2^+ = (0,0,0)$	$v_3^+ = (1,1,1)$	$v_4^+ = (1,1,1)$	$v_5^+ = (1,1,1)$
A^-	$v_1^- = (0,0,0)$	$v_2^- = (1,1,1)$	$v_3^- = (0,0,0)$	$v_4^- = (0,0,0)$	$v_5^- = (0,0,0)$

Table 4.7: Results of Fuzzy TOPSIS

Alternatives	D_i^+	D_i^-	C_i^*
cloud 1	3.802	1.225	0.244
cloud 2	3.267	1.743	0.348
cloud 3	3.402	1.624	0.323
cloud 4	3.365	1.662	0.331

are cloud 2, cloud 4, cloud 3 and cloud 1, where cloud 2 with $C_2^* = 0.348$ is the optimal alternative among the four clouds. In other words, we should choose cloud 2 for offloading when considering the five criteria simultaneously.

Qualitative and Quantitative Measurements of Criteria

For the objective criteria and sub-criteria listed in Fig. 4.1, such as the criterion *bandwidth* and the sub-criteria of *performance*: *speed*, *accuracy* and *response time*, the triangular fuzzy numbers can be used directly. Since it is difficult to measure or acquire in time, we use the historical data based on the mobile user's experience to construct the evaluation value, e.g. the triangular fuzzy number return on assets can be expressed as: $\left(\min_i\{h_i\}, \left(\prod_{i=1}^{t} h_i\right)^{1/t}, \max_i\{h_i\}\right)$, where $h_1, h_2, \ldots, h_t$ denote the return on assets of the past t periods. For example, the results of triangular fuzzy numbers are obtained in Table 4.8 when three historical observations are used. The weights of security and bandwidth are obtained from the AHP method as shown in Table 4.5. Similarly, we can get

the weights of sub-criteria according to the AHP method. We can also get the graded mean integration representation from Table 4.8. Let $A_i = (a_i, b_i, c_i), i = 1, 2, \ldots, n$, be n triangular fuzzy numbers. By the graded mean integration representation method [27], the graded mean integration representation $P(A_i)$ of A_i is $P(A_i) = (a_i + 4b_i + c_i)/6$.

Table 4.8: Results of Triangular Fuzzy Numbers

Criteria (Weights)	Sub-criteria (Weights)	Historical data				Triangular fuzzy numbers			
		Cloud 1	Cloud 2	Cloud 3	Cloud 4	Cloud 1	Cloud 2	Cloud 3	Cloud 4
performance (0.252)	*accuracy* (0.3)	60% 70% 68%	95% 99% 97%	70% 75% 80%	80% 85% 90%	(60%, 65.85%, 70%)	(95%, 96.99%, 99%)	(70%, 74.89%, 80%)	(80%, 84.90%, 90%)
	speed (0.6)	7 8 10	20 21 22	10 12 11	18 20 16	(7, 8.24, 10)	(20, 20.98, 22)	(10, 10.97, 12)	(16, 17.93, 20)
	response time (0.1)	400 320 440	20 30 50	80 100 120	300 280 320	(320, 383.31, 440)	(20, 31.07, 50)	(80, 98.65, 120)	(280, 299.55, 320)
bandwidth (0.528)	*bandwidth* (1)	32 40 46	200 256 300	80 90 120	180 160 170	(32, 38.90, 46)	(200, 248.58, 300)	(80, 95.24, 120)	(160, 169.80, 180)

For the subjective criteria and sub-criteria, such as the sub-criteria of *security*: *data integrity*, *data privacy* and *data loss*, the triangular fuzzy numbers to evaluate the superiority of alternatives can be $S = \{VL, L, ML, F, MH, H, VH\}$, where $VL =$ Very Low, $L =$ Low, $ML =$ Medium Low, $F =$ Fair, $MH =$Medium High, $L =$ High, and $VH =$ Very High. The fuzzy values are as shown in detail in Table 4.3.

Overall, both methods can be used to evaluate the importance weights of all criteria and sub-criteria as well as the fuzzy ratings of alternative cloud service, when it is still a challenge to measure or acquire the parameters of criteria timely in practical systems.

4.2 Energy-Efficient Offloading Decisions

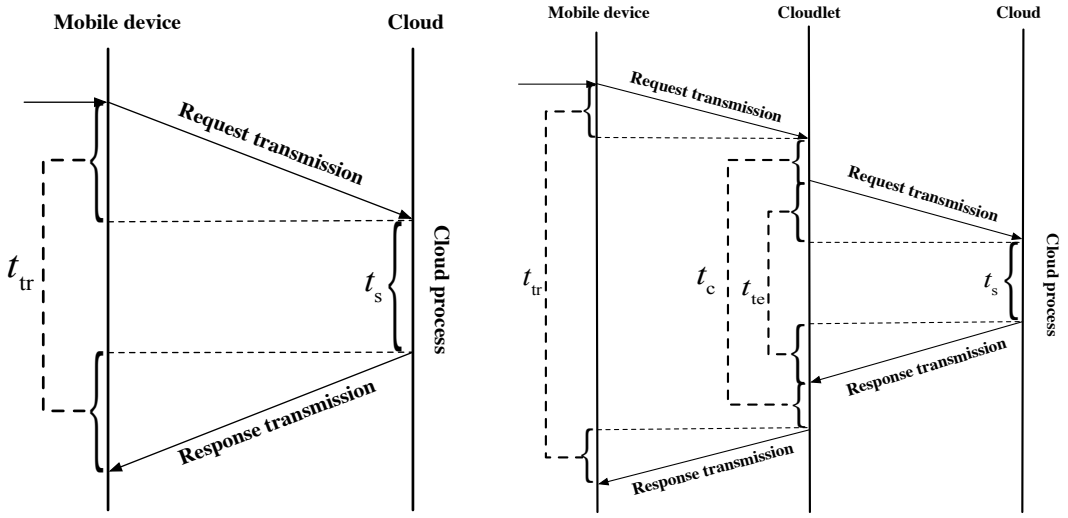
To prolong battery life, mobile devices can offload part of their computational workload via a nearby cloudlet to a remote cloud under varying wireless environment conditions. The design objective of our algorithm is to identify under which circumstances would offloading be beneficial and to minimize the energy consumed by the mobile device, while meeting a deadline. Accordingly, there are three constraints of the proposed approach [48]:

1) Minimizing the average energy consumption of the mobile device.
2) Satisfying the given deadline on runtime of the data processing for each application.

3) Opportunistic partitioning of the application tasks into different categories (e.g. run on mobile device, cloudlet or cloud).

4.2.1 Mobile Cloud Offloading Services

As discussed in Section 2.1.1, offloading infrastructures can be organized as a two- or three-level hierarchy [32].



(a) Two-level offloading service (b) Three-level offloading service

Figure 4.4: Different mobile cloud offloading services

Two-Level Offloading Systems

Rather than running applications locally and directly requesting data from content providers, a mobile device can offload parts of its workload to the cloud, taking advantage of the abundant cloud resources to help gather, store and process data [71]. As shown in Fig. 4.4(a), there are three steps for computation offloading: sending the required data to the cloud, waiting for the cloud to complete execution of the offloaded computation and receiving execution results from the cloud. The total response time includes the transmission delay $t_{\text{tr}} = D/B$ and the time to process the requested task on the cloud t_{s}. Therefore, the response time taken and the energy consumed to handle a cloud

service request can be calculated as follows:

$$T_{\text{2-level}} = t_{\text{tr}} + t_{\text{s}}, \tag{4.10}$$

$$E_{\text{2-level}} = p_{\text{tr}} \cdot t_{\text{tr}} + p_{\text{i}} \cdot t_{\text{s}}, \tag{4.11}$$

where the parameters are defined in Table 4.9.

Table 4.9: Parameters for Offloading Decisions

Symbol	Meaning
t_{m}	Execution time on the mobile device
t_{s}	Time taken to process the actual service on the cloud server
t_{c}	Time taken to process the request on the cloudlet
t_{te}	Transmission time between the cloudlet and cloud
t_{tr}	Transmission time between the mobile device and cloud/cloudlet
D	Transmitted data between the mobile device and cloud
B	Bandwidth between the mobile device and cloud
B_1	Bandwidth between the mobile device and cloudlet
B_2	Bandwidth between the cloudlet and cloud
p_{m}	Power for computing
p_{i}	Power while being idle
p_{tr}	Power for sending and receiving data

Three-Level Offloading Systems

The cloudlet becomes a better choice for mobile offloading when direct offloading to the cloud is unstable. As shown in Fig. 2.1, cloudlets are dispersed and located close to the mobile device while clouds are far away. The mobile device does not need to communicate with the distant cloud during an entire offload process, but only with the cloudlet. It acts a middleware, does some preprocessing and reduces the latency to the cloud. Mobile users seamlessly utilize nearby computers to obtain the resource benefits of cloud computing without incurring delays and jitter. The need for real-time interactive response can be met by low latency, one-hop, high-bandwidth wireless access to the cloudlet.

The three-level offloading service in Fig. 4.4(b) consists of a local tier of mobile devices, a middle tier of nearby cloudlets, typically located at the mobile devices' access point but characterized by limited resources and a remote tier of distant cloud servers, which have practically infinite resources [33]. It needs five steps to perform computation offloading: the mobile device sends the required data to the cloudlet, the cloudlet sends the required data to the cloud, waiting for the cloud to complete

execution, the cloudlet receives the execution results from the cloud, and the mobile device receives execution results from the cloudlet [16]. Similarly, the total response time and energy consumption are calculated as:

$$T_{\text{3-level}} = t_{\text{tr}} + t_{\text{te}} + t_{\text{c}} + t_{\text{s}}, \tag{4.12}$$

$$E_{\text{3-level}} = p_{\text{tr}} \cdot t_{\text{tr}} + p_{\text{i}} \cdot (t_{\text{te}} + t_{\text{c}} + t_{\text{s}}), \tag{4.13}$$

where t_{te} is the transmission time between the cloudlet and cloud, and t_{c} is the time taken to process the request at the cloudlet.

Offloading Decision Criteria

Communication cost between the mobile device and the cloud depends on the network bandwidth. Since the bandwidth of WLAN is considerably higher than the bandwidth provided by radio access to a mobile device, different wireless technologies offer a competitive choice to connect to a nearby cloudlet and then to the cloud [105]. As depicted in Fig. 4.5, the bandwidth between the mobile device and the cloudlet is B_1, which generally uses Bluetooth or a high-bandwidth WLAN. The connection between the cloudlet and the cloud is usually wired with bandwidth B_2, which uses broadband technology like internet. The connection between the mobile device and the cloud is wireless with bandwidth B, which uses cellular or WiFi interface. Mostly, we have $B \leq B_1$ and $B \leq B_2$.

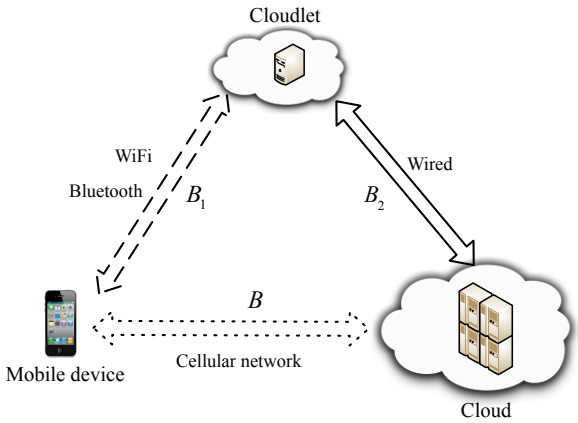


Figure 4.5: Model of mobile offloading systems

From Fig. 4.5, the two-level offloading scheme saves time only if [118]:

$$t_{\text{m}} > t_{\text{s}} + \frac{D}{B}. \tag{4.14}$$

It is more profitable to offload the program directly to the cloud (two-level offloading) instead of executing locally, when the following condition is satisfied:

$$p_{\mathrm{m}} \cdot t_{\mathrm{m}} > p_{\mathrm{tr}} \cdot \frac{D}{B} + p_{\mathrm{i}} \cdot t_{\mathrm{s}}, \tag{4.15}$$

we compare the energy consumed by local execution with the energy consumption due to offloading to the cloud, and if the former is greater than the latter, then we decide to run the application at the remote cloud server.

Similarly, the three-level offloading scheme saves time only if:

$$t_{\mathrm{m}} > t_{\mathrm{s}} + \frac{D}{B_1} + \frac{D}{B_2} + t_{\mathrm{c}}, \tag{4.16}$$

it saves energy only when the following condition is met:

$$p_{\mathrm{m}} \cdot t_{\mathrm{m}} > p_{\mathrm{tr}} \cdot \frac{D}{B_1} + p_i \cdot \left(\frac{D}{B_2} + t_{\mathrm{c}} + t_{\mathrm{s}} \right), \tag{4.17}$$

which is obtained by substituting $t_{\mathrm{tr}} = D/B_1$ and $t_{\mathrm{te}} = D/B_2$ into (4.13). We compare the local energy consumption with the energy cost due to offloading via a cloudlet to the cloud, and if the former is greater than the latter, then we decide to migrate the application to the cloud server.

Therefore, comparing (4.14) with (4.16), we find that the three-level offloading scheme works better than the two-level offloading scheme only if it meets:

$$\frac{D}{B} > \frac{D}{B_1} + \frac{D}{B_2} + t_{\mathrm{c}}. \tag{4.18}$$

According to (4.15) and (4.17), it is easy to see that the three-level offloading scheme performs better than the two-level offloading only if it satisfies:

$$p_{\mathrm{tr}} \cdot \frac{D}{B} > p_{\mathrm{tr}} \cdot \frac{D}{B_1} + p_{\mathrm{i}} \cdot \left(\frac{D}{B_2} + t_{\mathrm{c}} \right). \tag{4.19}$$

An offloading decision making process based on the predicted energy consumption is explained in Fig. 4.6. Given an application, we first estimate the average bandwidth of the current network, trigger the energy consumption predictor to get an expected energy consumption of the mobile device, and then use the offloading-decision criteria to take an offloading decision [136]. On one hand, if the predicted energy consumption satisfies both (4.17) and (4.19), we will apply the three-level offloading model; on the other hand, if the predicted energy consumption does not satisfy (4.19)

but (4.15), we choose the two-level offloading scheme. Apart from these cases, the application is preferably executed locally on the mobile device.

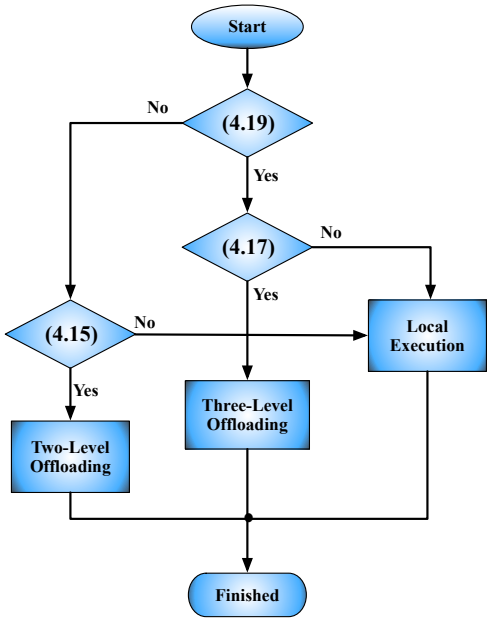


Figure 4.6: Offloading decision making based on the predicted energy consumption

4.2.2 Mathematical Model

A graphical model of where to perform the computation (locally, delegate it directly or via a cloudlet to cloud resources) is depicted in Fig. 4.7. The mobile device, the cloud and the cloudlet are represented as queueing nodes to capture the resource contention on these systems [16]. The wireless access network and the internet are denoted as simple delay centers representing average network delay when a task is remotely executed. Different tasks of applications emerge in a mobile device according to some process, each consisting of one or more tasks. We assume a simple model where functions in an application are not hierarchically called and all tasks run sequentially without parallelism. Suppose there are $N+1$ application tasks, with N unoffloadable tasks and $N+1-m$ offloadable tasks.

We further investigate the mathematical model by including offloading decision criteria. For the n^{th} application task, if minimum response time is selected as the offloading decision criterion at the t^{th} execution, it can be expressed mathematically as:

$$T_n(t) = \min\left\{T_n^{\text{local}}(t), T_n^{\text{cloud}}(t), T_n^{\text{cloudlet}}(t)\right\}, \forall n \in \{0, 1, \cdots, N\}, \forall t \in \{0, 1, \cdots, \infty\}, \quad (4.20)$$

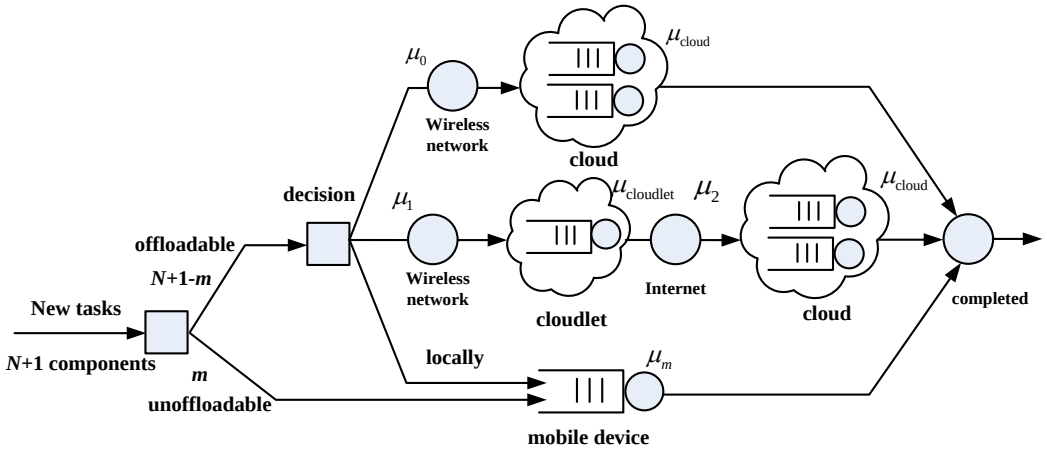


Figure 4.7: A mathematical model of adaptive offloading decision making

where $T_n^{\text{local}}(t)$, $T_n^{\text{cloud}}(t)$ and $T_n^{\text{cloudlet}}(t)$ are the time taken locally without offloading, the time taken due to direct offloading to the cloud and the time taken due to offloading via a cloudlet to the cloud, respectively. Further, $T_n^{\text{cloud}}(t)$ and $T_n^{\text{cloudlet}}(t)$ can be calculated as:

$$\begin{aligned} T_n^{\text{cloud}}(t) &= T_n^{\text{s}}(t) + \frac{D_n}{B(t)}, \\ T_n^{\text{cloudlet}}(t) &= \frac{D_n}{B_1(t)} + \frac{D_n}{B_2(t)} + T_n^{\text{c}}(t) + T_n^{\text{s}}(t), \end{aligned}$$

where T_n^s and T_n^c are the times taken to process the n^{th} task on the cloud and cloudlet, respectively.

Similarly, if minimum energy consumption is chosen as the offloading decision criterion, then it can be expressed as:

$$E_n(t) = \min\left\{E_n^{\text{local}}(t), E_n^{\text{cloud}}(t), E_n^{\text{cloudlet}}(t)\right\}, \tag{4.21}$$

where $E_n^{\text{local}}(t)$, $E_n^{\text{cloud}}(t)$ and $E_n^{\text{cloudlet}}(t)$ are the energy consumed locally, the energy consumed due to direct offloading to the cloud, and the energy consumed due to offloading via a cloudlet to the cloud, respectively. They can be calculated as:

$$\begin{aligned} E_n^{\text{local}}(t) &= p_{\text{m}} \cdot T_n^{\text{local}}(t), \\ E_n^{\text{cloud}}(t) &= p_{\text{tr}} \cdot \frac{D_n}{B(t)} + p_{\text{i}} \cdot T_n^{\text{s}}(t), \\ E_n^{\text{cloudlet}}(t) &= p_{\text{tr}} \cdot \frac{D_n}{B_1(t)} + p_{\text{i}} \cdot \left[\frac{D_n}{B_2(t)} + T_n^{\text{c}}(t) + T_n^{\text{s}}(t)\right]. \end{aligned}$$

Taking average in (4.20) and (4.21), we have the minimum average response time and average energy consumption as follows:

$$\bar{T} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{n=0}^{N} \mathbb{E}\{T_n(\tau)\}, \tag{4.22}$$

$$\bar{E} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{n=0}^{N} \mathbb{E}\{E_n(\tau)\}. \tag{4.23}$$

Therefore, the motivation of offloading could be to save energy, reduce execution time, or both. To save energy while satisfying a given response time requirement, we propose a dynamic offloading decision algorithm of where to offload based on Lyapunov optimization.

We assume that an application is composed of $N+1$ tasks and we independently make an offloading decision for each one. We use a graph $G = (R, S)$ with $|R| = N+1$ to represent the relationship among tasks. Each vertex $v \in R$ denotes a task and D_{uv} along the undirected edge (u, v) represents the size of data migrating from vertex u to v. When there is a request for application execution, a controller in the mobile device determines which tasks to be executed locally and which to be executed remotely [48].

At the t^{th} execution, let the offloading decision vector $\boldsymbol{\omega}(t)$ be defined as:

$$\boldsymbol{\omega}(t) = \Big\{\omega_n(t) | \omega_n(t) \in \{0,1,2\}\Big\}_{1\times(N+1)},\ \forall n \in \{0,1,\cdots,N\},\ t \in \{0,1,\cdots,\infty\}, \tag{4.24}$$

where $\omega_n(t) = 1$ denotes that the n^{th} task should run on the mobile device, $\omega_n(t) = 0$ represents that it is directly offloaded to the remote cloud, and $\omega_n(t) = 2$ denotes that it is first migrated to a nearby cloudlet and then to the cloud. We assume that the task with index 0 is an unoffloadable task that should always be executed locally, and therefore we always have $\omega_0(t) = 1$. The other N tasks can be offloadable such that $\omega_n(t)$ should be selected among $\{0,1,2\}$.

Total Response Time

The total response time is equal to the time taken by the components running locally and those running remotely, and plus the additional communication cost when they are in different places.

$$T(\boldsymbol{\omega}(t)) = \underbrace{\sum_{v\in R} \omega_v(t) \cdot T_v^{\text{m}}(t)}_{\text{local}} + \underbrace{\sum_{v\in R} |1-\omega_v(t)| \cdot T_v^{\text{r}}(t)}_{\text{remote}} + \underbrace{\sum_{(u,v)\in S} \Big[2 - |\omega_u(t) - \omega_v(t)|\Big] \cdot T_{uv}(t)}_{\text{communication}}, \tag{4.25}$$

where $\omega_v(t)$ and $\omega_u(t)$ are elements from (4.24), the local and remote execution times are separately denoted as:

$$T_v^{\mathrm{m}}(t) = \begin{cases} T_v^{\mathrm{m}}(t) & \text{if } \omega_v(t) = 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad T_v^{\mathrm{r}}(t) = \begin{cases} T_v^{\mathrm{s}}(t) & \text{if } \omega_v = 0 \text{ or } 2 \\ 0 & \text{otherwise} \end{cases},$$

and the transfer time from task u to task v can be calculated as:

$$T_{uv}(t) = \begin{cases} \frac{D_{uv}}{B(t)} & \text{if } \omega_u(t) \oplus \omega_v(t) = 1 \\ \frac{D_{uv}}{B_1(t)} + \frac{D_{uv}}{B_2(t)} + T_v^{\mathrm{c}}(t) & \text{if } \omega_u(t) \odot \omega_v(t) = 0 \\ 0 & \text{otherwise} \end{cases},$$

where D_{uv} is the communication data from task u to v, $\oplus$ and $\odot$ represent XOR computation and NOR computation for binary variables, respectively.

The total response time when all the tasks are executed locally is denoted as:

$$T_{\text{local}}(t) = \sum_{v \in R} T_v^{\mathrm{m}}(t). \tag{4.26}$$

Total Energy Consumption

The total energy consumption is equal to the one consumed by local components plus the one consumed in idle state for remote components and plus the extra energy consumed for transmission.

$$E(\boldsymbol{\omega}(t)) = \underbrace{\sum_{v \in R} \omega_v(t) \cdot E_v^{\mathrm{m}}(t)}_{\text{local}} + \underbrace{\sum_{v \in R} |1 - \omega_v(t)| \cdot E_v^{\mathrm{i}}(t)}_{\text{idle}} + \underbrace{\sum_{(u,v) \in S} \Big(2 - |\omega_u(t) - \omega_v(t)|\Big) \cdot E_{uv}(t)}_{\text{communication}}, \tag{4.27}$$

where $E_v^{\mathrm{m}}(t) = p_{\mathrm{m}} \cdot T_v^{\mathrm{m}}(t)$ is the local energy cost, $E_v^{\mathrm{i}}(t) = p_{\mathrm{i}} \cdot T_v^{\mathrm{r}}(t)$ is the energy consumed in idle state due to offloading and the energy consumed for data transfer is

$$E_{uv}(t) = \begin{cases} p_{\mathrm{tr}} \frac{D_{uv}}{B(t)} & \text{if } \omega_u(t) \oplus \omega_v(t) = 1 \\ p_{\mathrm{tr}} \frac{D_{uv}}{B_1(t)} + p_{\mathrm{i}} \Big[\frac{D_{uv}}{B_2(t)} + T_v^{\mathrm{c}}(t) \Big] & \text{if } \omega_u(t) \odot \omega_v(t) = 0 \\ 0 & \text{otherwise} \end{cases}.$$

Similarly, the total local energy consumption when all the tasks are executed on the mobile device

is denoted as:

$$E_{\text{local}}(t) = \sum_{v \in R} E_v^{\text{m}}(t). \tag{4.28}$$

As a partitioning example, three cases after making offloading decisions are listed in Fig. 4.8. Suppose task 1 is unoffloadable that can only be executed locally, while the others are offloadable tasks that can either be processed locally or offloaded to the cloud, directly or via a cloudlet. We use dotted arrows to represent offloading via the cloudlet to the cloud. In case 1, task 3 is executed on the mobile device, task 4 is offloaded directly to the cloud while task 2 is offloaded via the cloudlet to the cloud, thus the decision combination vector is $\boldsymbol{\omega}_1(t) = \{1, 2, 1, 0\}$. In case 2, task 2 and 4 are offloaded via the cloudlet to the cloud while task 3 is offloaded directly to the cloud, thus we have $\boldsymbol{\omega}_2(t) = \{1, 2, 0, 2\}$. In case 3, all three tasks are offloaded directly to the remote cloud and the decision combination vector is $\boldsymbol{\omega}_3(t) = \{1, 0, 0, 0\}$.

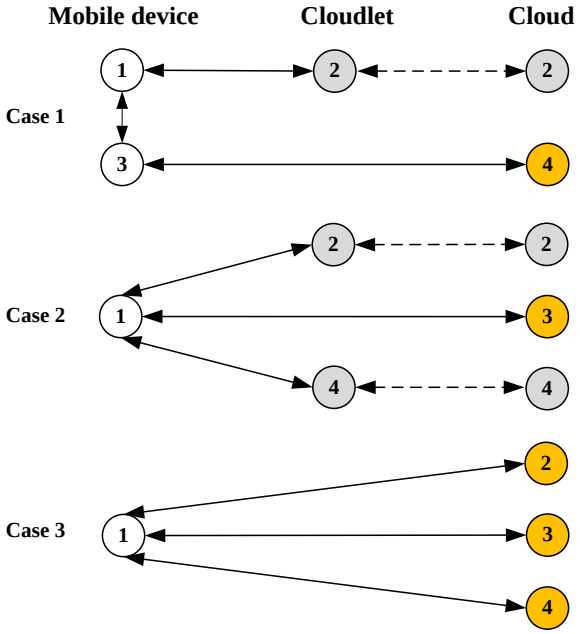


Figure 4.8: A partitioning example of where to offload

Challenges: Let $\boldsymbol{\Phi}$ be the set of all possible decision combinations. When the application has N offloadable tasks, we can obtain $|\boldsymbol{\Phi}| = 3^N$. For each execution, the steps to search for the optimal solution (i.e. to determine whether $\omega_n(t)$ should be 0, 1 or 2) grow exponentially with the number of vertices [21]. Therefore, it is difficult to obtain the optimal solution directly.

4.2.3 Lyapunov-based Algorithm

For a given decision combination vector $\boldsymbol{\omega}(t)$, the corresponding energy consumption for different executions may change due to variation in the available wireless network. In this case, it will be difficult to obtain the optimal solution. Therefore, we suppose that the available wireless network stays constant at the t^{th} execution.

The constraint is that the total response time of that partition should be less than or equal to a deadline named T_d. Let the execution indicator variable be defined as:

$$\sigma\left(\boldsymbol{\omega}(t)\right) = \begin{cases} 0 & \text{if } T\left(\boldsymbol{\omega}(t)\right) \leq T_d \\ 1 & \text{otherwise} \end{cases}. \tag{4.29}$$

A decision combination vector $\boldsymbol{\omega}(t)$ is feasible if the total response time satisfies the delay constraint, which is denoted as $\sigma\left(\boldsymbol{\omega}(t)\right) = 0$, otherwise, we have $\sigma\left(\boldsymbol{\omega}(t)\right) = 1$. A feasible decision combination vector $\boldsymbol{\omega}^*(t)$ with minimum energy consumption is the optimal solution among all the feasible decision vectors. Formally, we have:

$$\min_{\boldsymbol{\omega}(t)} \ \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\left\{E\left(\boldsymbol{\omega}(\tau)\right)\right\}, \tag{4.30}$$

$$\text{s.t.} \ \ \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\left\{\sigma\left(\boldsymbol{\omega}(\tau)\right)\right\} \leq \rho, \tag{4.31}$$

where ρ is the violation ratio of the number of executions which do not meet the deadline to the total number of executions. (4.31) ensures that the system is stable.

We define the dynamic offloading system as:

$$Q(t+1) = \max[Q(t) - \rho, 0] + \sigma\left(\boldsymbol{\omega}(t)\right), \quad \forall t \in \{0, 1, \cdots, \infty\}, \tag{4.32}$$

where $Q(t)$ is defined as the system state at the t^{th} execution, which depends on the violation ratio. Therefore, the larger $Q(t)$ is, the longer the system's response time is.

Before further discussing the decision function, we first present a Lemma from [39], which is related to the derivation of the decision function.

Theorem 2. *Let W, U, μ, and A be nonnegative real numbers and $W = \max[U - \mu, 0] + A$, then $W^2 \leq U^2 + \mu^2 + A^2 - 2U(\mu - A)$.*

For each execution, we define Lyapunov function as $L(Q(t)) = Q^2(t)/2$ and Lyapunov drift as the change in the Lyapunov function from one execution to the next [83]. According to Theorem 2,

we have:

$$\begin{aligned} L(Q(t+1)) - L(Q(t)) &= \frac{1}{2}\Big[Q^2(t+1) - Q^2(t)\Big] \\ &= \frac{1}{2}\Big\{\Big[\max[Q(t)-\rho,0] + \sigma\left(\boldsymbol{\omega}(t)\right)\Big]^2 - Q^2(t)\Big\} \\ &\leq \frac{\rho^2 + \sigma^2(\boldsymbol{\omega}(t))}{2} + Q(t)\cdot\Big[\sigma\left(\boldsymbol{\omega}(t)\right) - \rho\Big]. \end{aligned} \tag{4.33}$$

The conditional Lyapunov drift $\Delta(Q(t))$ is the expected change in the continuous execution of the Lyapunov function, i.e. $\Delta(Q(t)) \triangleq \mathbb{E}\Big\{L(Q(t+1)) - L(Q(t))|Q(t)\Big\}$. According to (4.33), we have that $\Delta(Q(t))$ for a general control policy satisfies:

$$\Delta(Q(t)) \leq C - \rho Q(t) + \mathbb{E}\Big\{Q(t)\sigma\left(\boldsymbol{\omega}(t)\right)|Q(t)\Big\}, \tag{4.34}$$

where $C \triangleq \mathbb{E}\Big\{\frac{\rho^2+\sigma^2(\boldsymbol{\omega}(t))}{2}|Q(t)\Big\} = \frac{\rho^2}{2} + \mathbb{E}\Big\{\frac{\sigma^2(\boldsymbol{\omega}(t))}{2}|Q(t)\Big\}$.

To stabilize the queue state while minimizing the average energy consumption, we incorporate the expected energy consumption over one execution. It can be designed to make control actions that greedily minimize a bound on the following drift-plus-penalty term at each execution [83]:

$$\Delta(Q(t)) + V\cdot\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(t)\right)|Q(t)\Big\}, \tag{4.35}$$

where $V \geq 0$ is a control parameter that represents an "importance weight" on how much we emphasize the energy minimization compared to the violation ratio of the deadline. In other words, V can be thought of as a threshold on the system queue state on which the control algorithm takes offloading decision. So V controls the tradeoff between the energy consumption and response time. Then substituting (4.34) into (4.35), yields:

$$\begin{aligned} \Delta(Q(t)) + V\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(t)\right)|Q(t)\Big\} &\leq C - \rho Q(t) + V\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(t)\right)|Q(t)\Big\} + \mathbb{E}\Big\{Q(t)\sigma\left(\boldsymbol{\omega}(t)\right)|Q(t)\Big\} \\ &= C - \rho Q(t) + \mathbb{E}\Big\{\Big[VE\left(\boldsymbol{\omega}(t)\right) + Q(t)\sigma\left(\boldsymbol{\omega}(t)\right)\Big]|Q(t)\Big\}. \end{aligned}$$

Note that our target is to minimize the average energy consumption. If we minimize the right-hand-side of the above inequality, we can save energy while keeping (4.32) stable. This is accomplished by searching for a feasible $\boldsymbol{\omega}(t)$ that greedily minimizes the decision criterion:

$$\arg\min_{\boldsymbol{\omega}(t)} \Big[VE\left(\boldsymbol{\omega}(t)\right) + Q(t)\sigma\left(\boldsymbol{\omega}(t)\right)\Big]. \tag{4.36}$$

Since the average violation rate is $\mathbb{E}\{\sigma(\boldsymbol{\omega}(t))\} \leq \rho$, the system is stable. We define the decision function as:

$$d(Q(t), \boldsymbol{\omega}(t)) = VE(\boldsymbol{\omega}(t)) + Q(t)\sigma(\boldsymbol{\omega}(t)). \tag{4.37}$$

For the t^{th} execution, we choose a decision combination vector $\boldsymbol{\omega}^*(t)$ such that $d(Q(t), \boldsymbol{\omega}^*(t))$ is minimized. We apply 1-opt local search algorithm that seeks for an optimal solution around the initial estimate, whose vectors of candidates are composed by all possible solutions with unitary Hamming distance. The 1-opt algorithm has low computational complexity that in terms of runtime is $O(|d|^3)$ [1].

Performance Bounds

For any control parameter $V > 0$, we achieve average energy consumption and queue backlog satisfying the following constraints [83]:

$$\bar{E} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\Big\{E(\boldsymbol{\omega}(\tau))\Big\} \leq \frac{C}{V} + E^*, \tag{4.38}$$

$$\bar{Q} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\Big\{Q(\tau)\Big\} \leq \frac{C + V(E^* - \bar{E})}{\varepsilon}. \tag{4.39}$$

Discussion: It can be seen from (4.38) and (4.39) that performance of the dynamic offloading decision algorithm depends on V, which controls the energy-delay tradeoff. Since the system state is closely related with response time, it follows a $[O(1/V), O(V)]$ tradeoff between the energy consumption and response time. We can achieve an average energy consumption $\bar{E}$ arbitrarily close to the optimum E^* with a diminishing gap $(1/V)$ while maintaining queue stability. However, this reduction is achieved at the expense of a larger delay because the average system state $\bar{Q}$ increases linearly with V. Therefore, we can tune V to flexibly trade off between energy consumption and response time. When the power constraint is stringent (e.g. the mobile device is running out of battery and no charger is available), choosing a larger V can save more energy at the expense of higher average response time and instead, when the battery supply is not so limited (e.g. a charger is available), we can reduce V to shorten the response time and enjoy better quality of service.

Proof. Because our decision combination vector $\boldsymbol{\omega}(\tau)$ minimizes the right-hand-side of the drift-

plus-penalty inequality at every τ^{th} execution, given the observed $Q(\tau)$, we have:

$$\begin{aligned}\Delta\left(Q(\tau)\right)+V\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(\tau)\right)|Q(\tau)\Big\} &\leq C-\rho Q(\tau)+V\mathbb{E}\Big\{E\left(\boldsymbol{\omega}^*(\tau)\right)|Q(\tau)\Big\}\\ &\quad +\mathbb{E}\Big\{Q(\tau)\sigma\left(\boldsymbol{\omega}^*(\tau)\right)|Q(\tau)\Big\}\\ &\leq C-\rho Q(\tau)+VE^*+Q(\tau)(\rho-\varepsilon)\\ &= C+VE^*-\varepsilon Q(\tau),\end{aligned}$$

where $\boldsymbol{\omega}^*(\tau)$ is any other (possibly randomized) transmission decision that can be made at the τ^{th} execution and E^* is the minimum energy consumption. Since $\mathbb{E}\left\{\sigma\left(\boldsymbol{\omega}^*(\tau)\right)\right\}\leq\rho$, there exists some $\boldsymbol{\omega}^*(\tau)$ and an arbitrarily small $\varepsilon>0$ that meet the requirement that $\mathbb{E}\left\{\sigma\left(\boldsymbol{\omega}^*(\tau)\right)\right\}\leq\rho-\varepsilon$. Taking expectations of the above inequality and using the law of iterated expectations, yields:

$$\mathbb{E}\Big\{L\left(Q(\tau+1)\right)\Big\}-\mathbb{E}\Big\{L\left(Q(\tau)\right)\Big\}+V\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(\tau)\right)\Big\}\leq C+VE^*-\varepsilon\mathbb{E}\{Q(\tau)\}.$$

Summing the above over $\tau\in\{0,1,\cdots,t-1\}$ for some positive integer t, yields:

$$\mathbb{E}\Big\{L\left(Q(t)\right)\Big\}-\mathbb{E}\Big\{L\left(Q(0)\right)\Big\}+V\sum_{\tau=0}^{t-1}\mathbb{E}\Big\{E\left(\boldsymbol{\omega}(\tau)\right)\Big\}\leq Ct+VE^*t-\varepsilon\sum_{\tau=0}^{t-1}\mathbb{E}\{Q(\tau)\}.$$

Rearranging terms in the above inequality and neglecting non-negative quantities where appropriate yields the following two inequalities:

$$\begin{aligned}\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}\left\{E\left(\boldsymbol{\omega}(\tau)\right)\right\} &\leq E^*+\frac{C}{V}+\frac{\mathbb{E}\left\{L\left(Q(0)\right)\right\}}{Vt},\\ \frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}\{Q(\tau)\} &\leq \frac{C+V\Big[E^*-\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}\left\{E\left(\boldsymbol{\omega}(\tau)\right)\right\}\Big]}{\varepsilon}+\frac{\mathbb{E}\left\{L\left(Q(0)\right)\right\}}{Vt}.\end{aligned}$$

Taking limits as $t\to\infty$, we derive (4.38) and (4.39). □

4.2.4 LARAC-based Algorithm

For comparison, we propose a dynamic offloading decision algorithm according to Lagrangian relaxation based aggregated cost (LARAC), which uses the concept of aggregated cost and provides an efficient method to find the optimal multiplier based on Lagrange relaxation [53].

Our objective is still the same, i.e. to find the minimum in terms of energy consumption subject to

the constraint that the total response time should not exceed the deadline T_d. A decision combination vector $\boldsymbol{\omega}(t)$ is feasible if the total response time meets the deadline. The decision combination vector $\boldsymbol{\omega}^*(t)$ with the minimum energy consumption is the optimal solution among all the feasible decision combination vectors. Mathematically, we have:

$$\min_{\boldsymbol{\omega}(t)} \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\Big\{ E\left(\boldsymbol{\omega}(\tau)\right) \Big\}, \tag{4.40}$$

$$\text{s.t.} \quad \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\Big\{ T\left(\boldsymbol{\omega}(\tau)\right) \Big\} \leq T_d. \tag{4.41}$$

Specifically, we can define a Lagrangian function as:

$$f(\lambda) = \mathbb{E}\Big\{ E\left(\boldsymbol{\omega}(t)\right) + \lambda T\left(\boldsymbol{\omega}(t)\right) \Big\} - \lambda T_d, \tag{4.42}$$

where λ is a Lagrange multiplier [143].

Using Lagrange duality principle, we obtain:

$$f(\lambda) \leq \mathbb{E}\Big\{ E\left(\boldsymbol{\omega}^*(t)\right) \Big\}, \tag{4.43}$$

which gives a lower bound for the optimal solution of the offloading policy.

Then we formulate the LARAC-based dynamic offloading decisions as shown in Algorithm 4, in order to find an optimal $\boldsymbol{\omega}^*(t)$ among all the possible offloading decision combinations. If we can find a minimum-energy combination vector that satisfies the deadline, this combination is the solution. However, if the minimum-time combination vector violates the deadline, there is no solution; otherwise we repeatedly update $\boldsymbol{\omega}^{\boldsymbol{E}}(t)$ and $\boldsymbol{\omega}^{\boldsymbol{T}}(t)$ to search for the optimal $\boldsymbol{\omega}^*(t)$ [143]. Although we cannot guarantee to find the optimal decision combination, a lower bound on the theoretical optimal solution along with the result can be achieved. The computational complexity of the LARAC-based algorithm in terms of runtime is $O(|f|^2 \log^4 |f|)$ [53], which has a higher complexity than the Lyapunov-based algorithm.

4.2.5 Simulation and Results

Since our algorithms utilize the knowledge of current states (i.e. the current network bandwidth is supposed to be known), they closely depend on the bandwidth estimation. We could use the predictors proposed in [118], which consider the classical bandwidth predictors synthetically. The framework unifies such decision models by formulating the problem as a statistical decision problem

Algorithm 4 A LARAC-based Offloading Decision Algorithm

//Find the optimal solution with offloading decision combination vector $\boldsymbol{\omega}^*(t)$
Function $\left[\boldsymbol{\omega}^*(t)\right] = LARAC\big(\mathbb{E}\big\{E\left(\boldsymbol{\omega}(t)\right)\big\}, \mathbb{E}\big\{T\left(\boldsymbol{\omega}(t)\right)\big\}, T_d\big)$
Input: $\mathbb{E}\big\{E\left(\boldsymbol{\omega}(t)\right)\big\}$: the mean energy consumption
$\mathbb{E}\big\{T\left(\boldsymbol{\omega}(t)\right)\big\}$: the mean response time
T_d: the deadline
Output: $\boldsymbol{\omega}^*(t)$: the optimal offloading decision combination vector

$\boldsymbol{\omega}^{\boldsymbol{E}}(t) = \arg\min_{\boldsymbol{\omega}(t)} \ \mathbb{E}\big\{E\left(\boldsymbol{\omega}(t)\right)\big\}$
$\boldsymbol{\omega}^{\boldsymbol{T}}(t) = \arg\min_{\boldsymbol{\omega}(t)} \ \mathbb{E}\big\{T\left(\boldsymbol{\omega}(t)\right)\big\}$
if $\mathbb{E}\big\{T\left(\boldsymbol{\omega}^{\boldsymbol{E}}(t)\right)\big\} \leq T_d$ **then**
    return $\boldsymbol{\omega}^{\boldsymbol{E}}(t)$
end if
if $\mathbb{E}\big\{T\left(\boldsymbol{\omega}^{\boldsymbol{T}}(t)\right)\big\} > T_d$ **then**
    return "There is no feasible solution"
end if
while true **do**
    $\lambda = \frac{\mathbb{E}\left\{E\left(\boldsymbol{\omega}^{\boldsymbol{E}}(t)\right)\right\} - \mathbb{E}\left\{E\left(\boldsymbol{\omega}^{\boldsymbol{T}}(t)\right)\right\}}{\mathbb{E}\left\{T\left(\boldsymbol{\omega}^{\boldsymbol{T}}(t)\right)\right\} - \mathbb{E}\left\{T\left(\boldsymbol{\omega}^{\boldsymbol{E}}(t)\right)\right\}}$
    $\boldsymbol{\omega}^*(t) = \arg\min_{\boldsymbol{\omega}(t)} \ \mathbb{E}\big\{E\left(\boldsymbol{\omega}(t)\right) + \lambda T\left(\boldsymbol{\omega}(t)\right)\big\}$
    if $\mathbb{E}\big\{E\left(\boldsymbol{\omega}^*(t)\right) + \lambda T\left(\boldsymbol{\omega}^*(t)\right)\big\} == \mathbb{E}\big\{E\left(\boldsymbol{\omega}^{\boldsymbol{E}}(t)\right) + \lambda T\left(\boldsymbol{\omega}^{\boldsymbol{E}}(t)\right)\big\}$ **then return** $\boldsymbol{\omega}^{\boldsymbol{T}}(t)$
    else
        if $\mathbb{E}\big\{T\left(\boldsymbol{\omega}^*(t)\right)\big\} \leq T_d$ **then**
            $\boldsymbol{\omega}^{\boldsymbol{T}}(t) = \boldsymbol{\omega}^*(t)$
        else
            $\boldsymbol{\omega}^{\boldsymbol{E}}(t) = \boldsymbol{\omega}^*(t)$
        end if
    end if
end while
return $\boldsymbol{\omega}^*(t)$

that can either be treated “classically" or using a Bayesian approach. However, we will not focus on bandwidth estimation here, and instead we assume the current network bandwidth is well predicted and could be directly in use.

We need to estimate the achievable bandwidths: $B(t)$, $B_1(t)$ and $B_2(t)$ at the beginning of the t^{th} execution, and they stay the same during each execution. We suppose that $B(t)$, $B_1(t)$ and $B_2(t)$ follow uniform distributions on [1, 200], [1, 400] and [1, 500] Kbps, respectively.

Static Offloading Decisions

We first consider static offloading decisions based on criteria in (4.20) or (4.21). For convenience, suppose D stays the same in each execution, $T_n^{\text{local}} = 100$ s, $T_n^{\text{s}} = 10$ s and $T_n^{\text{c}} = 10$ s, where $n \in \{0, \cdots, N\}$. According to the power models developed in [10], we set the system parameters to: $N = 4$, $p_{\text{m}} = 0.3$ W, $p_{\text{i}} = 0.03$ W and $p_{\text{tr}} = 0.2$ W.

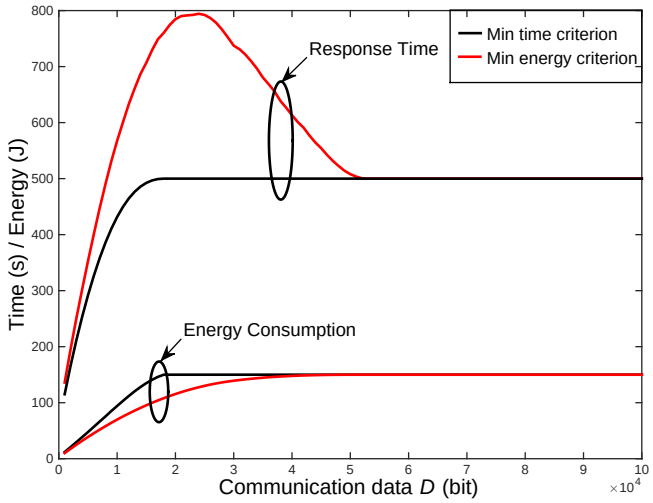


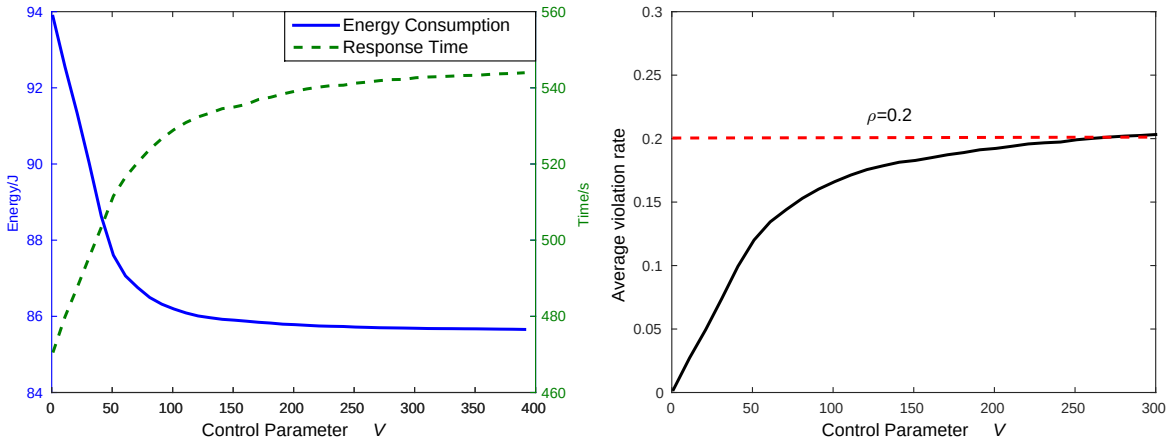
Figure 4.9: The impact of communication data under different offloading decision criteria

It can be observed from Fig. 4.9 that when we choose the minimum response time criterion, at first the energy consumption and response time rise with increasing of the communication data D. Then both do not change any more with the increase of D, since there is no benefit from offloading in this case. Therefore, all the application tasks are executed locally. However, when the minimum energy criterion is selected, the total response time first increases, then arrives at a peak, and subsequently decreases to a stable value which is the same as for the time executed locally. According to Fig. 4.9, we can not achieve optimal energy consumption and response time together. Actually, there is a tradeoff between the mean energy consumption and mean response time that can be further explored

in dynamic offloading decision algorithms.

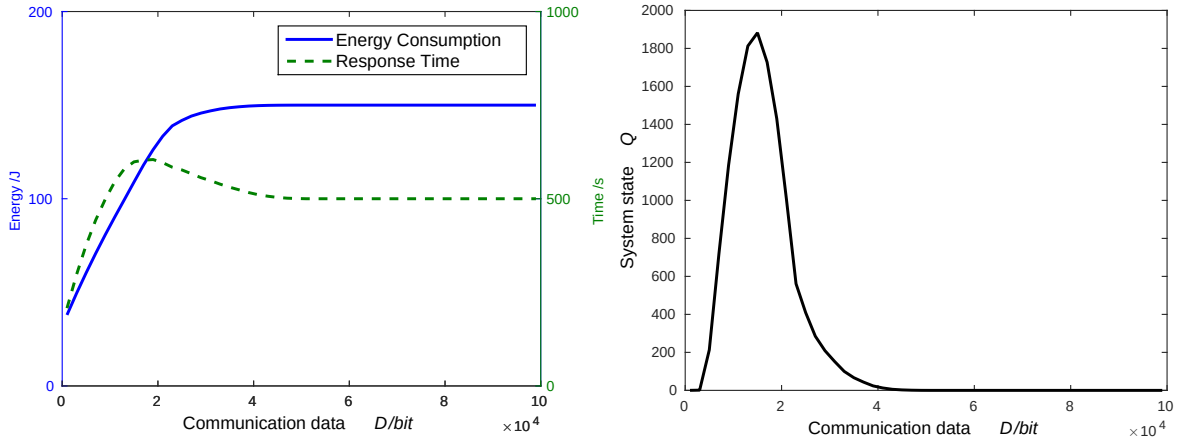
Dynamic Offloading Decisions

We assume that the communication data between different tasks is $D_{uv} = 10$ Kbits, the violation ratio $\rho = 0.2$, the deadline $T_d = 600$ s and the other parameters are the same as in the previous section. We simulate our algorithm in 10^4 times for each value of V ranging from 1 to 400.



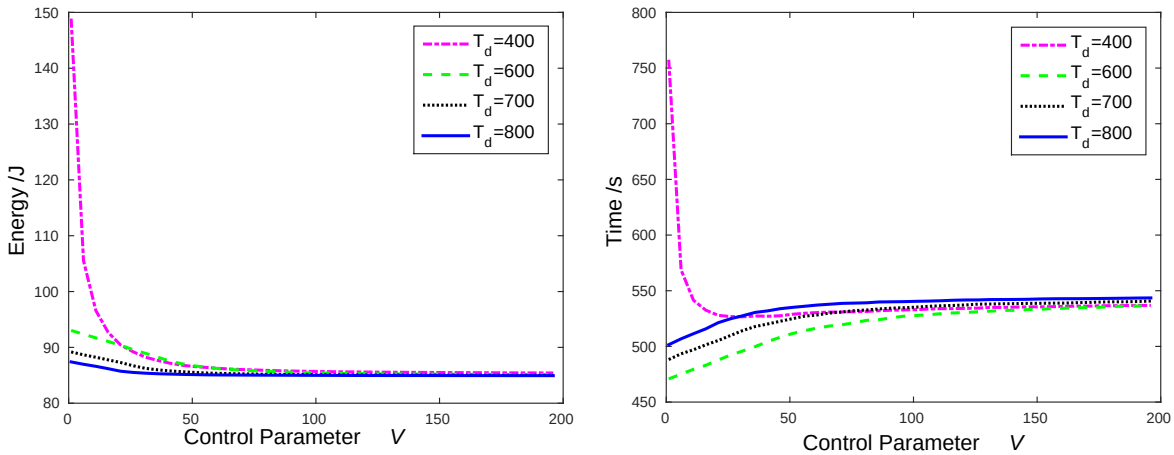
(a) Energy consumption and response time (b) Violation rate

Figure 4.10: The impact of V on average energy consumption, response time and violation rate



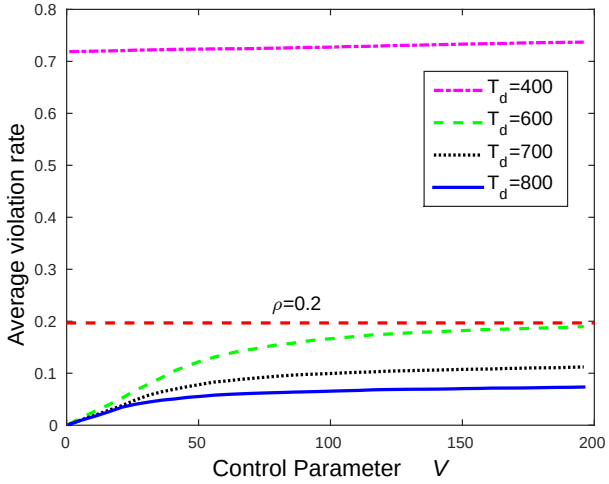
(a) Energy consumption and response time (b) System state

Figure 4.11: The impact of communication data on average energy consumption, response time and system state, when $V = 100$



(a) Energy consumption

(b) Response time



(c) Violation rate

Figure 4.12: The impact of parameter V under different deadlines

As depicted in Fig. 4.10(a), the energy consumption falls quickly at the beginning and then tends to descend slowly while the response time grows linearly with V at first and then increases slowly. This finding confirms that there is a $[O(1/V), O(V)]$ tradeoff between average energy consumption and average response time. A good operating point would be to pick a V value where a unit increase in V yields a very small reduction in $\bar{Q}$. At such point, the energy gains may not be worth the response time rising from the increase of V [103]. There exists a sweet spot of value V (e.g. $V = 100$) beyond which increasing V leads to a marginal energy conservation yet consistently growing delays. In Fig. 4.10(b) the average violation ratio $\mathbb{E}\{\sigma(\boldsymbol{\omega}(t))\}$ first grows linearly with V

and then increases slowly, finally it approaches to a fixed ratio $\rho = 0.2$, denoted by the dotted red line. The queuing system state is stable, since $\mathbb{E}\{\sigma(\boldsymbol{\omega}(t))\} \leq \rho$ that satisfies (4.31).

In Fig. 4.11(a) the average energy consumption increases with the increase of D, while the average response time has a peak and then it decreases again. However, there is no benefit from offloading when D is very large, and thus all the application tasks are executed locally in this case. From Fig. 4.11(b), when D is large enough (e.g. $D \geq 45$ Kbit), the system queue state is always 0, which means $T(\boldsymbol{\omega}(t)) \leq T_d$, and all the tasks are executed locally. This is because the transmission time is so large that it dominates the response time. Then we would rather perform the computation on the mobile device than to offload it to the remote cloud.

As the average violation rate is much higher than the constant $\rho = 0.2$ denoted by the red dotted line in Fig. 4.12(c), the system is unstable when $T_d = 400$. Therefore, we do not consider this situation, since the result under such deadline is unreasonable. From Fig. 4.12(a), it can be seen that the average energy consumption decreases with increasing T_d when V is small, while the average response time increases as T_d rises from 600 to 800 in Fig. 4.12(b). Therefore, setting the deadline a little larger can reduce the average energy consumption but leads to the increase of average response time.

Comparison of Different Decision Schemes

To gain insight on the performance of the proposed energy-efficient dynamic offloading decision algorithm, we compare the results using the following methods:

- *Local scheme*: all application tasks are executed locally on the mobile device.
- *Cloud scheme*: all offloadable application tasks are directly offloaded to the cloud for further processing.
- *Cloudlet scheme*: all offloadable application tasks are offloaded via the cloudlet to the cloud for further processing.
- *Lyapunov scheme*: using the Lyapunov-based dynamic offloading decision algorithm (e.g. $V = 100$).
- *LARAC scheme*: using the LARAC-based dynamic offloading decision algorithm.

Figure 4.13 shows the average response time and energy consumption, normalized to the local scheme. The red dotted line denotes the deadline. It can be seen that our proposed Lyapunov scheme can help to save around 50% of the energy consumption compared to the local scheme while only sacrificing a small portion of response time. This is because the Lyapunov scheme offloads tasks dynamically according to network bandwidth and transmit power, while both the cloud scheme and the cloudlet scheme do not take the network bandwidth into consideration. Especially, when the

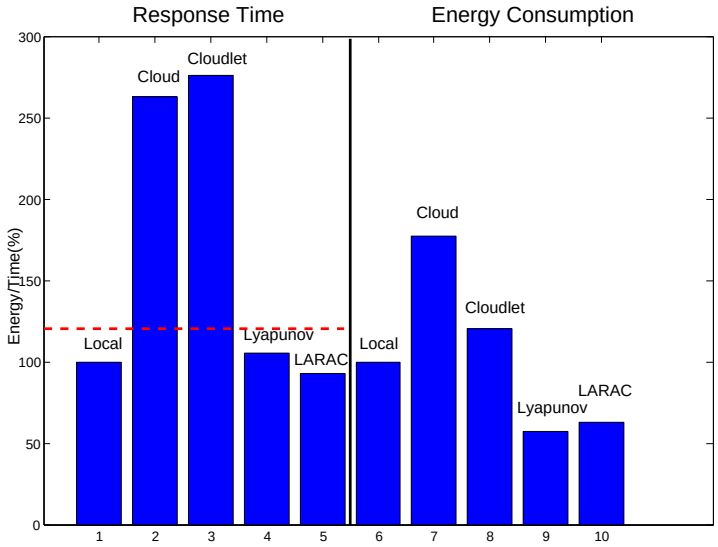


Figure 4.13: Comparison of mean response time and energy consumption under different schemes

network bandwidth is so low that offloading tasks to the cloud or via the cloudlet to the cloud may not be beneficial. Besides, when compared with the LARAC-based algorithm, the Lyapunov scheme also saves more energy while only sacrificing a small portion of response time.

4.3 Performance Analysis of Offloading Systems

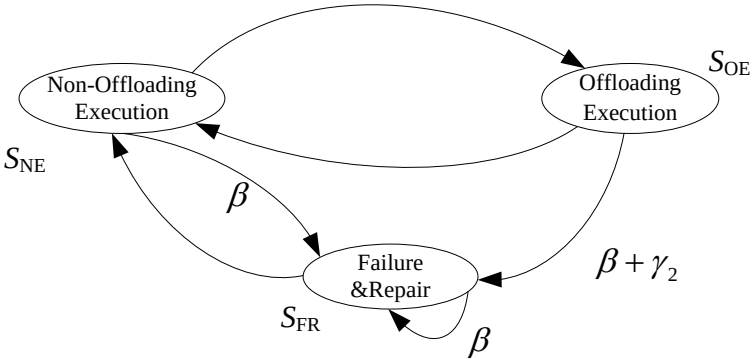
Offloading critically depends on a reliable end-to-end communication and on the availability of the cloud. In addition, it suffers from high network access latencies and low network bandwidth. We want to investigate how effective and efficient they are and what factors influence their performance. With this purpose, we introduce a mathematical model and analyze offloading systems with failures, considering application execution time and failure recovery time.

4.3.1 Offloading Systems with Failures

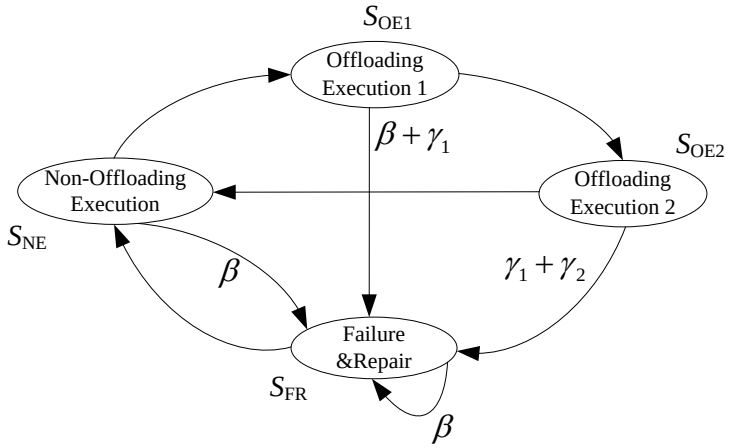
The state transitions of offloading systems in the presence of failures are given as Fig. 4.14. There are four states as depicted in Fig. 4.14(b), named non-offloading execution (S_{NE}), offloading execution 1 on cloudlet (S_{OE1}), offloading execution 2 on cloud (S_{OE2}), and failure and repair (S_{FR}), respectively. After triggering the begin of offloading event, the execution state changes from S_{NE} to S_{OE1}. If the cloud is available, without hesitation, the execution state continues to S_{OE2}. Once the executions of all the offloaded tasks are finished successfully, the state changes back from S_{OE2} to S_{NE} [90].

However, failures may occur in all the four states:

- S_{NE}: failures can occur in state S_{NE} due to the mobile device's failures (e.g. running out of battery, abnormal shutdown).
- S_{OE1}: failures may occur in state S_{OE1} due to the cloudlet's failures (e.g. cloudlet's shutdown, wireless link failures) or the cloudlet is becoming unreachable owing to the mobile device's movement.
- S_{OE2}: failures can occur in state S_{OE2} due to the cloud's failures (e.g. unavailable or outage of the cloud) or the cloud is becoming unreachable due to cloudlet failures.
- S_{FR}: nested failures may also happen in state S_{FR}.



(a) Two-level offloading systems [90]



(b) Three-level offloading systems

Figure 4.14: State transitions of offloading systems with failures

Once a failure event is triggered, the application's execution state transits to the failure and repair state S_{FR}. The offloading execution is completed when an execution period elapses without failure. When a failure occurs, a period of time R for repairing will be taken. Independent failures caused by the mobile device, cloudlet and cloud are modelled as non-homogenous Poisson processes with different rates β, γ_1 and γ_2, respectively. In state S_{NE}, only the failures caused by the mobile device lead to a state change to S_{FR}; in state S_{OE1}, any failure caused by the mobile device or the cloudlet can induce a state transition to S_{FR}; in the state S_{OE2}, any failure caused by the cloudlet or cloud can induce a state turn to S_{FR}, and in state S_{FR}, the rate of nested failures is β. Therefore, the failure rate function $\lambda(t)$ is a discrete function of time and defined as:

$$\lambda(t) = \begin{cases} \beta, & \text{in state } S_{\text{NE}} \text{ and } S_{\text{FR}}, \\ \beta + \gamma_1, & \text{in state } S_{\text{OE1}}, \\ \gamma_1 + \gamma_2, & \text{in state } S_{\text{OE2}}. \end{cases} \tag{4.44}$$

Failure Repair Time

In the offloading systems without fault-tolerance, a failure repair time is a period spent to re-execute the application from the beginning to the point of the failure. Once an application enters state S_{FR}, the time period R is required to complete a repair time in the presence of nested failures. The expectation of the failure repair time is given by:

$$\mathbb{E}(R^*) = \frac{1}{\beta}\left[\frac{1}{\mathbb{E}(\mathrm{e}^{-\beta R})} - 1\right]. \tag{4.45}$$

Proof. Let X be the random variable denoting the time to the first failure after starting repair, let R^* be the failure repair time in the presence of failures within the repair period R, then we get:

$$R^* = \begin{cases} R, & \text{if } R < X, \\ X + \tilde{R}^*, & \text{if } R \geq X. \end{cases}$$

If $R < X$, then a repair will be successful completed without nested failures; otherwise, if $R \geq X$, a failure occurs after which another repair is simply repeated, denoted as $\tilde{R}^*$. Thus, the repair time is $X + \tilde{R}^*$ in this case [18]. Taking conditional expectation of R^*, we have:

$$\mathbb{E}(\mathrm{e}^{-sR^*}|R, X) = \begin{cases} \mathrm{e}^{-sR}, & \text{if } R < X, \\ \mathrm{e}^{-sX}\mathbb{E}(\mathrm{e}^{-s\tilde{R}^*}), & \text{if } R \geq X. \end{cases}$$

Since X is independent of $\tilde{R}^*$, un-conditioning on X, we get:

$$\begin{aligned}\mathbb{E}(e^{-sR^*}|R) &= \int_{x=0}^{\infty} \mathbb{E}(\mathrm{e}^{-sR^*}|R, X=x)\beta \mathrm{e}^{-\beta x}\mathrm{d}x \\ &= \int_{x=0}^{R} \mathrm{e}^{-sx}\mathbb{E}(\mathrm{e}^{-sR^*})\beta \mathrm{e}^{-\beta x}\mathrm{d}x + \int_{x=R}^{\infty} \mathrm{e}^{-sR}\beta \mathrm{e}^{-\beta x}\mathrm{d}x \\ &= \frac{\beta E(\mathrm{e}^{-sR^*})(1-\mathrm{e}^{-(s+\beta)R})}{s+\beta} + \mathrm{e}^{-(s+\beta)R}.\end{aligned}$$

Further, removing the condition on R, it yields:

$$\begin{aligned}\int_{R=0}^{\infty} \mathbb{E}(e^{-sR^*}|R)f(R)\mathrm{d}R &= \int_{R=0}^{\infty} \left[\frac{\beta\mathbb{E}(\mathrm{e}^{-sR^*})(1-\mathrm{e}^{-(s+\beta)R})}{s+\beta} + \mathrm{e}^{-(s+\beta)R}\right] f(R)\mathrm{d}R, \\ \mathbb{E}(e^{-sR^*}) &= \int_{R=0}^{\infty} \frac{\beta\mathbb{E}(\mathrm{e}^{-sR^*})(1-\mathrm{e}^{-(s+\beta)R})}{s+\beta} f(R)\mathrm{d}R + \int_{R=0}^{\infty} \mathrm{e}^{-(s+\beta)R} f(R)\mathrm{d}R,\end{aligned}$$

and by using the Laplace transform:

$$L_R(s) = \int_{R=0}^{\infty} \mathrm{e}^{-sR} f(R)\mathrm{d}R = \mathbb{E}(\mathrm{e}^{-sR})$$

on both sides, respectively, we get:

$$L_{R^*}(s) = \frac{\beta L_{R^*}(s)(1-L_R(s+\beta))}{s+\beta} + L_R(s+\beta).$$

Rearranging the above equation, we have:

$$L_{R^*}(s) = \frac{(s+\beta)L_R(s+\beta)}{\beta L_R(s+\beta)+s}.$$

According to the property of Laplace transform [29], the expected repair time in the presence of failures can be calculated as:

$$\mathbb{E}(R^*) = -\frac{\mathrm{d}L_{R^*}(s)}{\mathrm{d}s}|_{s=0},$$

and then we derive (4.45). □

Application Response Time

With the presence of failures, the application response time can be approximated by:

- For two-level offloading systems:

$$T_{\text{FT}}(n) = (1-\alpha_2)\cdot\mathbb{E}[T_{\text{OE}}(n)] + \alpha_2\mathbb{E}[T_{\text{NE/FR}}(n)],$$

where α_2 is the unreachable probability of cloud, the time spent in state S_{OE} and in state S_{NE}/S_{FR} can be calculated as:

$$\begin{aligned}\mathbb{E}[T_{\text{OE}}(n)] &= \left[\frac{1}{\beta+\gamma_2}+\mathbb{E}(R^*)\right]\cdot\left[\mathrm{e}^{(\beta+\gamma_2)\mathbb{E}[T_{\text{OPT}}(n)]}-1\right],\\ \mathbb{E}[T_{\text{NE/FR}}(n)] &= \left[\frac{1}{\beta}+\mathbb{E}(R^*)\right]\cdot\left(\mathrm{e}^{\beta\mathbb{E}[T_{\text{OPT}}(n)]}-1\right).\end{aligned}$$

- For three-level offloading systems:

$$\begin{aligned}T_{\text{FT}}(n) &= \frac{1}{2}\Big\{(1-\alpha_1)\mathbb{E}[T_{\text{OE1}}(n)] + \alpha_1\mathbb{E}[T_{\text{NE/FR}}(n)] + (1-\alpha_2)\mathbb{E}[T_{\text{OE2}}(n)] + \alpha_2\mathbb{E}[T_{\text{NE/FR}}(n)]\Big\}\\ &= \frac{1}{2}\Big\{(1-\alpha_1)\mathbb{E}[T_{\text{OE1}}(n)] + (1-\alpha_2)\mathbb{E}[T_{\text{OE2}}(n)] + (\alpha_1+\alpha_2)\mathbb{E}[T_{\text{NE/FR}}(n)]\Big\},\end{aligned}$$

where α_1 is the unreachable probability of cloudlet, the time spent in state S_{OE1} and S_{OE2} can be calculated as:

$$\begin{aligned}\mathbb{E}[T_{\text{OE1}}(n)] &= \left[\frac{1}{\beta+\gamma_1}+\mathbb{E}(R^*)\right]\cdot\left[\mathrm{e}^{(\beta+\gamma_1)\mathbb{E}[T_{\text{OPT}}(n)]}-1\right]\\ \mathbb{E}[T_{\text{OE2}}(n)] &= \left[\frac{1}{\gamma_1+\gamma_2}+\mathbb{E}(R^*)\right]\cdot\left[\mathrm{e}^{(\gamma_1+\gamma_2)\mathbb{E}[T_{\text{OPT}}(n)]}-1\right]\end{aligned}$$

4.3.2 Analytical Evaluation

In order to analyze the offloading system with failures, we can list different types of response time as follows:

- $T_{\text{m}}(n)$: the response time when the whole application is executed on the mobile device without the presence of offloading and failure repair.
- $T_{\text{OPT}}(n)$: the response time when the application runs in an ideal offloading system without failures [92], it can be calculated as:

$$T_{\text{OPT}}(n) = (1-\pi)\cdot T_{\text{m}}(n) + \frac{\pi\cdot T_{\text{m}}(n)}{F} + T_{\text{c}}(n) = \left[1-\frac{(F-1)\pi}{F}\right]\cdot T_{\text{m}}(n) + T_{\text{c}}(n), \quad (4.46)$$

where $T_{\mathrm{c}}(n) = \frac{D}{B_1} + \frac{D}{B_2}$ is the total communication time, $0 \leq \pi \leq 1$ is the proportion of sub-tasks performed by the cloud compared to the mobile device, $(1-\pi) \cdot T_{\mathrm{m}}(n)$ denotes the time spent on the cloud server for performing $(1-\pi)n$ sub-tasks and $\pi \cdot T_{\mathrm{m}}(n)/F$ denotes the time spent on the mobile device for executing πn sub-tasks. $T_{\mathrm{OPT}}(n)$ is monotonically decreasing with F and π since:

$$\frac{\partial T_{\mathrm{OPT}}(n)}{\partial F} = -\frac{1}{F^2} \cdot \pi \cdot T_{\mathrm{m}}(n) < 0 \quad \text{and} \quad \frac{\partial T_{\mathrm{OPT}}(n)}{\partial \pi} = -\frac{F-1}{F} \cdot T_{\mathrm{m}}(n) < 0.$$

- $T_{\mathrm{FT}}(n)$: the response time when the application is totally offloaded to the cloud and executed with failure repair. Once a failure occurs, it needs to restart from scratch. $T_{\mathrm{FT}}(n)$ is also monotonically decreasing with F and π since:

$$\frac{\partial T_{\mathrm{FT}}(n)}{\partial F} = -\frac{1}{F^2} \cdot C_1 < 0 \quad \text{and} \quad \frac{\partial T_{\mathrm{FT}}(n)}{\partial \pi} = -\frac{F-1}{F} \cdot C_2 < 0,$$

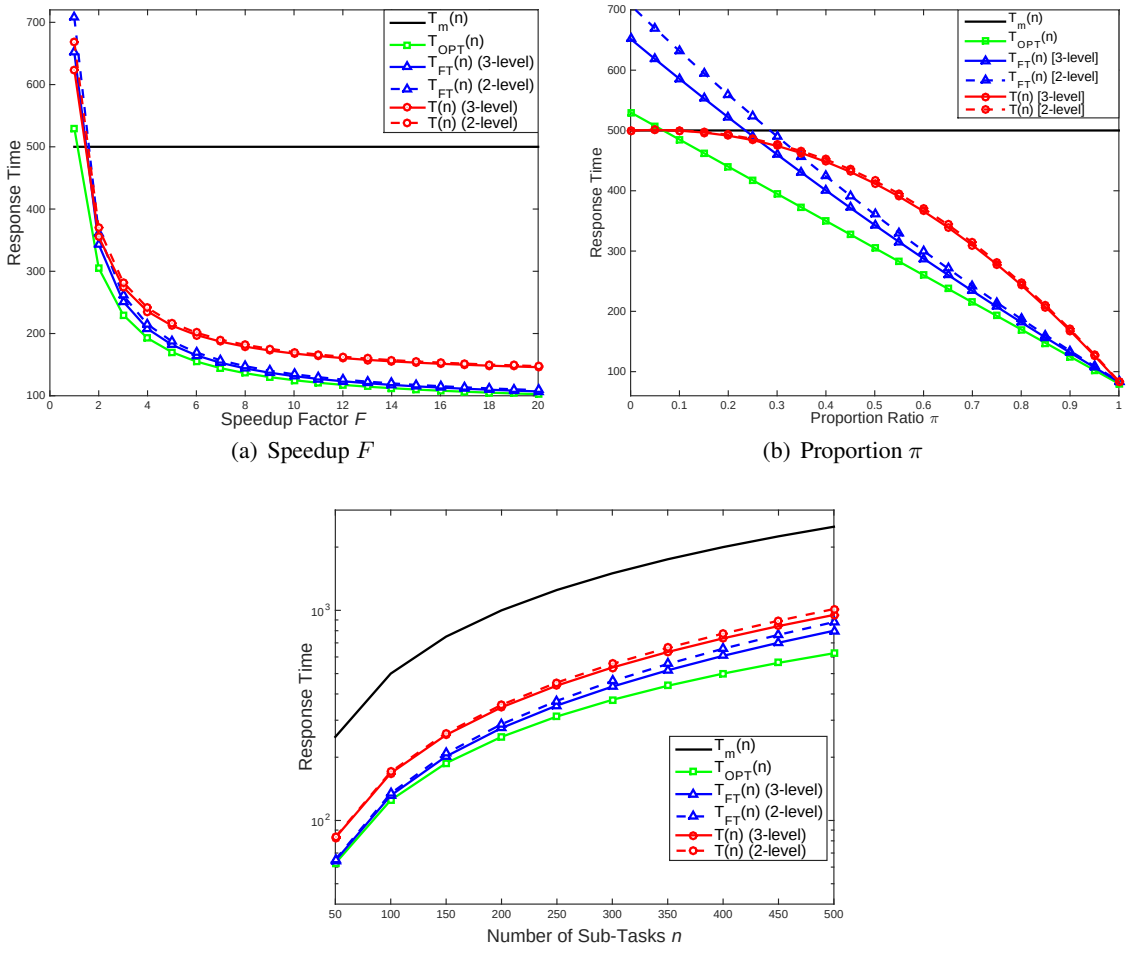
  where C_1 and C_2 are positive constants independent of F and π, respectively.
- $T(n)$: partial sub-tasks are offloaded to the cloud through cloudlet and the others are executed by the mobile device. Only part of sub-tasks that are executed by the cloud need to be re-executed when a failure occurs. Thus, we have:

$$T(n) = T_{\mathrm{m}}[(1-\pi)n] + T_{\mathrm{FT}}(\pi n), \tag{4.47}$$

  where $T_{\mathrm{m}}[(1-\pi)n]$ and $T_{\mathrm{FT}}(\pi n)$ are the time spent on the mobile device and cloud, respectively. If π is close to 1, $T(n)$ reduces to $T_{\mathrm{FT}}(n)$, and otherwise if π approaches to 0, $T(n)$ is close to $T_{\mathrm{m}}(n)$.

Parameters are set as follows: the number of sub-tasks $n = 100$, the average period of repair time $\mathbb{E}(R) = 10$, the proportion of sub-tasks performed by the cloud compared to the mobile device $\pi = 0.9$, the failure rate of the mobile device, the cloudlet and the cloud are $\beta = 10^{-3}$, $\gamma_1 = 10^{-4}$ and $\gamma_2 = 10^{-5}$, respectively. The probability of unreachability of cloudlet and cloud are $\alpha_1 = 0.1$ and $\alpha_2 = 0.2$, respectively, the average execution time for each sub-task on the mobile device is 5 seconds and the communication time $T_{\mathrm{c}} = 0.3n$ is proportional to n.

In Fig. 4.15(a) the curve $T_{\mathrm{m}}(n)$ is a horizontal line at 500 s, and in addition to $T_{\mathrm{m}}(n)$, the response time decreases along with the increase of speedup factor F. $T_{\mathrm{FT}}(n)$ and $T(n)$ produce a much longer time than $T_{\mathrm{m}}(n)$ when $F < 2$, due to offloading and failure repairs. With larger F, the cloud can save response time on the mobile device. However, the time spent on offloading operation and failure handling will increase the total response time, especially in environments with less higher cloudlet



(a) Speedup F

(b) Proportion π

(c) Sub-tasks n

Figure 4.15: Response time under different parameters

or cloud reachability. And also an offloading process may not be able to reduce the application response time under small F. Further, the three-level offloading scheme costs less time than the two-level approach since it is much more stable and reliable.

Figure 4.15(b) shows the response time under different π when $F = 10$. It can be seen that $T(n)$ reduces to $T_{\mathrm{m}}(n)$ when $\pi = 0$, where the tasks are totally executed by the mobile device, and $T(n)$ approaches $T_{\mathrm{FT}}(n)$ when $\pi = 1$, where the tasks are totally migrated to the cloud. $T_{\mathrm{FT}}(n)$ and $T_{\mathrm{OPT}}(n)$ decrease along with the increase of π, while $T(n)$ increases slightly and decreases afterwards. The response time of $T_{\mathrm{FT}}(n)$ and $T(n)$ are greater than $T_{\mathrm{m}}(n)$ when $\pi < 0.66$ due to offloading and failure recoveries. The larger π is, the more time the three-level offloading system

saves and it works better than the two-level offloading approach owing to its higher reliability.

Figure 4.15(c) considers the effect of the number of sub-tasks n on the application response time. The larger n is, the more time the three-level offloading system costs. Besides, a failure handling process adds extra time to the system. Further, the time saved by the three-level offloading system from the two-level offloading scheme raises along with increasing of n.

4.4 Summary

To sum up, we have discussed the issue of where to offload in offloading decision making.

First we tried to find the optimal cloud service among several alternative clouds, which is suitable for an offloading destination. A scheme that combines AHP and fuzzy TOPSIS methods is proposed, which considers the subjective judgments of evaluators and makes a final decision based on the results from multi-criteria decision analysis. It is proposed as an effective way to solve optimal cloud service selection through numerical analysis.

Then we presented an approach for dynamic offloading decision making based on different criteria and particularly considered the changing landscape of network connectivity (cellular network vs. WiFi to cloud vs. cloudlet). We formulated an optimization problem whose solution can guide the required decision making, which minimizes the total energy usage subject to keeping the average queue length finite. It is able to partition individual portions of the offloading task pool into different groups, each with very specific combinations of offloadable characteristics. Numerical results show that the proposed algorithm can save around 50% of the energy consumed as compared with local execution while only slightly sacrificing response time, and it has less computational complexity than the LARAC-based algorithm.

Finally, we compared the performance of different offloading systems. In the environments with less cloudlet or cloud reachability, longer disconnection time or even smaller speedup factor F, the three-level scheme will not benefit from offloading. However, under the same reachability and failure rate of the cloud, the three-level offloading scheme is much more stable and reliable than the two-level offloading approach and thus it reduces response time.

Chapter 5

Offloading Decision Making: How to Offload

Offloading can be performed statically or dynamically via different wireless networks like WLAN and cellular networks. Since the transmission techniques differ in energy requirements and speeds, we should determine how to leverage the complementary strength of WiFi and cellular networks by choosing heterogeneous wireless interfaces for offloading. This chapter addresses the issue of how to offload by minimizing both the energy consumption and response time, with the contributions being threefold:

- Using queueing models to carry out optimality analysis of the energy-performance tradeoff for mobile offloading systems, which captures both energy and performance metrics and also intermittently available access links.
- Applying different offloading policies (static and dynamic), where arriving jobs are processed either locally or remotely. The dynamic offloading policy considers the increase in each queue and the change in metric that newly arriving jobs bring in should they be assigned to that queue, while the static policy does not capture the dynamic increase.
- Developing two different offloading strategies: the uninterrupted strategy, which uses WiFi whenever possible, but switches to a cellular interface if no WiFi connection exists [77], and data is continuously transmitted while switching between different channels; the interrupted strategy, which assigns jobs upon arrival to one of two parallel queues which describe cellular or WiFi transmission. Data transmission of the WiFi queue can be interrupted for short periods when the connection is lost.

5.1 The Queueing Model

We consider a queueing model for mobile offloading systems as depicted in Fig. 5.1. The mobile device, the cloud and the wireless networks are represented as queueing nodes to capture the resource contention and delay on these systems [16]. The mobile device executes an application with different types of jobs that can be classified into: unoffloadable and offloadable. The problem of taking offloading decisions does not exist for the unoffloadable jobs. However, for the offloadable ones, the mobile device should judiciously make decisions that optimize the response time energy tradeoff expressed in one of the metrics defined in Section 2.2.2.

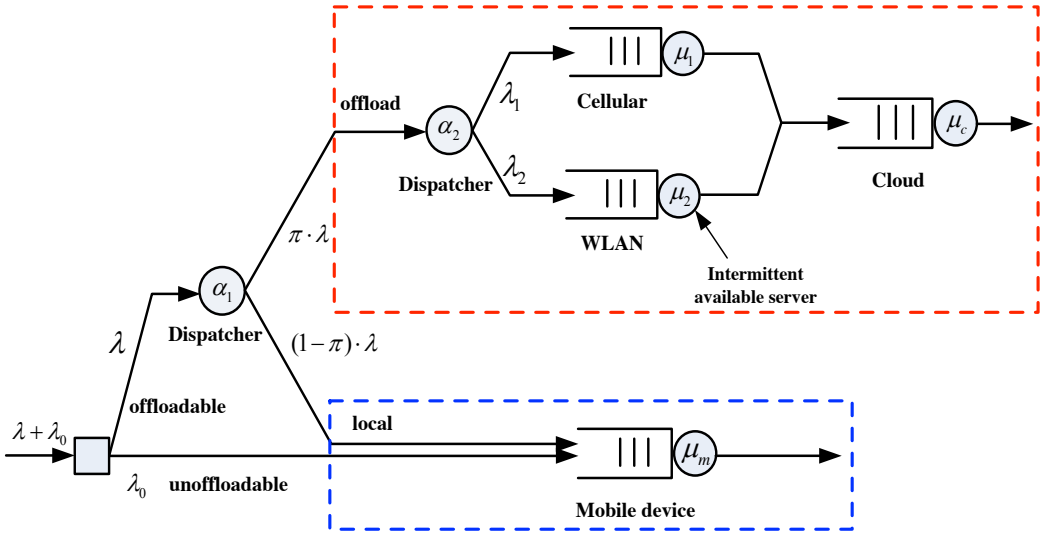


Figure 5.1: A queueing model for mobile cloud offloading systems

As indicated in Fig. 5.1, job arrivals at the mobile device are assumed to follow a Poisson process with an average arrival rate of $\lambda + \lambda_0$, where λ and λ_0 are the rates of offloadable and unoffloadable jobs, respectively. The arrival rate is based on the behavior of the application. The unoffloadable jobs with an arrival rate λ_0 are unconditionally executed locally. As for the offloadable ones with an arrival rate λ, the mobile device chooses to offload each job with a probability $0 \leq \pi \leq 1$. According to the properties of the Poisson distribution [94], the jobs are offloaded to the cloud following a Poisson process with an average arrival rate of $\lambda_c = \pi \cdot \lambda$, the offloading rate. Similarly, jobs that are proceed locally instead of being offloaded follow a Poisson process with rate $\lambda_m = (1 - \pi) \cdot \lambda$.

There are several ways to offload computation to the cloud, either via a cellular connection or an available WLAN access point (AP). According to the mobile network traces collected in Section

2.2.1, the cellular interface can provide near-ubiquitous coverage for mobile devices in a wide area, but has lower data transmission rate and consumes more transmission energy than the WiFi interface, while the WiFi network experiences frequently intermittent connectivity. The mobile device, the cellular and WLAN connections are modelled as $M/G/1$ queues, and the remote cloud is modelled as an $M/G/\infty$ queue, i.e. as a delay center [16]. We denote $1/\mu_m$ and $1/\mu_c$ the expected execution time of jobs on the mobile device and the cloud, respectively. The expected rates to transfer data to the cloud over the cellular network and WLAN are μ_1 and μ_2, respectively.

Two dispatchers are needed: α_1 is used to allocate the offloadable jobs either to the cloud or to the mobile device, while α_2 is to offload the jobs either via a cellular connection or a WLAN network to the cloud. It should be noted that when $\pi = 0$, all the offloadable jobs are processed locally, when $\pi = 1$, they are all offloaded to the cloud. The total cost, in terms of energy or response time for processing all offloadable jobs, is composed of the remote cost (sending some jobs to the cloud and waiting for the cloud to complete them), and the local cost (processing the remaining jobs locally on the mobile device).

Since the delay caused by the transmission in the uplink usually dominates the total cost, our analysis focuses on the dispatcher inside the red dotted block, which is responsible for selecting the best transmission channel. The objective is to minimize a weighted sum of the mean energy consumption and response time under the assumption that the dispatcher is aware of the remaining service time of each job in the system, including that of the arriving job [95].

5.2 Offloading Policies

In this section, we derive three offloading policies under the ERWS metric, defining different strategies to determine λ_1 and λ_2, i.e. to assign jobs upon arrival to one of two parallel queues which describe cellular or WLAN transmission. We consider both static and dynamic offloading policies:

- For static offloading policy, the offloading decision is independent of the system's state. It only depends on the job itself (size, value, class), neither on past decisions nor current state of the queues. Therefore, it only finds an overall optimal assignment scheme but could not capture the dynamic situation when the mobile environment changes.
- For dynamic offloading policy, the offloading decision depends on the system's state, it takes the new job into account when considering the following questions:
  - Whether to join the shortest queue or not?
  - What if some server is slower than another?
  - What if some server is not available?

Since the decision is made dynamically at runtime to adapt to different operating conditions, the dynamic offloading policy can achieve much better performance than the static one, but it requires a higher overhead during execution.

5.2.1 Model and Problem Formulation

The interface selection problem in offloading systems is modelled as the decision to which queue an arriving job should be assigned. In this decision the offloading dispatcher takes both performance and energy costs into account. This seems natural for heterogeneous servers where a job needs a different amount of energy and time to be served by different servers [75]. For example, whereas assigning each job to a low power server would be beneficial from the energy consumption perspective at low loads, such a policy may end up in difficulties at higher loads as the response time increases rapidly [95]. Indeed, the energy-performance tradeoff takes on different explicit expressions and under different offloading policies. Our objective is to minimize the ERWS metric under static and dynamic offloading policies.

As shown in Fig. 5.2, we consider a queuing model that consists of two parallel queues of cellular and WiFi with work conserving queuing disciplines. λ_1 and λ_2 are the mean rates of jobs into Queue 1 and Queue 2. Since the original offloading jobs arrive at the system according to a Poisson process with rate λ_c, one policy would be randomly assigned to Queue i ($i \in \{1, 2\}$), which results in independent Poisson processes with rate λ_i, and we have $\lambda_1 + \lambda_2 = \lambda_c$.

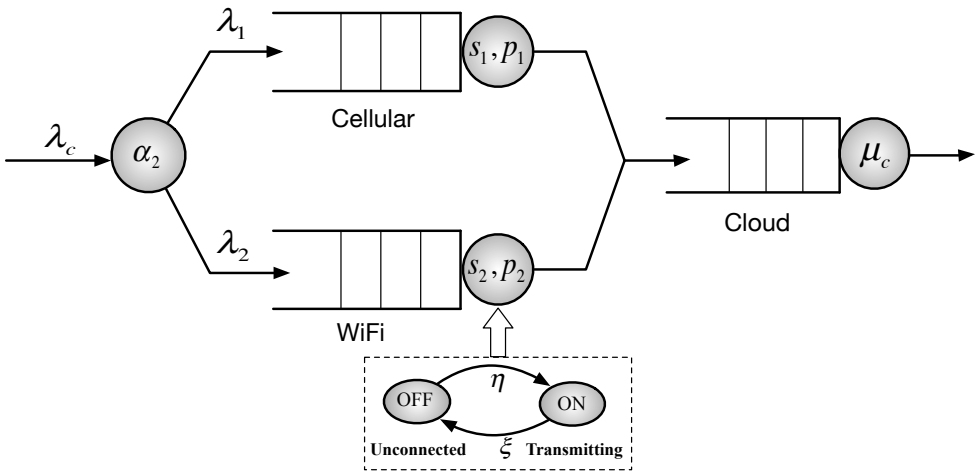


Figure 5.2: The interrupted offloading strategy

The two queues have the following characteristics:

- **Queue 1**: When a job is offloaded to the cloud via a cellular network (2G/3G), there is queue-

ing due to the transmission speed of the cellular link. Costs arise in terms of transmission delays (queueing and actual transmission time) and transmission energy consumption. We assume that the service speed (or transmission rate) is s_1, and the operating power for Server 1 is p_1 when serving jobs and zero whenever idle. Further, Server 1 is always available since the cellular connection is always on.

- **Queue 2**: When a job is offloaded to the cloud via a WLAN network, there is queueing due to the transmission speed of the WLAN link. We assume that Server 2 runs at speed s_2, and its operating power is p_2 when serving jobs and zero whenever idle. We model the intermittent availability of hotspots as a first come first served (FCFS) queue with occasional server break-down [50]. The availability of Server 2 is governed by an interrupted Poisson process (IPP) with exponentially distributed ON-OFF periods. Specifically, the server is either in ON-state processing the existing jobs, or in OFF-state during which no job receives service. We assume that the sojourn time in a hotspot and the time to move from one hotspot to another are exponentially distributed with parameters ξ (failure rate), and η (recovery rate), respectively.

These imply that WiFi is much faster and more energy-efficient than the cellular interface for transmitting the same quantity of data. Therefore, we consider here a simple scenario where the transmission rate of the cellular network is smaller than that of WLAN, i.e. $s_1 < s_2$ and the power consumption when transmitting jobs via the cellular link is larger than the WLAN link, i.e. $p_1 > p_2$.

In minimizing the ERWS metric, it seems favorable to assign jobs to Queue 2 rather than to Queue 1. However, when considering the WLAN link's intermittent availability in Queue 2, the optimal assignment has to be reconsidered.

Upon arrival of a job an assignment decision is made and the job is placed into the corresponding queue according to the ERWS metric denoted as:

$$ERWS \quad = \quad \frac{1}{\lambda_c} \sum_{i=1}^{2} \lambda_i \Big\{ \omega \mathbb{E}[\mathcal{E}_i] + (1-\omega)\mathbb{E}[T_i] \Big\}, \tag{5.1}$$

where λ_c is the total job arrival rate for offloading, λ_i is the mean rate of jobs into Queue i, $\mathbb{E}[\mathcal{E}_i]$ and $\mathbb{E}[T_i]$ are the mean energy consumption and mean response time in Queue i, respectively.

We assume that each server operates at a constant power p_i whenever the server is busy. Since the average consumed power is $\mathbb{E}[P_i] = \lambda_i \mathbb{E}[\mathcal{E}_i]$ [7], further by Little's Law, $\mathbb{E}[N_i] = \lambda_i \mathbb{E}[T_i]$, the ERWS metric in (5.1) can be more conveniently expressed by the Power-Queue length Weighted

Sum (PQWS) metric:

$$\begin{aligned} PQWS &= \frac{1}{\lambda_c}\sum_{i=1}^{2}\Big\{\omega\mathbb{E}[P_i] + (1-\omega)\mathbb{E}[N_i]\Big\} \\ &= \frac{1}{\lambda_c}\sum_{i=1}^{2}\Big\{\omega p_i \Pr\{N_i > 0, e_i = 1\} + (1-\omega)\mathbb{E}[N_i]\Big\}, \end{aligned} \tag{5.2}$$

where PQWS is a weighted sum of the mean power consumption and mean queue length (or average number of jobs) in a queueing system, Pr is the probability operation, $e_i = 1$ indicates that Server i is available and N_i is the number of jobs in Queue i.

Since the number of jobs in Queue i can be dependent on the state of Server i, we have:

$$\Pr\{N_i > 0, e_i = 1\} = \Pr\{N_i > 0 | e_i = 1\} \cdot \Pr\{e_i = 1\}. \tag{5.3}$$

Further, since Server 1 is always available, we have $\Pr\{e_1 = 1\} = 1$ and $\Pr\{N_1 > 0 | e_1 = 1\} = \Pr\{N_1 > 0\}$. The fraction of time that Server 2 is available to process jobs is:

$$\Pr\{e_2 = 1\} = \frac{\eta}{\xi + \eta} \triangleq \gamma, \tag{5.4}$$

where as the recovery rate $\eta \to \infty$, the availability of Server 2 tends to be 1.

Since the probability that the corresponding server is busy is equal to the utilization [7], we have $\Pr\{N_i > 0\} = \rho_i$, where ρ_i is the utilization of Queue i. We can further formulate the optimization of the PQWS metric for the offloading assignment as:

$$\lambda_i^* = \arg\min_{\lambda_i} \ PQWS, \tag{5.5}$$

where we find the arrival rate λ_i^* to Queue i such that $PQWS$ is minimized when both queues are in operation.

5.2.2 Static Offloading Policy

The static offloading policy is to assign arriving jobs using the optimal assignment scheme which corresponds to the smallest possible cost in the PQWS metric. We always assign the offloading jobs according to that scheme.

Queue 1 refers to offloading jobs from the mobile device to the cloud via a cellular network, which is modelled as an $M/G/1$-FCFS queue. The expected number of jobs in Queue 1 is given by

the Pollaczek-Khinchine formula:

$$\mathbb{E}[N_1] = \lambda_1 \mathbb{E}[S_1] + \frac{\lambda_1^2 \mathbb{E}[S_1^2]}{2(1-\rho_1)}. \tag{5.6}$$

Note, that a job of size X served at speed s will be completed in time X/s. Since $\mathbb{E}[S_i] = \mathbb{E}[X/s_i] = \mathbb{E}[X]/s_i$, the utilization in Queue 1 during the busy period is $\rho_1 = \lambda_1 \mathbb{E}[S_1] = \lambda_1 \mathbb{E}[X]/s_1$. Especially, if the job size X is exponentially distributed, according to $\mathbb{E}[X^2] = 2\mathbb{E}^2[X]$ and $\mathbb{E}[S_i^2] = \mathbb{E}[(X/s_i)^2] = \mathbb{E}[X^2]/s_i^2$, we further have:

$$\begin{aligned} \mathbb{E}[N_1] &= \lambda_1 \mathbb{E}[X]/s_1 + \frac{2\lambda_1^2 \mathbb{E}^2[X]/s_1^2}{2(1-\rho_1)} \\ &= \rho_1 + \frac{2\rho_1^2}{2(1-\rho_1)} = \frac{\rho_1}{1-\rho_1}. \end{aligned} \tag{5.7}$$

Similarly, if the server in Queue 2 is always available like Queue 1, we can have $\Pr\{N_2 > 0 | e_2 = 1\} = \lambda_2 \mathbb{E}[S_2] = \lambda_2 \mathbb{E}[X]/s_2$. However, Queue 2 refers to offloading jobs from the mobile device to the cloud via a WLAN network, which is modelled as an $M/G/1$-FCFS queue with intermittently available service. When a server recovers, it continues to serve the customer whose service has been interrupted, i.e. the work already completed is not lost (cf. data transfers with resume). The mean response time for Queue 2 is then given by [50]:

$$\mathbb{E}[T_2] = \mathbb{E}[Y] + \frac{\lambda_2 \mathbb{E}[Y^2]}{2(1-\rho_2)} + \frac{\kappa(1+\lambda_2/\eta)}{1+\lambda_2 \kappa}, \tag{5.8}$$

where $\kappa = \frac{\xi/\eta}{\lambda_2+\xi+\eta}$, $\mathbb{E}[Y] = \mathbb{E}[S_2]/\gamma$ and $\mathbb{E}[Y^2] = \mathbb{E}[S_2^2]/\gamma^2 + 2\xi/\eta^2 \mathbb{E}[S_2]$. The utilization for Queue 2 during the busy period is $\rho_2 \triangleq \lambda_2 \mathbb{E}[Y] = \lambda_2 \mathbb{E}[S_2]/\gamma$. Since we require $\rho_2 < 1$ for the queue to be stable, it yields: $\lambda_2 \mathbb{E}[S_2] < \gamma$.

According to Little's Law, the expected number of jobs in Queue 2 is $\mathbb{E}[N_2] = \lambda_2 \mathbb{E}[T_2]$. Espe-

cially, if X is exponentially distributed, $\mathbb{E}[X^2] = 2\mathbb{E}^2[X]$, we further have:

$$\begin{aligned}
\mathbb{E}[N_2] &= \lambda_2\mathbb{E}[Y] + \frac{\lambda_2^2\mathbb{E}[Y^2]}{2(1-\rho_2)} + \frac{\kappa(\lambda_2+\lambda_2^2/\eta)}{1+\lambda_2\kappa} \\
&= \rho_2 + \frac{\lambda_2^2\mathbb{E}[S_2^2]}{2\gamma^2(1-\rho_2)} + \frac{\frac{2\xi}{\eta^2}\lambda_2^2\mathbb{E}[S_2]}{2(1-\rho_2)} + \frac{\kappa(\lambda_2+\lambda_2^2/\eta)}{1+\lambda_2\kappa} \\
&= \rho_2 + \frac{2\lambda_2^2\mathbb{E}^2[X]/s_2^2}{2\gamma^2(1-\rho_2)} + \frac{\frac{\xi}{\eta^2}\lambda_2^2\mathbb{E}[X]/s_2}{1-\rho_2} + \frac{\kappa(\lambda_2+\lambda_2^2/\eta)}{1+\lambda_2\kappa} \\
&= \rho_2 + \frac{\rho_2^2}{1-\rho_2} + \frac{\xi\lambda_2^2\mathbb{E}[X]}{\eta^2 s_2(1-\rho_2)} + \frac{\kappa(\lambda_2+\lambda_2^2/\eta)}{1+\lambda_2\kappa} \\
&= \frac{\rho_2}{1-\rho_2} + \frac{\xi\lambda_2^2\mathbb{E}[X]}{\eta^2 s_2(1-\rho_2)} + \frac{\kappa(\lambda_2+\lambda_2^2/\eta)}{1+\lambda_2\kappa}. \qquad (5.9)
\end{aligned}$$

After substituting (5.3), (5.7) and (5.9) into (5.5), we obtain a new offloading policy called tradeoff offloading policy (TOP), and we bring in two previous policies named random offloading policy (ROP) and load-balanced offloading policy (LOP) as a comparison. All the three offloading policies used in our analysis are as follows:

- **ROP**: arriving jobs are randomly assigned to the two queues (Bernoulli split), assuming that each queue has the same probability of being chosen. Therefore, for the queueing model in Fig. 5.2, we have job arrival rate $\lambda_1 = \lambda_2 = \lambda_c/2$. This is a static policy that randomly chooses the transmission channel.
- **LOP**: since the service rate in Queue 2 is much higher than that in Queue 1, more jobs can be served in Queue 2 in the same time. However, since Server 2 is sometimes unavailable, the actual service rate could in fact be lower. When considering the service rate and the availability of servers in different queues, jobs are allocated to Queue i in a Poisson process with parameters $\lambda_1 = \frac{s_1}{s_1+s_2\gamma}\lambda_c$ and $\lambda_2 = \frac{s_2\gamma}{s_1+s_2\gamma}\lambda_c$. Thus, under this policy all the queue loads are equal and given by $\rho_1 = \rho_2 = \frac{\lambda_c\mathbb{E}[X]}{s_1+s_2\gamma}$. This is a static policy that balances the load across the queues.
- **TOP**: arriving jobs are assigned to Queue 1 and Queue 2 according to the optimized objective of the PQWS metric defined in (5.5), minimizing the weighted sum of mean energy consumption and mean response time. This is the proposed static policy that considers the tradeoff between energy consumption and performance according to the ERWS metric.

5.2.3 Dynamic Offloading Policy

The dynamic offloading policy inserts a newly arriving job tentatively into each of the queues and chooses the one corresponding to the smallest increase in the Power-Queue length Weighted Sum (PQWS) metric when also counting the jobs that have been already accepted to the system. We adopt a value function defined in [49] to assign jobs dynamically.

We assume a size-aware system, where the service time of jobs becomes known upon arrival. Let $\Delta_j^{(i)}$ denote the remaining service time of job j ($j = 1, 2, \cdots, n$) in Queue i ($i \in \{1, 2\}$), where jobs are ordered such that job 1 receives service first, then job 2, and so forth. The state of the server is known. Since Server 1 is always available (i.e. $e_1 = 1$), the state of Queue 1 can be denoted by vector $\vec{z}^{(1)}$:

$$\vec{z}^{(1)} = \Big(\Delta_1^{(1)}, \Delta_2^{(1)}, \cdots, \Delta_n^{(1)} |\ e_1 = 1\Big). \tag{5.10}$$

Similarly, the state of Queue 2 is denoted by vector $\vec{z}^{(2)}$:

$$\vec{z}^{(2)} = \Big(\Delta_1^{(2)}, \Delta_2^{(2)}, \cdots, \Delta_n^{(2)} |\ e_2 \in \{0, 1\}\Big), \tag{5.11}$$

where $e_2 \in \{0, 1\}$ indicates if the server 2 is available or not.

The backlog, i.e. the amount of service time for unfinished jobs in Queue i, is denoted by $U_{\vec{z}^{(i)}} = \sum_{j=1}^{n} \Delta_j^{(i)}$, and the cumulative response time during time interval $(0, t)$ is expressed as:

$$V_{\vec{z}^{(i)}}(t) \triangleq \int_0^t N_{\vec{z}^{(i)}}(\tau) d\tau, \tag{5.12}$$

where $N_{\vec{z}^{(i)}}(t)$ is the number of jobs in Queue i at time t. The relative value is the expected difference in the cumulative costs between a system initially in state $\vec{z}$ and a system initially in equilibrium [51]:

$$v_{\vec{z}^{(i)}} \triangleq \lim_{t \to \infty} \mathbb{E}\Big[V_{\vec{z}^{(i)}}(t) - \mathbb{E}[N_{\vec{z}^{(i)}}] \cdot t\Big]. \tag{5.13}$$

According to (5.2), the cumulative cost of the PQWS metric in Queue i is:

$$\omega p_i \Pr\{N_{\vec{z}^{(i)}} > 0, e_i = 1\} + (1 - \omega)\mathbb{E}[N_{\vec{z}^{(i)}}], \tag{5.14}$$

where the first term corresponds to the mean energy consumption and the second corresponds to the mean response time.

The difference $\vec{z}^* - \vec{z}$ that characterizes the expected difference in the future costs between states $\vec{z}^*$ and $\vec{z}$ is the quantity on which job assignment is based. When a new job is admitted to one of

the two queues, it increases the total cost of the PQWS metric. The complete understanding of the future costs is summarized in the relative value $v_{\vec{z}}$. For a fixed policy resulting in a stable system, the relative value $v_{\vec{z}} - v_{\vec{0}}$ gives the expected difference in the infinite horizon of cumulative costs between an arbitrary state $\vec{z}$, and an empty system initially in state $\vec{0}$ with no jobs.

Theorem 3. *For Queue i, the difference of relative values with respect to the weighted sum of the energy consumption and response time in (5.14) can be calculated by:*

$$v_{\vec{z}^{(i)}} - v_{\vec{0}^{(i)}} = \omega\Big(v^E_{\vec{z}^{(i)}} - v^E_{\vec{0}^{(i)}}\Big) + (1-\omega)\cdot\Big(v^T_{\vec{z}^{(i)}} - v^T_{\vec{0}^{(i)}}\Big). \tag{5.15}$$

Proof. The difference of relative values with respect to the weighted sum of the energy consumption and response time can be decomposed into:

$$\begin{aligned} v_{\vec{z}^{(i)}} - v_{\vec{0}^{(i)}} &= \Big[\omega v^E_{\vec{z}^{(i)}} + (1-\omega)v^T_{\vec{z}^{(i)}}\Big] - \Big[\omega v^E_{\vec{0}^{(i)}} + (1-\omega)v^T_{\vec{0}^{(i)}}\Big] \\ &= \omega\left(v^E_{\vec{z}^{(i)}} - v^E_{\vec{0}^{(i)}}\right) + (1-\omega)\cdot\left(v^T_{\vec{z}^{(i)}} - v^T_{\vec{0}^{(i)}}\right), \end{aligned}$$

from which the result follows directly. □

It can be seen from (5.15) that the difference of relative values with respect to the PQWS metric in state $\vec{z}$ can be decomposed into the difference of relative values with respect to energy consumption and the difference of relative values with respect to response time, which can be treated separately as follows.

Theorem 4. *For Queue i with a work conserving queuing discipline, the difference of relative energy costs can be calculated as [95]:*

$$v^E_{\vec{z}^{(i)}} - v^E_{\vec{0}^{(i)}} = \Pr\{e_i = 1\}\cdot p_i U_{\vec{z}^{(i)}}. \tag{5.16}$$

Proof. We assume identical arrivals to a single-server system initially in state $\vec{z}$ and to an empty single-server reference system. When all the initial work $U_{\vec{z}^{(i)}}$ in the queue i is served, both systems are empty. Thus, the difference of energy consumption between the two systems is $p_i U_{\vec{z}^{(i)}}$, which is simply the difference in the energy needed to serve the initial work. Further, since energy is consumed only when the server is available, we derive (5.16). □

According to **Proposition 2**, since Server 1 is always available, the difference of relative energy consumption is given by:

$$v^E_{\vec{z}^{(1)}} - v^E_{\vec{0}^{(1)}} = \Pr\{e_1 = 1\}\cdot p_1 U_{\vec{z}^{(1)}} = p_1 U_{\vec{z}^{(1)}}. \tag{5.17}$$

Similarly, since Server 2 is intermittently available, the difference of relative energy consumption is given by:

$$v^E_{\vec{z}^{(2)}} - v^E_{\vec{0}^{(2)}} = \Pr\{e_2 = 1\} \cdot p_2 U_{\vec{z}^{(2)}} = \gamma \cdot p_2 U_{\vec{z}^{(2)}}. \tag{5.18}$$

Theorem 5. *For the stable $M/G/1$-FCFS Queue 1, the difference of relative response times is given by [49]:*

$$v^T_{\vec{z}^{(1)}} - v^T_{\vec{0}^{(1)}} = \frac{\lambda_1 U^2_{\vec{z}^{(1)}}}{2(1-\rho_1)} + \sum_{j=1}^{n}(n+1-j)\Delta^{(1)}_j. \tag{5.19}$$

And for the stable $M/G/1$-FCFS Queue 2 with intermittently available service, the difference of relative response times is given by [50]:

$$v^T_{\vec{z}^{(2)}} - v^T_{\vec{0}^{(2)}} = \frac{\lambda_2 U^2_{\vec{z}^{(2)}}}{2\gamma^2(1-\rho_2)} + \frac{1}{\gamma}\sum_{j=1}^{n}(n+1-j)\Delta^{(2)}_j + \frac{1-e_2}{\eta}\left[n + \frac{\lambda_2 U_{\vec{z}^{(2)}}}{\gamma(1-\rho_2)}\right]. \tag{5.20}$$

From (5.20), we note that the terms with $(1-e_2)$ as factor correspond to an additional penalty due to the currently unavailable Server 2. According to **Proposition 1**, after substituting (5.17)-(5.20) into (5.15), we can get the differences of relative values with respect to the PQWS metric for Queue 1 and Queue 2 as follows, respectively:

$$v_{\vec{z}^{(1)}} - v_{\vec{0}^{(1)}} = \omega p_1 U_{\vec{z}^{(1)}} + (1-\omega)\left[\frac{\lambda_1 U^2_{\vec{z}^{(1)}}}{2(1-\rho_1)} + \sum_{j=1}^{n}(n+1-j)\Delta^{(1)}_j\right], \tag{5.21}$$

$$\begin{aligned} v_{\vec{z}^{(2)}} - v_{\vec{0}^{(2)}} = {} & \omega\gamma p_2 U_{\vec{z}^{(2)}} + (1-\omega)\Bigg\{\frac{\lambda_2 U_{\vec{z}^{(2)}}}{2\gamma^2(1-\rho_2)} + \frac{1}{\gamma}\sum_{j=1}^{n}(n+1-j)\Delta^{(2)}_j \\ & + \frac{1-e_2}{\eta}\left[n + \frac{\lambda_2 U_{\vec{z}^{(2)}}}{\gamma(1-\rho_2)}\right]\Bigg\}. \end{aligned} \tag{5.22}$$

We assume that n jobs have already arrived to the queuing system, for a newly arriving job of size x^* with index $(n+1)^{\text{th}}$, the service time of the new job $\Delta^{(i)}_{n+1} = x^*/s_i$ is inserted according to the queuing discipline used in Queue i. Therefore, the mean additional increase in future costs to assign the new job is $v_{\vec{z}^{(i)}_{\text{new}}} - v_{\vec{z}^{(i)}}$, where $\vec{z}^{(i)}_{\text{new}} = \vec{z}^{(i)} \oplus \Delta^{(i)}_{n+1}$ is the new state if the arriving job with service time $\Delta^{(i)}_{n+1}$ is added to Queue i.

Theorem 6. *The relative increment of the PQWS metric with the admission cost of a job with service*

time $\Delta_{n+1}^{(i)}$ in Queue i is as follows:

$$v_{\vec{z}_{new}^{(1)}} - v_{\vec{z}^{(1)}} = \omega p_1 \Delta_{n+1}^{(1)} + (1-\omega)\left[\frac{\lambda_1 \Delta_{n+1}^{(1)}\left(2U_{\vec{z}^{(1)}} + \Delta_{n+1}^{(1)}\right)}{2(1-\rho_1)} + U_{\vec{z}^{(1)}} + \Delta_{n+1}^{(1)}\right], \quad (5.23)$$

$$v_{\vec{z}_{new}^{(2)}} - v_{\vec{z}^{(2)}} = \omega\gamma p_2 \Delta_{n+1}^{(2)} + (1-\omega)\left\{\frac{\lambda_2 \Delta_{n+1}^{(2)}\left(2U_{\vec{z}^{(2)}} + \Delta_{n+1}^{(2)}\right)}{2\gamma^2(1-\rho_2)} + \frac{1}{\gamma}\left(U_{\vec{z}^{(2)}} + \Delta_{n+1}^{(2)}\right) + \frac{1-e_2}{\eta}\left[1 + \frac{\lambda_2 \Delta_{n+1}^{(2)}}{\gamma(1-\rho_2)}\right]\right\}. \quad (5.24)$$

Proof. According to (5.21) and (5.22), we have the relative values of the PQWS metric in the new state $\vec{z}_{\text{new}}^{(i)}$:

$$v_{\vec{z}_{\text{new}}^{(1)}} - v_{\vec{0}^{(1)}} = \omega p_1\left(U_{\vec{z}^{(1)}} + \Delta_{n+1}^{(1)}\right) + (1-\omega)\left[\frac{\lambda_1\left(U_{\vec{z}^{(1)}} + \Delta_{n+1}^{(1)}\right)^2}{2(1-\rho_1)} + \sum_{j=1}^{n+1}(n+2-j)\Delta_j^{(1)}\right], \quad (5.25)$$

$$v_{\vec{z}_{\text{new}}^{(2)}} - v_{\vec{0}^{(2)}} = \omega\gamma p_2\left(U_{\vec{z}^{(2)}} + \Delta_{n+1}^{(2)}\right) + (1-\omega)\left\{\frac{\lambda_2\left(U_{\vec{z}^{(2)}} + \Delta_{n+1}^{(2)}\right)^2}{2\gamma^2(1-\rho_2)} + \frac{1}{\gamma}\sum_{j=1}^{n+1}(n+2-j)\Delta_j^{(2)} + \frac{1-e_2}{\eta}\left[n+1+\frac{\lambda_2\left(U_{\vec{z}^{(2)}} + \Delta_{n+1}^{(2)}\right)}{\gamma(1-\rho_2)}\right]\right\}. \quad (5.26)$$

The relative increment of the PQWS metric can be decomposed into the difference of relative values:

$$v_{\vec{z}_{\text{new}}^{(i)}} - v_{\vec{z}^{(i)}} = \left(v_{\vec{z}_{\text{new}}^{(i)}} - v_{\vec{0}^{(i)}}\right) - \left(v_{\vec{z}^{(i)}} - v_{\vec{0}^{(i)}}\right). \quad (5.27)$$

After substituting (5.21) and (5.26) into (5.27), we can derive (5.23), and then after substituting (5.22) and (5.26) into (5.27), we further derive (5.24). □

According to (5.23) and (5.24), we obtain the dynamic offloading assignment policy that assigns the newly arriving job to Queue i^* according to:

$$i^* = \arg\min_i \left\{v_{\vec{z}_{\text{new}}^{(i)}} - v_{\vec{z}^{(i)}}\right\}. \quad (5.28)$$

5.2.4 Numerical Examples

Figure 5.3 describes an $M/M/1$ queuing system with an intermittently available server by using MATLAB Simulink tool. A server is either down (failed) or up (repaired). A Stateflow chart is used to implement the WiFi network's ON and OFF states. The state of the server is an output signal from the Stateflow block and is used to invoke the Enabled Gate block that precedes a server in a queuing system.

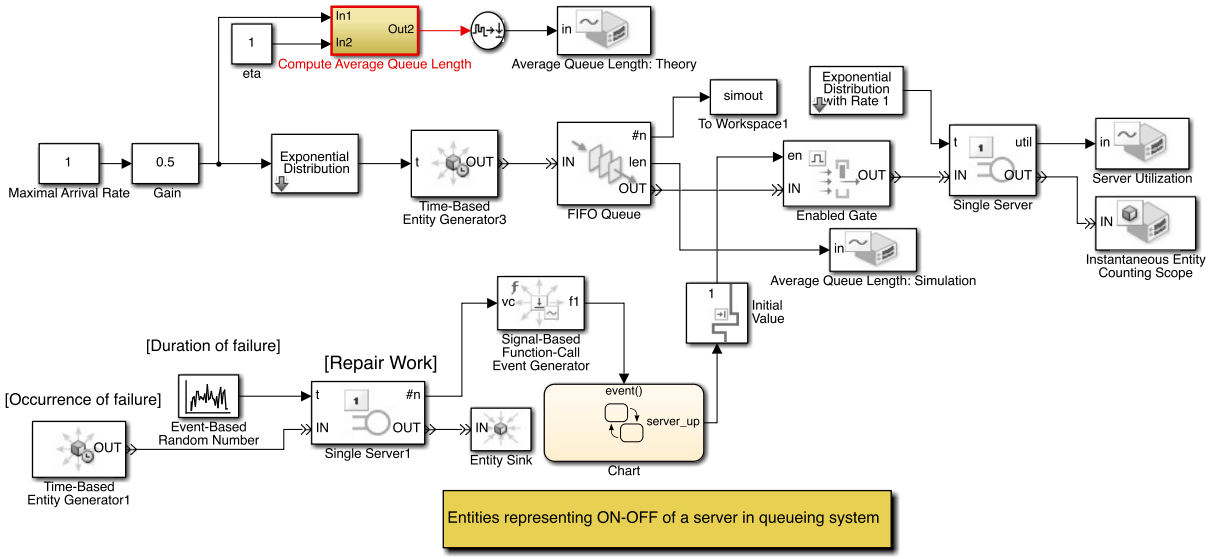
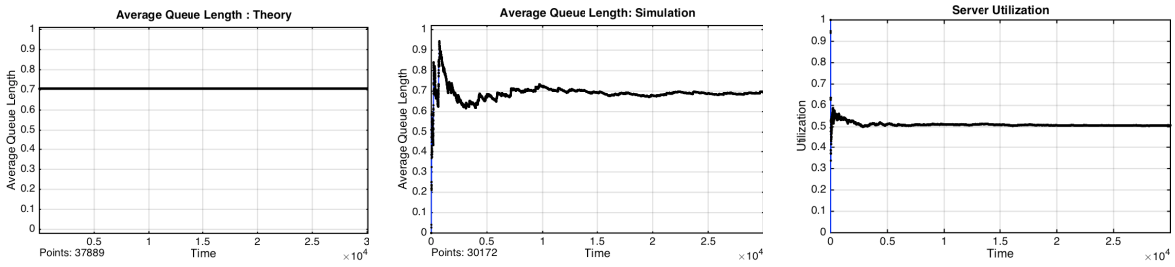


Figure 5.3: $M/M/1$ queuing system with an intermittently available server

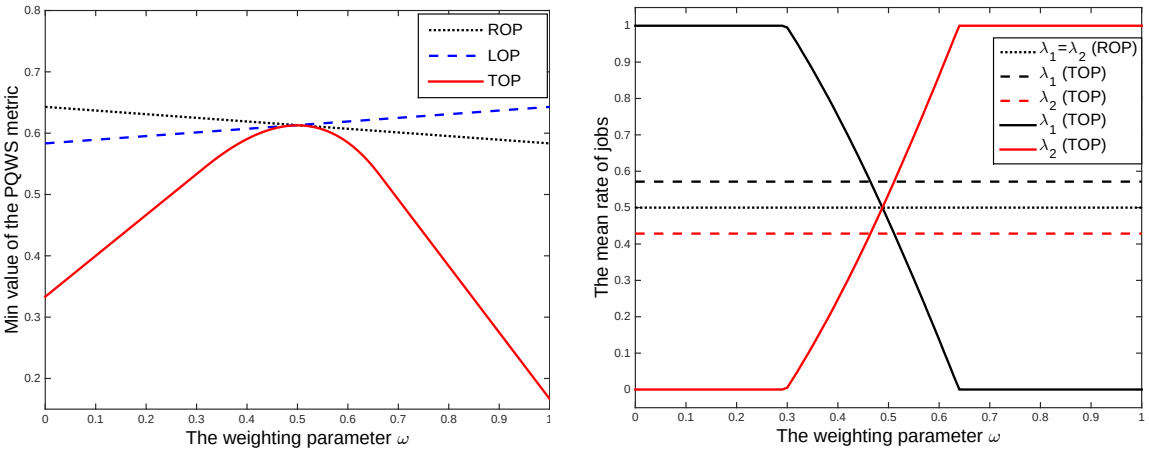
We set the failure rate $\xi = 1/8$ and recovery rate $\eta = 1$, $\lambda_2 = 0.5$ and $\mu_2 = 1$. The simulation result is as shown in Fig. 5.4. It can be seen that the average queue length in simulation is the same as the one in (5.9). The server utilization equals $\rho_2 = \lambda_2/\mu_2 = 0.5$. Therefore, we can observe a good match between theory and simulation.



(a) Theory (b) Simulation (c) Utilization

Figure 5.4: Simulation of $M/M/1$ queuing system with an intermittently available server

We evaluate the assignment policies defined in the previous sections using the model with two queues to represent the offloading scheme. We set the parameters for Server 1 to $s_1 = 400$ Kbps, $p_1 = 4$ W and for Server 2 to $s_2 = 600$ Kbps, $p_2 = 2$ W. Besides, we suppose that the total job arrival rate for offloading is $\lambda_c = 1$ packet/s, both the failure rate ξ and recovery rate η of Server 2 are equal to 1, and the packet sizes are exponentially distributed with $X \sim \text{Exp}(1/100)$.



(a) Minimum value of the PQWS metric

(b) Corresponding assignment results

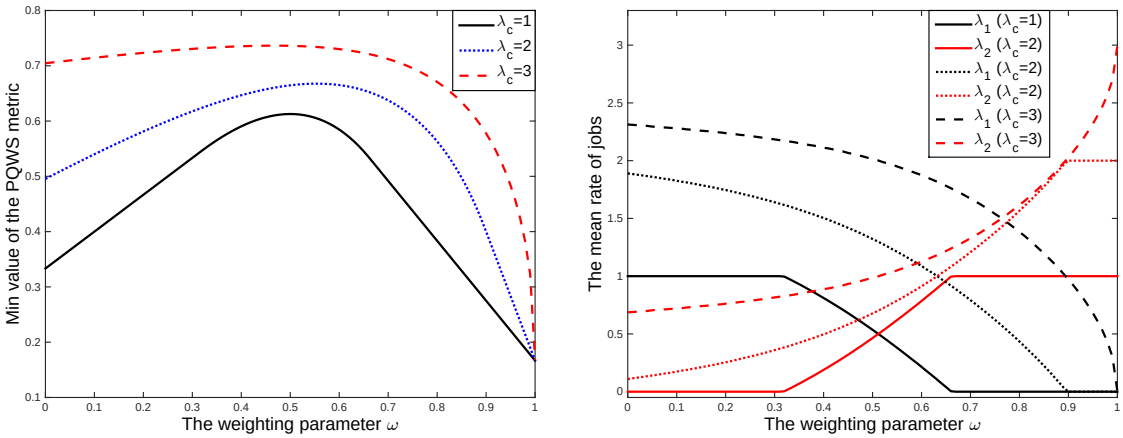
Figure 5.5: Comparison of different static offloading policies

One can see in Fig. 5.5(a) that the proposed tradeoff offloading policy (TOP) performs significantly better than the random offloading policy (ROP) and load-balanced offloading policy (LOP). Interestingly, all three policies achieve the same minimum value of the PQWS metric when the weighting parameter $\omega = 0.5$, where the mean energy consumption and mean response time are equally important. At low values of ω, the response time is always more important than the energy consumption and it dominates the queue selection. At high values, the energy consumption component becomes the decisive factor. The smallest PQWS metric is obtained when $\omega = 1$ and the TOP assignment is used. This shows that TOP outperforms other policies most in the case when only the energy consumption is considered in the offloading system. The corresponding assignments are in Fig. 5.5(b). For the ROP and LOP schemes, the assignment is insensitive to ω. As expected, the random policy assigns half of the jobs to either queue. However, for the TOP scheme, at low values of ω, all the jobs are directed to Queue 1, and as ω increases, slowly more and more jobs are assigned to Queue 2 to balance the tradeoff between energy consumption and performance.

The impact of the total job arrival rate λ_c is shown in Fig. 5.6(a), where we only use the static offloading policy TOP. The effect mentioned above, that optimizing one element of the combined

metric holds for different values of the arrival rate, but is much more pronounced at high load. In fact, for high load focussing on energy consumption ($\omega = 1$) gives much lower PQWS than only regarding the job response time ($\omega = 0$). In Fig. 5.6(b) for higher load the point where jobs are assigned to both queues in equal shares is when $\omega > 0.5$, i.e. when energy cost weighs more than response time in the metric.

In Fig. 5.7 we study the recovery rate of Server 2. The faster recovery is, the lower becomes the PQWS metric under TOP policy. This is because as $\eta \to \infty$, Server 2 is always available and then the preferred scheduling target.



(a) Minimum value of the PQWS metric

(b) Corresponding assignment results

Figure 5.6: The proposed TOP under different total job arrival rates

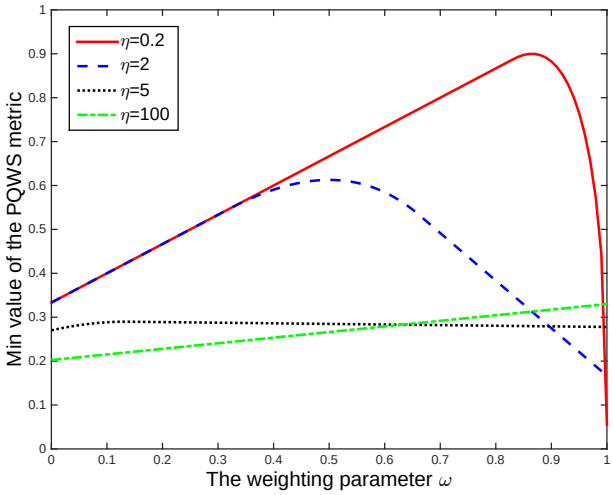
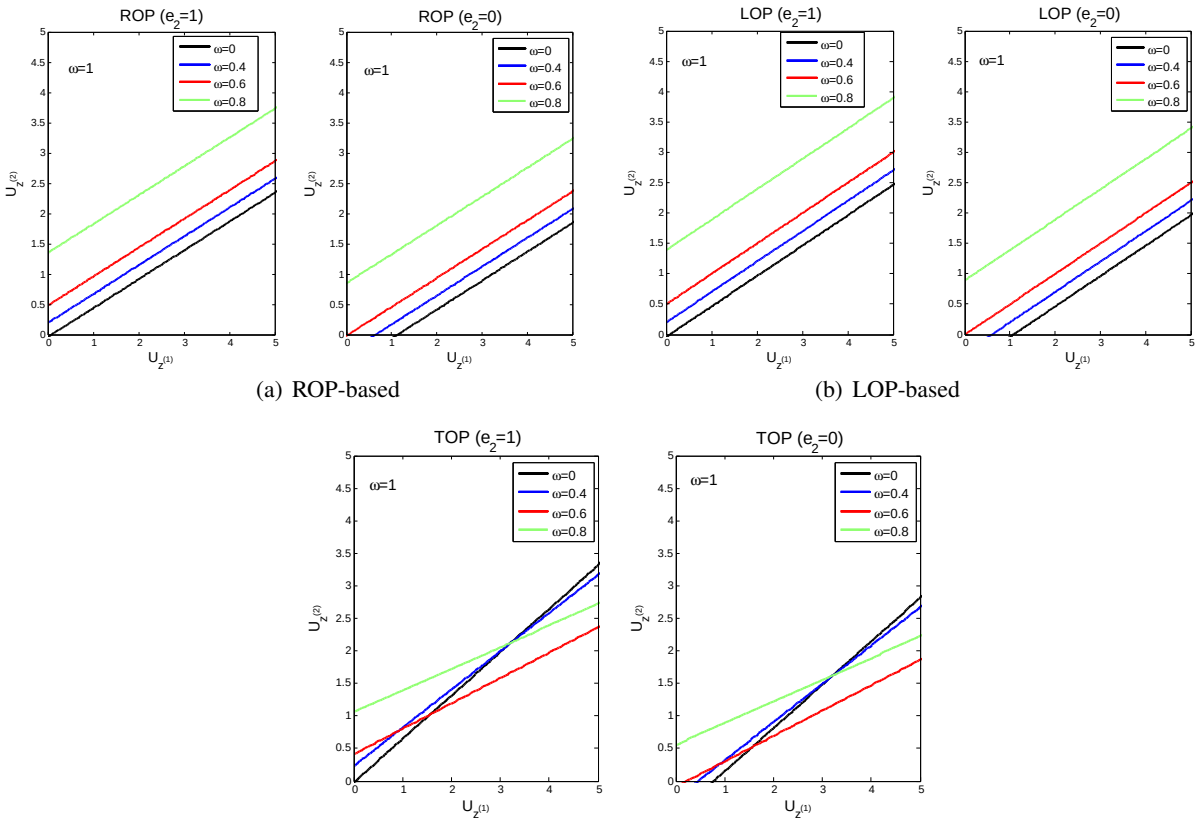


Figure 5.7: The proposed TOP under different recovery rates



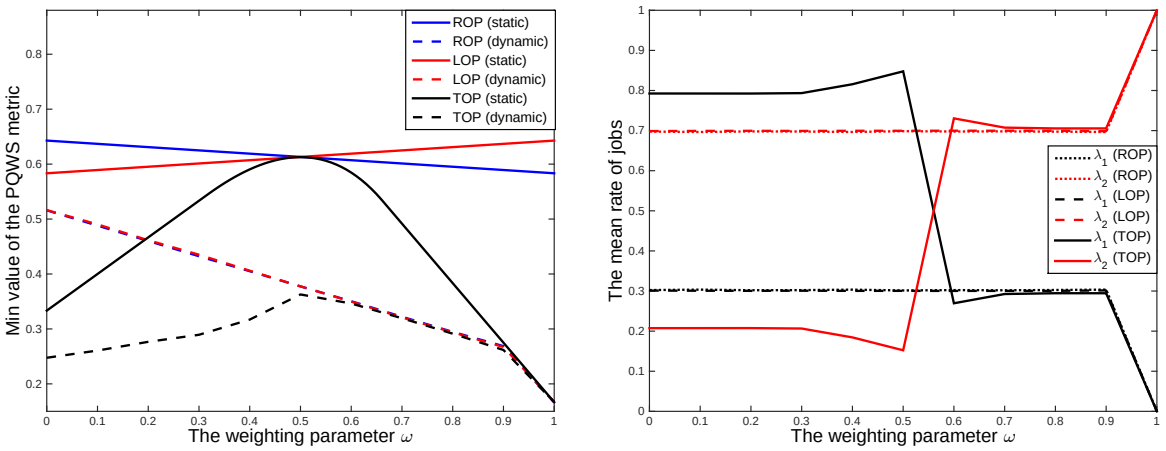
(a) ROP-based (b) LOP-based

(c) TOP-based

Figure 5.8: Dynamic decisions to choose Queue i under different static offloading policies

To illustrate the dynamic offloading policy, we assume that the length of a newly arriving job is $x^* = 100$ Kb. In Fig. 5.8 the axes represent the existing work $U_{\vec{z}(i)}$ in Queue i, the colored lines denote the threshold under different ω where the arriving job has equal probability to join both queues. The areas above the thresholds refer to choosing Queue 1, while the areas below the threshold refer to choosing Queue 2 for the new job. We dynamically assign the job into one of the two queues based on the three static policies. The area that assigns new jobs to Queue 2 when Server 2 is available ($e_2 = 1$) is much larger compared to the area when it is unavailable ($e_2 = 0$). For the ROP and LOP policies, the thresholds are always parallel under different values of $\omega \in [0, 1]$. Whereas assigning the job to a low power server (Queue 1) would be beneficial from the energy consumption perspective at low loads, such a policy may end up in difficulties at higher loads as the response time grows quickly. However, for the TOP scheme, the assignment thresholds cross under different ω. That is because the TOP scheme tries to dynamically balance the allocated jobs in order

to minimize the objective value of the PQWS metric.



(a) Minimum value of the PQWS metric

(b) Corresponding dynamic assignment results

Figure 5.9: Comparison of different static and dynamic offloading policies

Figure 5.9(a) compares the dynamic offloading schemes and the different static policies. The dynamic offloading scheme under the TOP policy achieves the best performance, while the ROP- and LOP-based schemes have almost the same PQWS value when jobs are dynamically assigned to the two queues. Further, the TOP-based dynamic offloading scheme yields a gain over the schemes where only the static policies are adopted, e.g. they are up to 70% better than the static policy of ROP. This is because the dynamic offloading policy considers effectively the dynamic increase in each queue that newly arriving jobs bring in. For the TOP-based dynamic offloading scheme in Fig. 5.9(b), there exists a turning point in the middle where allocating the jobs to Queue 2 begins to outweigh Queue 1. There is a leap when $\omega = 1$, since $U_{\vec{z}^{(2)}} < U_{\vec{z}^{(1)}}$ at that point, the new arrival jobs are always assigned to Queue 2.

5.3 Offloading Assignment Models

Our objective is to minimize the mean energy consumption and the mean response time. Since only one option exists for the jobs that cannot be offloaded (as they will always be processed locally) our attention solely falls on the offloadable jobs, where the right decision must be taken whether or not to offload or how to offload.

5.3.1 Problem Formulation

As indicated in Fig. 5.1, the key elements for the considered offloading system are as shown inside the blue block (local execution) and the red block (remote execution), which are analyzed separately as follows.

Local Execution

As shown inside the blue dotted block in Fig. 5.1, there are two kinds of jobs (offloadable and unoffloadable) arriving at the processor of the mobile device. We adopt the preemptive scheduling policy here. That is, from the perspective of an offloadable job, the unoffloadable jobs do not exist, since service to the unoffloadable jobs is immediately interrupted upon the arrival of an offloadable job. To the offloadable jobs, the system behaves exactly like an $M/G/1$ queue with arrival rate λ_m and service rate $\mu_m = \mathbb{E}[S_m]^{-1}$.

The utilization, i.e. the fraction of time when the server is busy, is denoted as: $\rho_m = \lambda_m/\mu_m$. The expected number of jobs in an ordinary $M/G/1$-FCFS queue is given by:

$$\mathbb{E}[N_m] = \rho_m + \frac{1 + C_{S_m}^2}{2} \cdot \frac{\rho_m^2}{1 - \rho_m}, \tag{5.29}$$

where $C_{S_m}^2 = \mu_m \sigma_{S_m}^2$ denotes the squared coefficient of variation for the service time distribution. Especially if the service is exponentially distributed we have $C_{S_m}^2 = 1$. Further, by Little's Law we obtain:

$$\mathbb{E}[T_m] = \frac{1}{\lambda_m}\mathbb{E}[N_m] = \frac{1}{\lambda_m} \cdot \frac{\rho_m}{1 - \rho_m}. \tag{5.30}$$

We assume that the mobile device consumes energy only when there are jobs in the system and that the mobile device operates at a constant power p_m whenever it is busy [116]. Since $P_m = \lambda_m \mathcal{E}_m$ is the consumed power, the mean energy consumption $\mathbb{E}[\mathcal{E}_m]$ can be more conveniently expressed as:

$$\mathbb{E}[\mathcal{E}_m] = \frac{1}{\lambda_m} \cdot \mathbb{E}[P_m] = \frac{1}{\lambda_m} \cdot p_m \Pr\{N_m > 0\} = \frac{1}{\lambda_m} \cdot p_m \rho_m. \tag{5.31}$$

Remote Execution

As shown inside the red dotted block in Fig. 5.1, the offloading process includes the transmission model and the cloud model. Transmission rates in real wireless networks are not stable and are

affected by signal quality and the presence of other users. According to network profiling in Section 2.2.1, we assume that a cellular network is available to mobile users all the time while the availability of a WiFi network depends on the location. That is, mobile users move in and out of a WiFi coverage area. To facilitate the analysis of the mobile cloud offloading system, we model this time variation of the WiFi connection state by the ON-OFF alternating renewal process $\left(T_{\text{ON}}^{(j)}, T_{\text{OFF}}^{(j)}\right)$, $j \geq 1$, as shown in Fig. 5.10.

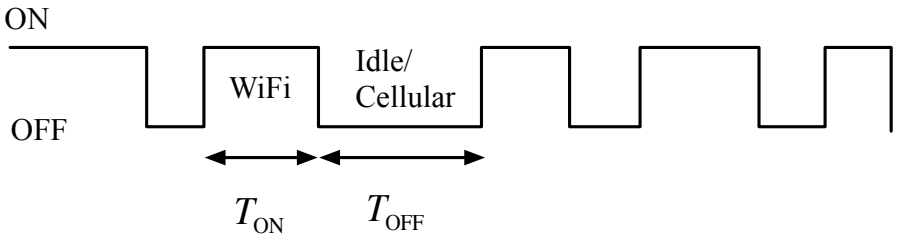


Figure 5.10: The WiFi network availability model [79]

The ON periods represent the presence of the WiFi connectivity, while the OFF periods represent the interruption of the WiFi connectivity. During the latter periods data is either not transmitted (the interface is idle) or it is transmitted only through the cellular network. The duration of each ON period $T_{\text{ON}}^{(j)}$ or OFF period $T_{\text{OFF}}^{(j)}$, is assumed to be an exponentially distributed random variable and independent of the duration of other ON or OFF periods [77]. The WiFi availability ratio (AR) can be defined as:

$$AR = \frac{\mathbb{E}[T_{\text{ON}}]}{\mathbb{E}[T_{\text{ON}}] + \mathbb{E}[T_{\text{OFF}}]}. \tag{5.32}$$

Accordingly, we build two different offloading strategies:

- **Uninterrupted Offloading Strategy**: we employ a single queue with two states to offload jobs to the cloud server. When there is a WiFi connection available, all jobs are sent over the WiFi network; otherwise, they are sent over the cellular interface [64]. Therefore, the process of offloading jobs to the cloud is not to be interrupted.
- **Interrupted Offloading Strategy**: we assign jobs upon arrival to one of two parallel queues which describe cellular or WiFi transmission [131]. Offloading is interrupted during the periods when WiFi is disconnected. It is a channel assignment problem where incoming jobs are splitted on arrival for service by two servers and join before departure. Only when all the jobs are transmitted and have rejoined can the cloud processing start.

We have two states (for the uninterrupted offloading strategy) or two parallel queues (for the interrupted offloading strategy). However, for a generalized offloading model, there may be N queues (states). Upon arrival of a job an assignment decision is made and the job is placed into

the corresponding queue (state). We assume each server (state) operates at a constant power p_i whenever it is busy. We use $e_i \in \{0, 1\}$ to indicate whether Server (State) i is available or not. If $e_i = 1$, Server (State) i is available, otherwise it is unavailable. Since $P_i = \lambda_i \mathcal{E}_i$ is the consumed power, further by Little's Law, $\mathbb{E}[N_i] = \lambda_i \mathbb{E}[T_i]$, the mean energy consumption and mean response time due to offloading can be calculated as follows, respectively.

$$\begin{aligned}
\mathbb{E}[\mathcal{E}_o] &= \frac{1}{\lambda_c} \sum_{i=1}^{N} \lambda_i \mathbb{E}[\mathcal{E}_i] = \frac{1}{\lambda_c} \sum_{i=1}^{N} \mathbb{E}[P_i] \\
&= \frac{1}{\lambda_c} \left\{ \sum_{i=1}^{N} p_i \Pr\{N_i > 0, e_i = 1\} \right\}, \qquad (5.33) \\
\mathbb{E}[T_o] &= \frac{1}{\lambda_c} \sum_{i=1}^{N} \lambda_i \mathbb{E}[T_i] + \mathbb{E}[T_c] \\
&= \frac{1}{\lambda_c} \left\{ \sum_{i=1}^{N} \mathbb{E}[N_i] + \mathbb{E}[N_c] \right\}, \qquad (5.34)
\end{aligned}$$

where λ_i is the mean rate of jobs arriving to Queue (State) i, $\mathbb{E}[\mathcal{E}_i]$ and $\mathbb{E}[T_i]$ are the mean energy consumption and mean response time in Queue (State) i, respectively, and $\mathbb{E}[T_c]$ is the expected execution time on the cloud server.

Since the utilization equals the probability that the corresponding server (state) is busy, we have $\Pr\{N_i > 0 | e_i = 1\} = \rho_i$. The number of jobs in Queue (State) i can be calculated as:

$$\Pr\{N_i > 0, e_i = 1\} = \Pr\{N_i > 0 | e_i = 1\} \cdot \Pr\{e_i = 1\} = \rho_i \cdot \Pr\{e_i = 1\}. \qquad (5.35)$$

The ERWP Metric

From the above analysis in the local and remote executions, we can derive the ERWP metric from (2.4) after combining the results in (5.30)-(5.34), which is expressed by:

$$\begin{aligned}
ERWP &= \left\{ \pi \mathbb{E}[\mathcal{E}_o] + (1 - \pi) \mathbb{E}[\mathcal{E}_m] \right\}^{\omega} \cdot \left\{ \pi \mathbb{E}[T_o] + (1 - \pi) \mathbb{E}[T_m] \right\}^{1-\omega} \\
&= \frac{1}{\lambda} \left\{ \sum_{i=1}^{N} \mathbb{E}[P_i] + \mathbb{E}[P_m] \right\}^{\omega} \cdot \left\{ \sum_{i=1}^{N} \mathbb{E}[N_i] + \mathbb{E}[N_c] + \mathbb{E}[N_m] \right\}^{1-\omega}. \qquad (5.36)
\end{aligned}$$

Similarly, we can also derive the ERWS metric from (2.1) and the ERP metric from (2.3) for the offloadable jobs.

5.3.2 Uninterrupted Offloading Strategy

Figure 5.11 depicts an uninterrupted offloading strategy based on the WiFi network's availability model. The total cost for offloading a job is composed of the cost for sending the job to the cloud and idly waiting for the cloud to complete the job. We propose a Markov modulated queue for uplink transmission. A single-server queuing system that oscillates between two feasible states is denoted by f_{ON} and f_{OFF}. The persistence of the system at any state is governed by a random mechanism: if the system functions at state f_{ON} it transits to the other state at rate ξ and if the system functions at state f_{OFF} it transits to the other state at rate η [11].

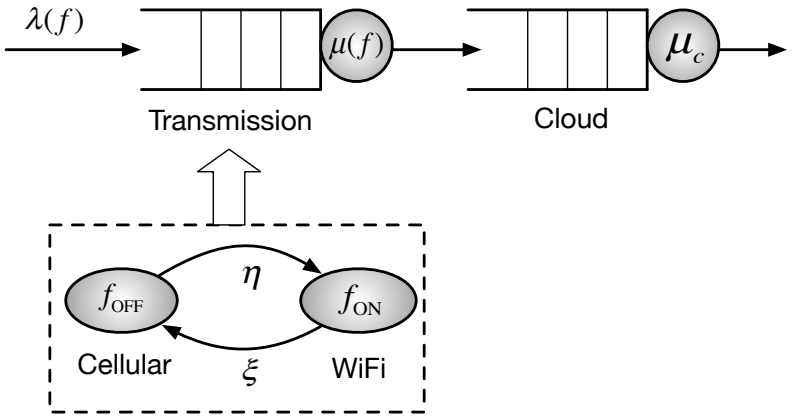


Figure 5.11: The uninterrupted offloading strategy

From Fig. 5.11, the jobs are offloaded either via a cellular connection or a WiFi network to the cloud. We assume that the mean job size is $\mathbb{E}[X]$, the transmission speed of the cellular network is s_1 with service rate $\mu_1 = s_1/\mathbb{E}[X]$, and its operating power is p_1 when serving jobs and zero whenever idle. Similarly, WiFi runs at speed s_2, with service rate $\mu_2 = s_2/\mathbb{E}[X]$, and its operating power is p_2. We assume that jobs arrive in a Poisson process with rate $\lambda(f)$, and the modulating process $f \in \{f_{\text{ON}}, f_{\text{OFF}}\}$ determines the arrival rates:

$$\lambda(f) = \begin{cases} \lambda_1, & \text{if } f = f_{\text{OFF}} \\ \lambda_2, & \text{if } f = f_{\text{ON}} \end{cases} \quad \text{and} \quad \mu(f) = \begin{cases} \mu_1, & \text{if } f = f_{\text{OFF}} \\ \mu_2, & \text{if } f = f_{\text{ON}} \end{cases}. \tag{5.37}$$

The original offloading jobs arrive at the system according to a Poisson process with rate λ_c, since the modulating process has two states, which results in independent Poisson processes with rate λ_i ($i \in \{1, 2\}$), we have $\lambda_1 + \lambda_2 = \lambda_c$. It is known from the literature [34] that a product-form for the tandem queues between the modulated (the exponential queue) and the modulating processes exists

if and only if the following condition holds:

$$\exists \rho \in R^{+}, \quad \text{s.t. } \forall f \in \{f_{\text{ON}}, f_{\text{OFF}}\}, \quad \frac{\lambda(f)}{\mu(f)} = \rho. \tag{5.38}$$

Then by substituting (5.37) into (5.38), we have: $\lambda_1/\mu_1 = \lambda_2/\mu_2$, which is the case where the traffic intensities ρ_1 and ρ_2 are equal, though arrival intensities and service capacities need not be equal. Now this transition from one state to the second state will carry no influence on the random variable 'number of jobs present in the queue', since the traffic intensity $\lambda_i/\mu_i(=\rho)$ has not changed [142].

Figure 5.11 demonstrates that the uninterrupted offloading strategy is a conditional product-form model consisting of a tandem between a transmission queue (with two alternating states of cellular and WiFi) and an exponential queue representing the cloud. The former queue alternates its service by means of mutual resets according to the availability of WiFi, which is governed by an IPP with exponentially distributed ON-OFF periods. We model the intermittent availability of WiFi hotspots as a FCFS queue with occasional server break-down [50]. The queue works in mutual exclusive states and the reset occurs when the WiFi period alternates, either from ON-state to OFF-state or from OFF-state to ON-state. When WiFi becomes unavailable, the cellular network starts transmitting, otherwise when the WiFi connection becomes available, jobs are served only by the WiFi network. Specifically, offloading is not interrupted in this strategy, either in ON-state where the WiFi network is processing the existing jobs, or in the OFF-state during which the job is serving by the cellular network

Since there is no waiting time before entering service, the $M/G/\infty$ queue of the cloud is occasionally referred to as a delay (sometimes pure delay) station, the probability distribution of the delay being that of the service time. Thus, the expected execution time taken on the cloud server can be calculated as $\mathbb{E}[T_c] = 1/\mu_c$. Since the application jobs are remotely executed on the cloud server rather than on the mobile device and we are only concerned with the energy consumption of the mobile device, we do not need to calculate the energy consumption of the cloud server.

Metric-Based Analysis

We use queueing analysis to derive formulas for the average number of jobs for the $M/M/1$ queue. Given the previously stated assumptions, the uninterrupted offloading strategy can be modelled with a 2D Markov chain, as shown in Fig. 5.12. The states with cellular network are denoted $\{1, n\}$, and the states with WiFi connectivity are denoted $\{2, n\}$. The parameter n corresponds to the number of

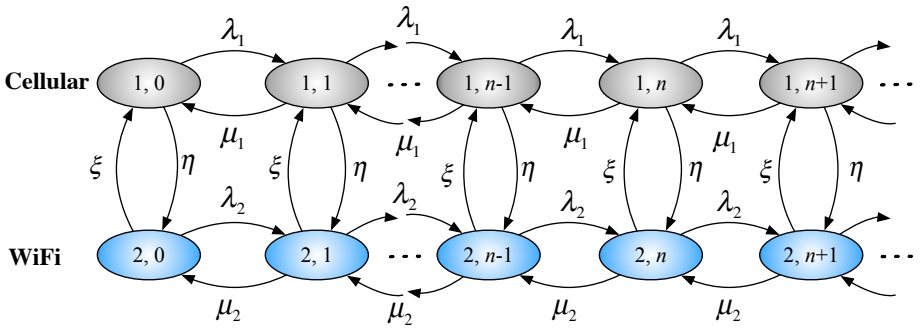


Figure 5.12: The Markov chain for the uninterrupted offloading strategy

jobs in the system (queuing and in service). Writing the balance equations for this chain gives:

$$\pi_{1,0}(\lambda_1+\eta) = \pi_{1,1}\mu_1+\pi_{2,0}\xi, \tag{5.39a}$$

$$\pi_{2,0}(\lambda_2+\xi) = \pi_{2,1}\mu_2+\pi_{1,0}\eta, \tag{5.39b}$$

$$\pi_{1,n}(\lambda_1+\eta+\mu_1) = \pi_{1,n-1}\lambda_1+\pi_{1,n+1}\mu_1+\pi_{2,n}\xi \qquad (n>0), \tag{5.39c}$$

$$\pi_{2,n}(\lambda_2+\xi+\mu_2) = \pi_{2,n-1}\lambda_2+\pi_{2,n+1}\mu_2+\pi_{1,n}\eta \qquad (n>0). \tag{5.39d}$$

The steady-state probabilities for the periods with only cellular access and with WiFi availability are separately calculated as:

$$\pi_1 = \frac{\mathbb{E}[T_{\text{OFF}}]}{\mathbb{E}[T_{\text{ON}}]+\mathbb{E}[T_{\text{OFF}}]} = \frac{\xi}{\eta+\xi}$$

$$\pi_2 = \frac{\mathbb{E}[T_{\text{ON}}]}{\mathbb{E}[T_{\text{ON}}]+\mathbb{E}[T_{\text{OFF}}]} = \frac{\eta}{\eta+\xi}.$$

Let two quantities λ^* and μ^* be defined as:

$$\lambda^* = \pi_1\cdot\lambda_1+\pi_2\cdot\lambda_2, \tag{5.40}$$

$$\mu^* = \pi_1\cdot\mu_1+\pi_2\cdot\mu_2\,. \tag{5.41}$$

The probability generating functions for both cellular and WiFi states are defined as:

$$G_i(z)=\sum_{n=0}^{\infty}\pi_{i,n}z^n, |z|\le 1, \qquad \forall i=1,2. \tag{5.42}$$

After some calculations, we obtain:

$$(\lambda_1+\eta+\mu_1)G_1(z) = \lambda_1 z G_1(z)+\xi G_2(z)+\frac{\mu_1}{z}[G_1(z)-\pi_{1,0}]+\pi_{1,0}\mu_1, \quad (5.43)$$

$$(\lambda_2+\xi+\mu_2)G_2(z) = \lambda_2 z G_2(z)+\eta G_1(z)+\frac{\mu_2}{z}[G_2(z)-\pi_{2,0}]+\pi_{2,0}\mu_2. \quad (5.44)$$

After solving the equations of (5.43) and (5.44), we have [142]:

$$g(z)G_1(z)=\pi_{2,0}\xi\mu_2 z+\pi_{1,0}\mu_1\Big[\xi z+\lambda_2 z(1-z)-\mu_2(1-z)\Big],$$

where $g(z)=\lambda_1\lambda_2 z^3-(\eta\lambda_2+\xi\lambda_1+\lambda_1\lambda_2+\lambda_1\mu_2+\lambda_2\mu_1)z^2+(\eta\mu_2+\xi\mu_1+\mu_1\mu_2+\lambda_1\mu_2+\lambda_2\mu_1)z-\mu_1\mu_2$, and it is proven that $g(z)$ has only one root z_0 in the interval $(0,1)$.

After some algebraic manipulations, we obtain:

$$\pi_{1,0} = \frac{\xi(\mu^*-\lambda^*)z_0}{\mu_1(1-z_0)(\mu_2-\lambda_2 z_0)}, \quad (5.45)$$

$$\pi_{2,0} = \frac{\eta(\mu^*-\lambda^*)z_0}{\mu_2(1-z_0)(\mu_1-\lambda_1 z_0)}. \quad (5.46)$$

Once the values of $\pi_{1,0}$ and $\pi_{2,0}$ have been established, the probability generating functions can be calculated as:

$$G_1(z) = \frac{\xi(\mu^*-\lambda^*)z+\pi_{1,0}\mu_1(1-z)(\lambda_2 z-\mu_2)}{g(z)}, \quad (5.47)$$

$$G_2(z) = \frac{\eta(\mu^*-\lambda^*)z+\pi_{2,0}\mu_2(1-z)(\lambda_1 z-\mu_1)}{g(z)}. \quad (5.48)$$

By using the following equation:

$$\mathbb{E}[N_i]=\sum_{n=0}^{\infty} n\pi_{i,n}=\frac{\mathrm{d}G_i(z)}{\mathrm{d}z}|_{z=1}, \quad \forall i=1,2, \quad (5.49)$$

we get the average number of jobs in the system [142]:

$$\begin{aligned}\mathbb{E}[N] &= \mathbb{E}[N_1]+\mathbb{E}[N_2]\\ &= \frac{\lambda^*}{(\mu^*-\lambda^*)}+\frac{\mu_1(\mu_2-\lambda_2)\pi_{1,0}+\mu_2(\mu_1-\lambda_1)\pi_{2,0}}{(\xi+\eta)(\mu^*-\lambda^*)}-\frac{(\mu_1-\lambda_1)(\mu_2-\lambda_2)}{(\xi+\eta)(\mu^*-\lambda^*)}, \quad (5.50)\end{aligned}$$

which contains a root of third order equation, and thus is difficult to calculate. However, when taking the balance traffic condition: $\lambda_1/\mu_1=\lambda_2/\mu_2$ into account, it can be further simplified. In such a

situation, let $\mu_1/\lambda_1 = \mu_2/\lambda_2 = \theta$, and then we substitute them into $\pi_{1,0}$, $\pi_{2,0}$ and $g(z)$, it is easy to prove that $\pi_{1,0}/\pi_{2,0} = \pi_1/\pi_2$ and $g(\theta) = 0$. Therefore, we have the decomposition:

$$g(z) = \lambda_1\lambda_2(z-\theta)(z^2 - kz + \theta),$$

where $k = \eta/\lambda_1 + \xi/\lambda_2 + 1 + \theta$. The root of interest z_0 that resides in the interval $(0, 1)$, equals $z_0 = (k - \sqrt{k^2 - 4\theta})/2$. Finally, we can get:

$$\pi_{i,0} = \pi_i \cdot (1 - 1/\theta), \quad \forall i = 1, 2,$$

and then we have $\rho = 1 - (\pi_{1,0} + \pi_{2,0}) = \lambda^*/\mu^*$. By using induction [142], it is easy to prove that:

$$\pi_{i,n} = \pi_i \cdot (1-\rho)\rho^n, \quad \forall i = 1, 2.$$

Therefore, the partial generating functions are derived as:

$$G_i(z) = \pi_i(1-\rho)\sum_{n=0}^{\infty}(z\rho)^n = \pi_i \cdot \frac{\mu^* - \lambda^*}{\mu^* - \lambda^* z}, \quad \forall i = 1, 2,$$

and then by using (5.49), we obtain:

$$\mathbb{E}[N_1] = \pi_1 \cdot \frac{\lambda^*}{\mu^* - \lambda^*}, \tag{5.51}$$

$$\mathbb{E}[N_2] = \pi_2 \cdot \frac{\lambda^*}{\mu^* - \lambda^*}. \tag{5.52}$$

The mean number of jobs in cloud queue is calculated as:

$$\mathbb{E}[N_c] = \frac{\lambda_c}{\mu_c}. \tag{5.53}$$

Since $\Pr\{e_i = 1\} = \pi_i$, according to (5.35), we have:

$$\Pr\{N_i > 0, e_i = 1\} = \rho_i \cdot \pi_i, \quad \forall i = 1, 2. \tag{5.54}$$

Further, by substituting (5.51)-(5.54) into (5.36), we can formulate the explicit expressions of the ERWP metric for the uninterrupted offloading strategy.

Generic Job Size Distribution Approximation

So far our analysis considers exponentially distributed job sizes. Unfortunately, generalizing the above 2D chain analysis for generic distribution is rather hard. Nevertheless, we can use the $M/G/1$ Pollaczek-Khintchine mean value formula as a guideline to introduce a similar "correction factor" related to variation in job sizes.

Let C_S^2 denote the squared coefficient of variation for the job size distribution, $\mathbb{E}[N]$ denote the mean number of jobs for exponentially distributed packet sizes (as derived before). The average number of jobs for the generic packet size distributions in the ordinary $M/G/1$ queue can be approximated by [79]:

$$\mathbb{E}[N_g] \approx \rho + \frac{1+C_S^2}{2} \cdot \mathbb{E}[N_q] = \frac{1-C_S^2}{2} \cdot \rho + \frac{1+C_S^2}{2} \cdot \mathbb{E}[N], \tag{5.55}$$

where $\mathbb{E}[N_q] = \mathbb{E}[N] - \rho$ is the average queueing length for the exponentially distributed packet sizes.

5.3.3 Interrupted Offloading Strategy

The interrupted offloading strategy is as shown in Fig. 5.2. The Markov chain of Queue 2 is depicted in Fig. 5.13, which is equivalent to assuming that $\lambda_1 = \lambda_2$ and $\mu_1 = 0$ in Fig. 5.12. We assume that the service station fails from time to time and resumes its operation after a random time. Writing the balance equations for this chain gives:

$$\begin{aligned}
\pi_{\text{OFF},0}(\lambda_2 + \eta) &= \pi_{\text{ON},0}\xi, && \text{(5.56a)}\\
\pi_{\text{ON},0}(\lambda_2 + \xi) &= \pi_{\text{OFF},0}\eta + \pi_{\text{ON},1}\mu_2, && \text{(5.56b)}\\
\pi_{\text{OFF},n}(\lambda_2 + \eta) &= \pi_{\text{OFF},n-1}\lambda_2 + \pi_{\text{ON},n}\xi \quad (n \geq 0), && \text{(5.56c)}\\
\pi_{\text{ON},n}(\lambda_2 + \xi + \mu_2) &= \pi_{\text{ON},n-1}\lambda_2 + \pi_{\text{ON},n+1}\mu_2 + \pi_{\text{OFF},n}\eta \quad (n \geq 0). && \text{(5.56d)}
\end{aligned}$$

After simple algebraic operations of equations in (5.56) and summation over all n, we obtain:

$$\pi_{\text{ON},0} = \pi_{\text{ON}} - \frac{\lambda_2}{\mu_2}, \tag{5.57}$$

where $\pi_{\text{ON},0} = \pi_{2,0}$, $\pi_{\text{ON}} = \pi_2$ and $\pi_{\text{OFF}} = \pi_1$.

According to (5.40) and (5.41), we have $\lambda^* = \lambda_2$ and $\mu^* = \pi_2\mu_2$. After substituting the above

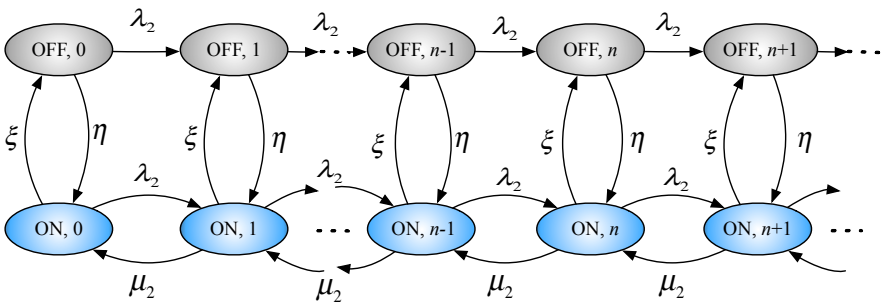


Figure 5.13: The Markov chain of Queue 2 for the interrupted offloading strategy

values into (5.50), we derive the mean number of jobs in Queue 2 as:

$$\begin{aligned}\mathbb{E}[N_2] &= \mathbb{E}[N_{\text{OFF}}] + \mathbb{E}[N_{\text{ON}}] \\ &= \frac{\lambda^*}{(\mu^* - \lambda^*)} + \frac{\mu_2\lambda_2\pi_{2,0} - (-\lambda_2)(\mu_2 - \lambda_2)}{(\xi+\eta)(\mu^* - \lambda^*)} \\ &= \frac{\lambda_2}{\pi_2\mu_2 - \lambda_2} + \frac{\pi_1\lambda_2\mu_2}{(\pi_2\mu_2 - \lambda_2)(\xi+\eta)},\end{aligned} \tag{5.58}$$

which is equivalent to the queuing formula obtained in (5.9).

Similarly, the cloud queue is a pure delay station at which jobs spend an exponentially distributed amount of time with mean equal to $1/\mu_c$ time units.

Since Server 1 is always available, we have $\Pr\{e_1 = 1\} = 1$ and the fraction of time that Server 2 is available to process jobs is: $\Pr\{e_2 = 1\} = \frac{\eta}{\xi+\eta} = \pi_2$, where as the recovery rate $\eta \to \infty$, the availability of Server 2 tends to be 1. Then we have:

$$\Pr\{N_1 > 0, e_1 = 1\} = \rho_1, \tag{5.59}$$

$$\Pr\{N_2 > 0, e_2 = 1\} = \rho_2 \cdot \pi_2. \tag{5.60}$$

Further, by substituting (5.7), (5.58)–(5.60) into (5.36), we can formulate the optimization of the ERWP metric for the offloading assignment as:

$$\lambda_i^{\min} = \arg\min_{\lambda_i} \ ERWP, \tag{5.61}$$

we seek to find the arrival rate $\lambda_i^{\min}$ to Queue i such that $ERWP$ is minimized when both queues are in operation. We can apply Newton's method to find $\lambda_i^{\min}$ iteratively [111].

In other words, arriving jobs are assigned to Queue 1 and Queue 2 according to the optimized objective defined in (5.61), minimizing the ERWP metric.

5.3.4 Generalised Offloading Strategies

In the previous analysis, we considered a simple scenario with only two networks (Cellular and WiFi). However, currently different areas might be covered by different cellular network technologies, such as 2G (GSM), 2.5G (GPRS), 2.75G (EDGE), 3G, 3.5G (HSDPA), 4G (LTE), and multiple "short-range" options might exist in addition to WiFi (e.g. femto- or other small-cell technologies) [79]. The transmission rates and power consumption may vary across different network interfaces, and even within the same network technology like WiFi, the rate might be different across different access points. A mobile user might be switching between a number of different technologies and/or rates during a large time window.

We briefly discuss how the generic uninterrupted and interrupted offloading strategies could be extended to more complex scenarios, where $N > 2$ possible options a mobile device can encounter when moving between locations with different network characteristics, rather than just two. We also consider the possibility of some intermittently available networks, with switching between different interfaces.

Uninterrupted Offloading Strategy

As shown in Fig. 5.14, a single-server queuing system that oscillates between N feasible states is denoted by $f \in \{f_1, f_2, \cdots, f_N\}$. The persistence of the system at any state is governed by a random mechanism: if the system functions at state f_i, its transition to the alternative state f_j at rate $\eta_{i,j}$.

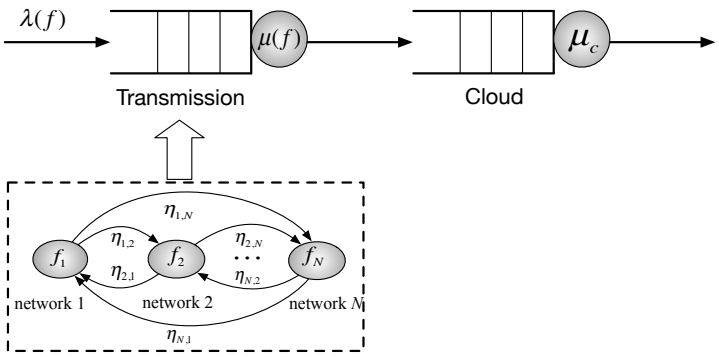


Figure 5.14: Uninterrupted offloading strategy with N heterogeneous networks

We assume that the jobs arrive as a Poisson process with rate $\lambda(f)$, job sizes are exponentially distributed and each state operates at a constant power p_i. The modulating process f determines the transition rates. If $f = f_i$, we have $\lambda(f) = \lambda_i$ and $\mu(f) = \mu_i$, where $i \in \{1, 2, \cdots, N\}$.

Since the modulating process has N states, which results in independent Poisson processes with rate λ_i, we have $\sum_{i=1}^{N} \lambda_i = \lambda_c$. The product-form for the tandem queues between the modulated (the exponential queue) and the modulating processes exists if and only if the following generalised condition holds:

$$\exists \rho \in R^{+}, \quad \text{s.t. } \forall f_i \in \{f_1, f_2, \cdots, f_N\}, \quad \frac{\lambda(f_i)}{\mu(f_i)} = \rho. \tag{5.62}$$

The Markov chain for multi-state uninterrupted offloading strategy is shown in Fig. 5.15. The possible state transitions are partially omitted for clarity of presentation. The chain moves (vertically) from state i to another state j with rate $\eta_{i,j}$. Each state corresponds to the time duration during which the mobile user is communicating through the same access network technology without interruption. The duration of each state is exponentially distributed with rate $\eta_i = \sum_{j=1, j\neq i}^{N} \eta_{i,j}$.

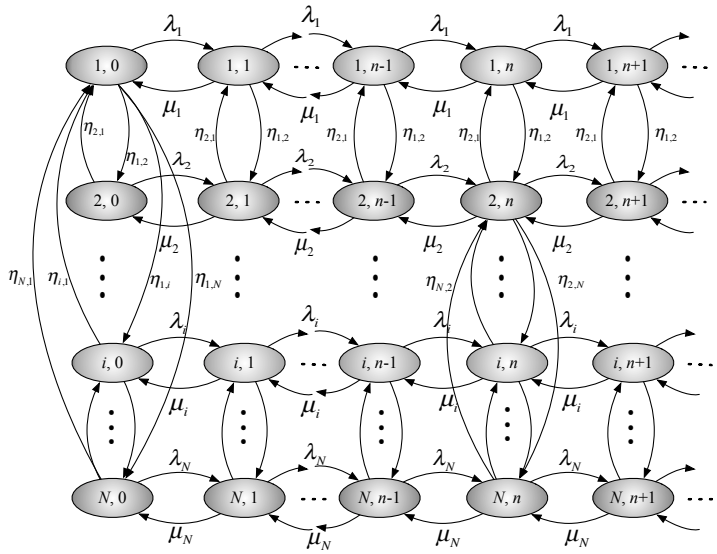


Figure 5.15: The Markov chain for multi-state uninterrupted offloading strategy

Similarly, we define:

$$\lambda^{*} = \sum_{i=1}^{N} \pi_i \cdot \lambda_i \quad \text{and} \quad \mu^{*} = \sum_{i=1}^{N} \pi_i \cdot \mu_i, \tag{5.63}$$

where $\pi_i = \eta_i / \sum_{i=1}^{N} \eta_i$ is the steady-state probability of finding the offloading system in some region with i^{th} interface availability. The partial generating functions are now derived explicitly as [140]:

$$G_i(z) = \pi_i (1-\rho) \sum_{n=0}^{\infty} (z\rho)^n = \pi_i \cdot \frac{\mu^{*} - \lambda^{*}}{\mu^{*} - \lambda^{*} z}, \quad \forall i = 1, 2, \cdots, N, \tag{5.64}$$

and then by using $\mathbb{E}[N_i] = \sum_{n=0}^{\infty} n\pi_{i,n} = dG_i(z)/dz|_{z=1}$, we obtain:

$$\mathbb{E}[N_i] = \pi_i \cdot \frac{\lambda^*}{\mu^* - \lambda^*}, \quad \forall i = 1, 2, \cdots, N. \tag{5.65}$$

Since $\Pr\{e_i = 1\} = \pi_i$, according to (5.35), we have:

$$\Pr\{N_i > 0, e_i = 1\} = \rho_i \cdot \pi_i, \quad \forall i = 1, 2, \cdots, N. \tag{5.66}$$

Further, substituting (5.65) and (5.66) into (5.36), we can formulate the explicit expressions of the ERWP metric for the uninterrupted offloading strategy with N heterogeneous networks.

Interrupted Offloading Strategy

Similarly, we can extend the interrupted offloading strategy in Fig. 5.2 to the environment with N heterogeneous networks as shown in Fig. 5.16. We assume some networks are always available such as Queue 1 and 2, while others are intermittently unavailable, e.g. Queue i and N. We model the former as $M/G/1$-FCFS queues with servers that are always on, and the latter are modelled as $M/G/1$-FCFS queues with occasional server break-down. The availability of these servers is governed by an IPP with exponentially distributed ON-OFF periods.

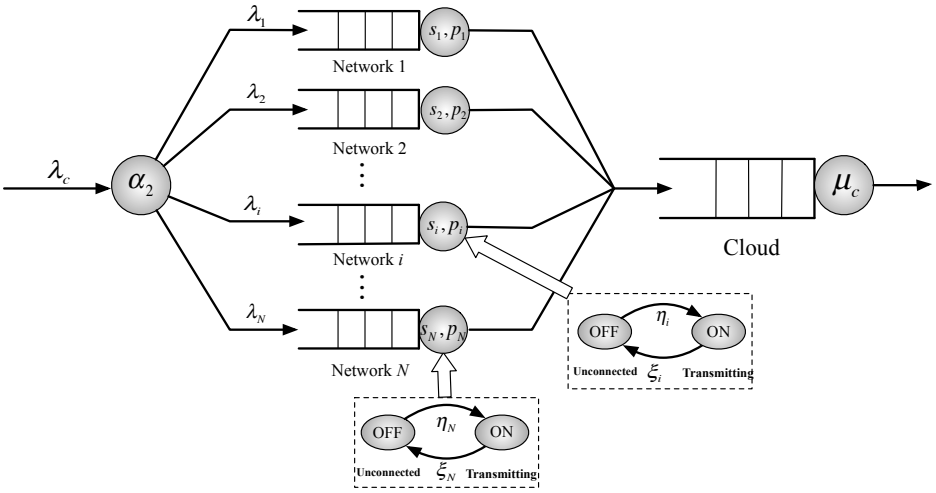


Figure 5.16: Interrupted offloading strategy with N heterogeneous networks

We can directly use the results from (5.7) for the queues with always available servers, and (5.58) for the queues with intermittently available servers. Further, by employing the ERWP metric in (5.36), the tradeoff between the mean energy consumption and mean response time can be analyzed.

5.3.5 Numerical Examples

Using measurements from real traces in [64], the average data rates for the cellular network and WiFi are set as $s_1 = 800$ Kbps and $s_2 = 2$ Mbps, respectively. The mean job size is assumed to be 125 KB, therefore the packet sizes are exponentially distributed with $X \sim \text{Exp}(1/125)$. According to the power models developed in [10], we set the power coefficients $p_1 = 2.5$ W, $p_2 = 0.7$ W and $p_m = 2$ W, respectively. Besides, suppose that the total job arrival rate for offloading is $\lambda = 0.6$ packet/s, the mobile service rate $\mu_m = 2$, the cloud service rate $\mu_c = 5$ and both the failure rate ξ and recovery rate η of Server 2 are equal to 1.

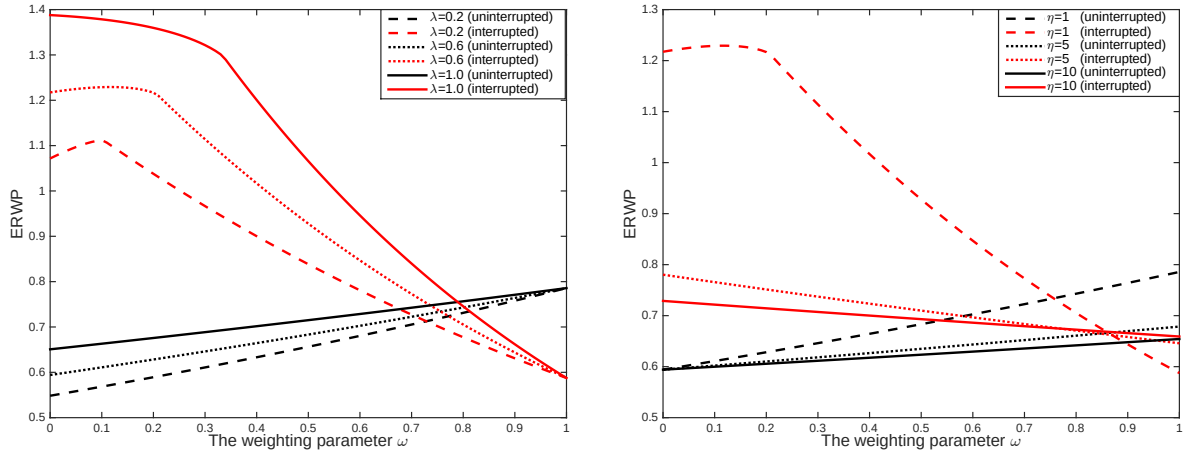
We first analyze the case when the offloading probability $\pi = 0.5$, indicating that half of the offloadable jobs are offloaded to the cloud, while the remaining jobs are executed locally on the mobile device.

From Fig. 5.17(a), we can observe that the uninterrupted offloading strategy performs significantly better than the interrupted one when ω is small, but as ω approaches to 1, the interrupted strategy performs much better. This means that when considering energy consumption more important than response time (for delay-tolerant applications), it is better to use the interrupted strategy; otherwise when considering response time more important (for delay-sensitive applications), the uninterrupted strategy is preferred, which fully uses the unavailable periods of WiFi by transmitting with a cellular network. Since energy is measured per job, when the weight is on energy consumption only the metric is for both strategies insensitive to the job arrival rate. As the arrival rate of the offloadable jobs λ increases, none of the offloading strategies can achieve a low ERWP value. However, the uninterrupted strategy varies less. The interrupted strategy is more sensitive to the job arrival rates.

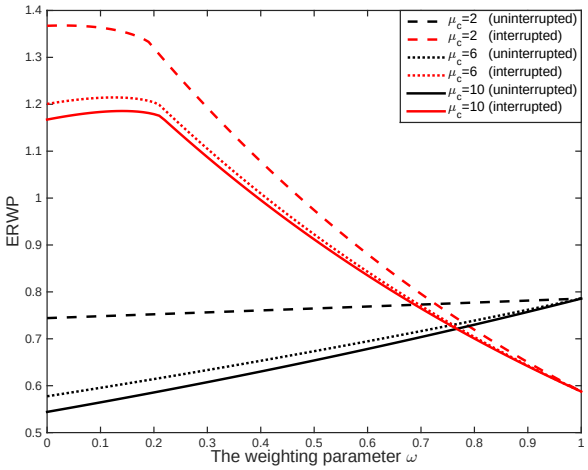
Similar observations can be made from Fig. 5.17(b) for sensitivity to the recovery rate η. The interrupted strategy suffers more from long repair times than the uninterrupted strategy. This is reasonable, due to the lower WiFi availability, resulting in most of the jobs being offloaded through the slower and more energy consuming cellular network interface. When $\omega < 0.85$, (i.e. response time is more important) the interrupted strategy also performs much better as η increases. The reason is that arriving jobs to the WiFi queue have a higher probability to be offloaded to the cloud. However, with more importance being given to the energy consumption, this strategy performs much worse as η increases and the down-times become less.

In Fig. 5.17(c) it is observed that the faster the cloud serves, the lower the ERWP value is. The cloud service rate μ_c has a great influence on the mean response time since we have to wait for the cloud service to be finished. But the cloud service rate has little influence on the energy consumption since the jobs are processed remotely rather than locally on the mobile device.

Changing the offloading probability π we find the optimal offloading decision. As shown in



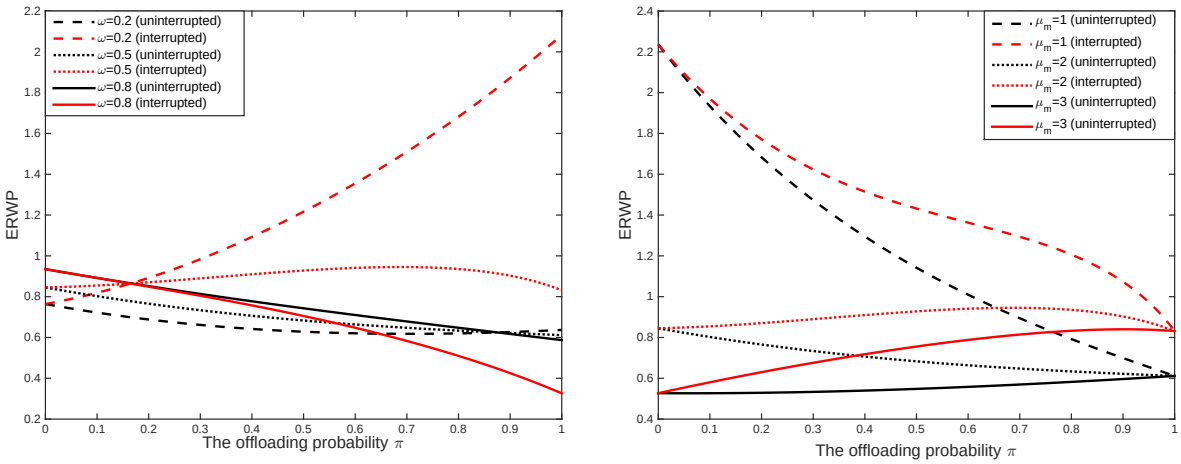
(a) Arrival rate λ (b) Recovery rate η



(c) Cloud service rate μ_c

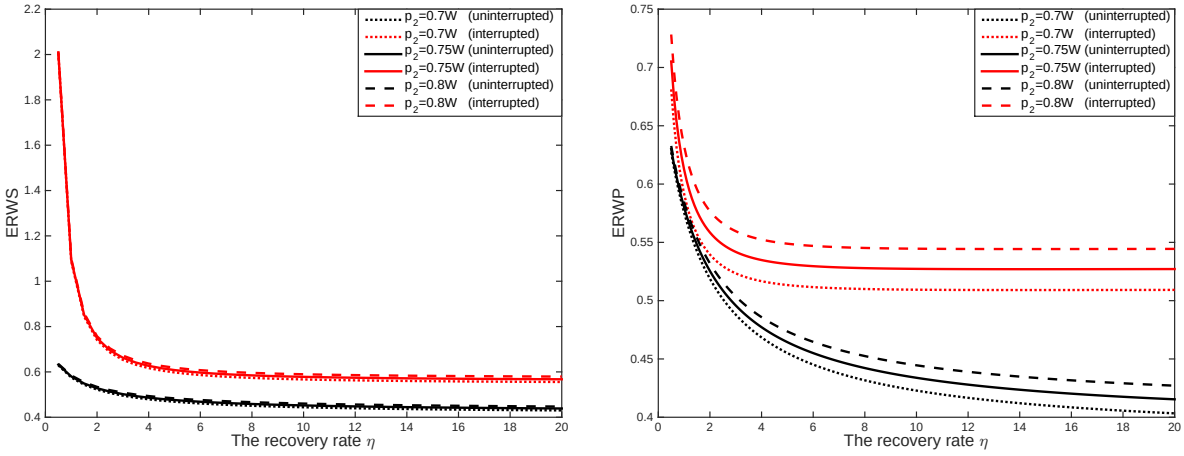
Figure 5.17: Comparison of two offloading strategies under different weighting parameters

Fig. 5.18(a), when ω is small ($\omega = 0.2$), it is not worth offloading any job in the interrupted strategy, while it is better to offload half of the offloadable jobs to the cloud in the uninterrupted strategy. However, with more focus on the energy consumption (ω approaches 1), it is better to offload all jobs for both strategies; meanwhile the interrupted offloading strategy obtains lower values for the metrics than the uninterrupted one. The ERWP metric can be treated as ERP metric when we set the weighting parameter $\omega = 0.5$, i.e. when both the energy consumption and response time have the equal importance. From Fig. 5.18(b), it is observed that the uninterrupted strategy should always be preferred. When the mobile service rate μ_m is very small, it is worthwhile to offload all the jobs



(a) Weighting parameter ω (b) Mobile service rate μ_m

Figure 5.18: Comparison of two offloading strategies under different offloading probabilities



(a) ERWS (b) ERWP

Figure 5.19: Comparison of two offloading strategies based on different metrics

to the cloud with both offloading strategies. Since the local execution is very slow it is beneficial to offload all jobs. However, when μ_m reaches some value, it is better not to offload since the cost saved from remote execution is not enough to cover the extra cost for offloading.

In Fig. 5.19 we compare the ERWP metric with the ERWS metric. When the power p_2 changes slightly, e.g. from 0.7 W to 0.8 W, it is difficult to tell the difference between the two offloading strategies according to the ERWS metric depicted in Fig. 5.19(a), while it is very clear to tell the difference according to the ERWP metric in Fig. 5.19(b). It seems that the ERWP metric is much

more sensitive to the large scale and can capture small changes when the ERWS metric has the disadvantage of a linear combination of two criteria on different scales.

5.4 Summary

The proposed queueing models address the issue of how to offload based on different wireless networks, which is used to describe complex real offloading systems.

The optimal policy can find an appropriate tradeoff between minimizing the energy costs and delay. The dynamic offloading policy derived from the tradeoff offloading policy (TOP) shows very good results and outperforms other policies like the random selection of transmission channel by a significant margin. The ERWS metric can be reduced more by considering either energy consumption or response time and it is minimal when optimizing only energy consumption.

The interrupted offloading strategy can save energy especially if the focus lies on the energy aspect. In general one can say that the uninterrupted strategy is faster, while the interrupted strategy uses less energy. For all configurations there is a break-even point at equal importance of both parts of the combined metrics for low load and recovery rate, which increases towards energy preference when the recovery rate increases (and down-time decreases) and moves towards response time preference when the load increases. While most findings correspond with intuition we see that in the interrupted strategy a slow transmission server can not benefit from fast recovery. A fast connection improves the response time much more than fast repair of a failed connection. In conclusion, short down-time of the transmission channel can mostly be tolerated.

For delay-tolerant applications, it is better to use the interrupted offloading strategy instead of the uninterrupted one, while for the delay-sensitive applications, the uninterrupted strategy shows very good results and outperforms the interrupted one by a significant margin. The offloading probability closely depends on the mobile service rate, the cloud service rate and the weighting parameter. Slow mobile service rate and fast cloud service rate will result in more jobs to be offloaded to the cloud. We can thus judiciously make the offloading decisions of whether to offload or not and how much to offload that optimize the ERWP metric.

Chapter 6

Offloading Decision Making: When to Offload

Remote execution is an opportunistic alternative, but not a must, since additional data communication is required, which may increase the response time and/or energy consumption when transferring the task related data [84]. Therefore, as sometimes it may be not worth offloading at all, decisions have to be made when encountering large communication data or low bandwidth. In this chapter, we address the issue of when to offload when considering both the energy consumption and response time. The contributions are threefold:

- Exploring the tradeoff between shortening response time and prolonging battery life of mobile devices by dividing three intervals, namely, *never offload*, *tradeoff* and *always offload*. Offloading decisions depend on whether the mobile device benefits from offloading or not.
- Proposing an energy-efficient offloading algorithm based on Lyapunov optimization which determines when and on which network to offload data so that energy-cost is minimized by leveraging delay tolerance.
- Optimality analysis of the energy-performance tradeoff for delayed offloading systems based on the Energy-Response time Weighted Product (ERWP) metric, which captures both energy and performance metrics and also intermittently available access links. We try to answer the following questions: (i) Given a deadline, how to choose the optimal offloading model and what parameters do the response time and energy depend on? (ii) How to choose the deadlines in order to optimize the ERWP by trading off the response time and energy consumption?

6.1 Tradeoff Analysis

In this section, we analyze the tradeoff between time and energy saving, and also the tradeoff between computation and communication in offloading decision making. Offloading is worthwhile when the estimated local execution cost is greater than the sum of its estimated remote execution cost on the cloud sever plus the predicted cost of transferring the related data. And offloading decision is made when encountering with large amounts of computation and relatively small amounts of communication based on time and/or energy saving criteria.

6.1.1 Time and Energy Tradeoffs

Offloading has the potential to save time and energy, but the savings from offloading need to exceed the additional communication cost between the mobile device and the cloud [82].

- *Time Saving*: the time incurred by offloading is the sum of communication time and computation time on the cloud server and it should be smaller than the execution time on the mobile device in order to save time. Therefore, it is worthwhile to offload the computation rather than execute it locally, when $t_{\mathrm{m}} > t_{\mathrm{s}} + \frac{D}{B}$ as in (4.14).
- *Energy Saving*: a performance seeking offload is not guaranteed to save energy, as the energy overhead of data transfer may exceed the energy savings from reduced CPU usage [38]. To make offloading worthwhile, it has to satisfy: $p_{\mathrm{m}} t_{\mathrm{m}} > p_{\mathrm{i}} t_{\mathrm{s}} + p_{\mathrm{tr}} \frac{D}{B}$ as in (4.15), i.e. the energy spent due to offloading must be smaller than the energy consumed by the mobile device.

To ensure that offloading will be beneficial (saving time and energy simultaneously), it has to satisfy the following two conditions:

$$t_{\mathrm{m}} > \frac{t_{\mathrm{m}}}{F} + \frac{D}{B}, \tag{6.1}$$

$$p_{\mathrm{m}} t_{\mathrm{m}} > p_{\mathrm{i}} \frac{t_{\mathrm{m}}}{F} + p_{\mathrm{tr}} \frac{D}{B}, \tag{6.2}$$

where $t_{\mathrm{m}} = F t_{\mathrm{s}}$ and again the speedup factor F indicates how powerful a cloud server is in terms of execution speed when compared with that of the mobile device. The inequalities in (6.1) and (6.2) hold under several conditions: large F that the server is much faster than the mobile device, small D that only a small amount of data is exchanged, and large B that the network bandwidth between the mobile device and the server is high [60].

In order to analyze the relation between remote execution on the cloud server and local execution using the mobile device intuitively, we apply the Energy-Response time Weighted Product (ERWP)

metric in (2.4):

$$ERWP = r_E^{\omega} \cdot r_T^{1-\omega} \tag{6.3}$$

$$= \left[\frac{p_m t_m}{p_i \frac{t_{\mathrm{m}}}{F} + p_{\mathrm{tr}} \frac{D}{B}}\right]^{\omega} \cdot \left[\frac{t_{\mathrm{m}}}{\frac{t_{\mathrm{m}}}{F} + \frac{D}{B}}\right]^{1-\omega}, \tag{6.4}$$

where r_T is the ratio of local and remote execution time, r_E is the ratio of local and remote energy cost and $ERWP$ is a performance indicator which considers both energy and time saving simultaneously. The larger $ERWP$ is, the better the offloading system works.

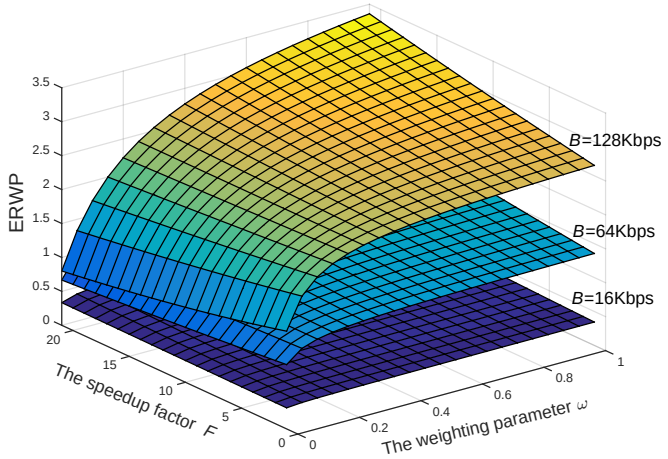


Figure 6.1: The ERWP value under different parameters

We first consider two extreme situations here. If $r_E = 1$, we have $ERWP = r_T^{1-\omega}$ that is independent of r_E, where $r_T = \frac{Fp_{\mathrm{m}}}{(F-1)p_{\mathrm{tr}}+p_{\mathrm{i}}}$. If $r_T = 1$, we have $ERWP = r_E^{\omega}$ is not related with r_E, where $r_E = \frac{Fp_{\mathrm{tr}}}{Fp_{\mathrm{m}}+p_{\mathrm{tr}}-p_{\mathrm{i}}}$. For general circumstances. We set the execution time on the mobile device $t_{\mathrm{m}} = 2$ s and the exchanged data $D = 64$ KB. It can be seen from Fig. 6.1 that the network bandwidth has a huge impact on $ERWP$, for example, when $B = 16$ Kbps, it is always below 1 no matter how F and ω change, which means offloading the program from mobile device to remote cloud can neither save energy nor reduce time when B is small. However, as B increases, $ERWP$ is also getting larger, indicating that offloading works better. And also F has the similar effect once ω is fixed, that is the larger F is, the more energy and time saving can be achieved. Besides, $ERWP$ increases slowly with the increase of F when ω is small, however, it rises faster with the increase of F when ω is large. Therefore, the weighting coefficient ω should be chosen carefully and it should be adjusted to the actual offloading systems.

For offloading break-even, i.e. when t_{m} arrives at an equilibrium point in inequality (6.1) or (6.2), if we assume t_{be1} and t_{be2} to be the critical time values, respectively, we could further derive the following two equalities:

$$t_{\mathrm{be1}} = \frac{t_{\mathrm{be1}}}{F} + \frac{D}{B} \Longrightarrow t_{\mathrm{be1}} = \frac{D/B}{1 - 1/F}, \tag{6.5}$$

$$p_{\mathrm{m}} t_{\mathrm{be2}} = p_{\mathrm{i}} \frac{t_{\mathrm{be2}}}{F} + p_{\mathrm{tr}} \frac{D}{B} \Longrightarrow t_{\mathrm{be2}} = \frac{p_{\mathrm{tr}} D/B}{p_{\mathrm{m}} - p_{\mathrm{i}}/F}. \tag{6.6}$$

Since the value of t_{be1} should be positive, from (6.5) it requires that $F > 1$. Similarly, (6.6) requires $F > p_{\mathrm{i}}/p_{\mathrm{m}}$, since $p_{\mathrm{i}} < p_{\mathrm{m}}$ (i.e. the power while being idle must be less than the active power of computing). Therefore, these constraints can always be met due to $F > 1$. Especially, when $p_{\mathrm{i}} = p_{\mathrm{tr}}$, inequality (6.2) reduces to $t_{\mathrm{m}} > \frac{p_{\mathrm{i}}}{p_{\mathrm{m}}} \left(\frac{t_{\mathrm{m}}}{F} + \frac{D}{B} \right)$, thus there is no need to discuss energy saving in inequality (6.2), as long as it meets the performance improvement in inequality (6.1). Thus, using computation offloading to shorten execution time and the prolong the battery life of the mobile device, t_{m} has to meet the following requirement:

$$t_{\mathrm{m}} > \max(t_{\mathrm{be1}}, t_{\mathrm{be2}}). \tag{6.7}$$

Moreover, in order to compare t_{be1} with t_{be2}, let

$$\frac{t_{\mathrm{be1}}}{t_{\mathrm{be2}}} = \frac{C}{1 - 1/F} \cdot \frac{p_{\mathrm{m}} - p_{\mathrm{i}}/F}{p_{\mathrm{tr}} D/B} < 1,$$

it can be seen that when $F < \frac{p_{\mathrm{i}} - p_{\mathrm{tr}}}{p_{\mathrm{m}} - p_{\mathrm{tr}}}$, we have $t_{\mathrm{be1}} < t_{\mathrm{be2}}$, and otherwise when $F > \frac{p_{\mathrm{i}} - p_{\mathrm{tr}}}{p_{\mathrm{m}} - p_{\mathrm{tr}}}$ or $p_{\mathrm{m}} = p_{\mathrm{tr}}$, we have $t_{\mathrm{be1}} > t_{\mathrm{be2}}$.

We consider two situations as shown in Fig. 6.2, where the area is divided in three intervals: *never offload*, *tradeoff* and *always offload*. As illustrated in Fig. 6.5(a) (assume $t_{\mathrm{be1}} < t_{\mathrm{be2}}$), it can be seen that when $t_{\mathrm{m}} < t_{\mathrm{be1}}$, offloading is neither beneficial for improving performance nor saving energy, and thus a program should never be offloaded to the server if its values fall into this area. When $t_{\mathrm{be1}} < t_{\mathrm{m}} < t_{\mathrm{be2}}$, offloading saves time while it costs much more energy to execute the program. However, when $t_{\mathrm{m}} > t_{\mathrm{be2}}$, shifting the complex parts of the program to a cloud server is always beneficial, therefore we should always offload if its values fall into this interval. Similarly, as depicted in Fig. 6.5(b) (assume $t_{\mathrm{be1}} > t_{\mathrm{be2}}$), we should never offload the program to the server when $t_{\mathrm{m}} < t_{\mathrm{be2}}$, but should always offload when $t_{\mathrm{m}} > t_{\mathrm{be1}}$. When $t_{\mathrm{be2}} < t_{\mathrm{m}} < t_{\mathrm{be1}}$, offloading saves energy while it takes much more time to execute the program.

Therefore, there exists a tradeoff between improving performance and saving energy in the inter-

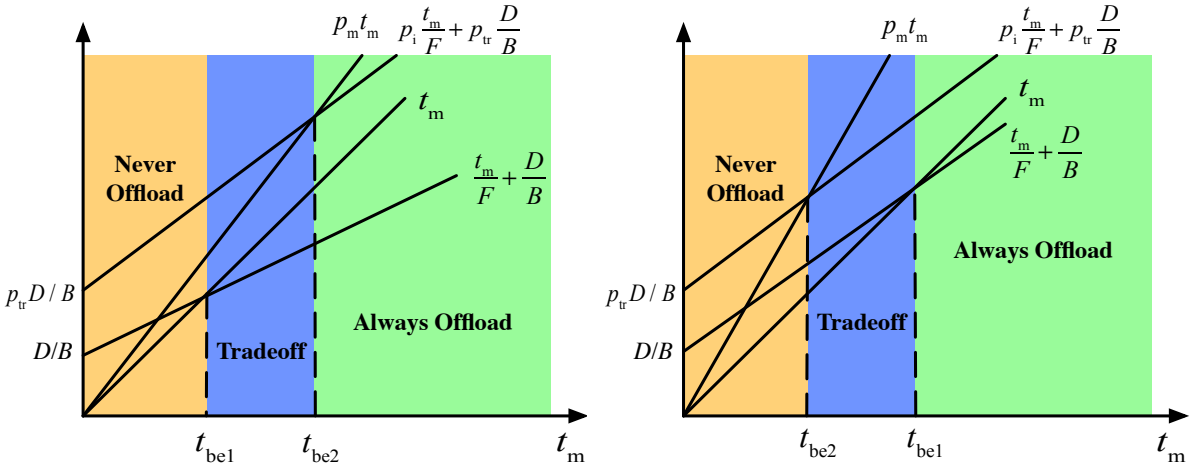


Figure 6.2: Diagram of when to offload

val between t_{be1} and t_{be2}. The values of p_m, p_i and p_{tr} are parameters specific to the mobile system. Besides, the speedup factor F is determined by the given mobile device and server. The value of F is elastic since different numbers of processors can be obtained from the cloud on demand. The larger F is, the more resources are needed to ensure a certain speedup and this costs more. When a certain value of F meets the requirement of the system, it is worth offloading a program to such a server on cloud. Furthermore, we can compare F with $\frac{p_i - p_{tr}}{p_m - p_{tr}}$, and thus the relationship between t_{be1} and t_{be2} is known.

6.1.2 Computation and Communication Tradeoffs

According to previous research work [43, 76] in our group, an offloading decision engine based on different decision criteria is developed to capture the tradeoff between computation and communication.

At the cloud side, a server of Freie Universität Berlin is used, which has 4 cores of type Intel Xeon CPU E5649 2.53 GHz, with main memory of 7786 MB. The server runs Apache Tomcat 6 and uses Java 1.6. At the mobile side, actual mobile devices (see Table 2.1) are applied in mobile cloud environments with various mobile communication networks. The server is about 17 times faster than the slow device (Xiaomi Redmi 2) and 1.1 times faster than the fast device (Samsung Galaxy S6). The communication with the server is based on the basic query/response structure. PowerTutor is used for battery usage calculations.

Figure 6.3 shows an overview of the offloading decision engine. The left part shows all relevant

parameters, such as speedup factor, wireless bandwidth, network and server availability information. On the middle part, we can choose different amounts of computation (expressed by floating-point operations per second (FLOPS)) and communication data (MB). Offloading decisions can be made based on one of the three criteria, i.e. *time saving*, *energy saving*, and *time & energy saving*. The right part shows the estimated and real cost for both the local execution and the remote execution. The engine decides whether the task should be offloaded or not, depending on which estimated option (local or remote) has relatively lower cost.

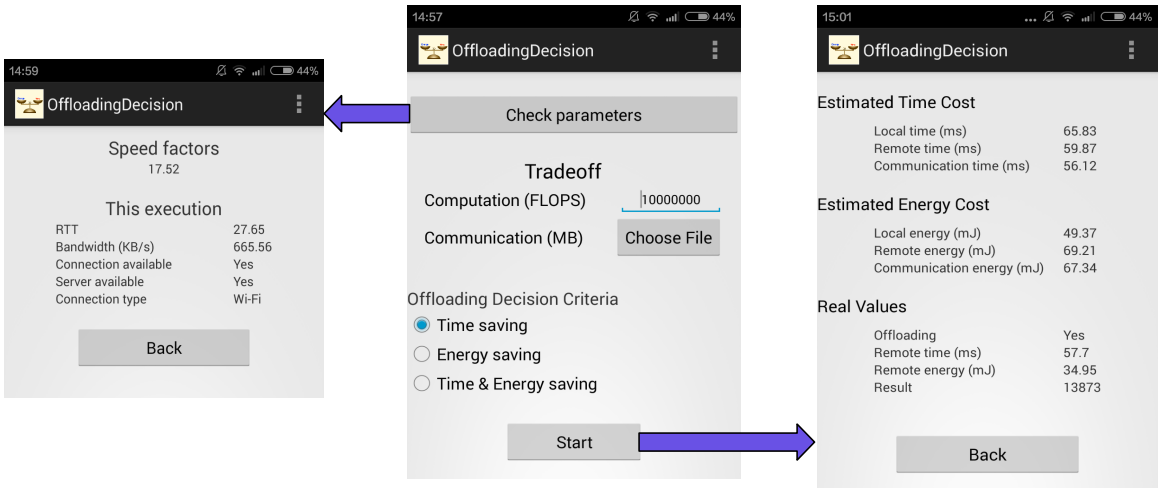
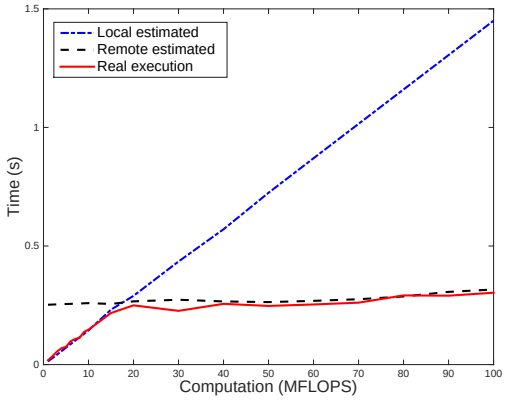


Figure 6.3: The offloading decision engine based on different criteria

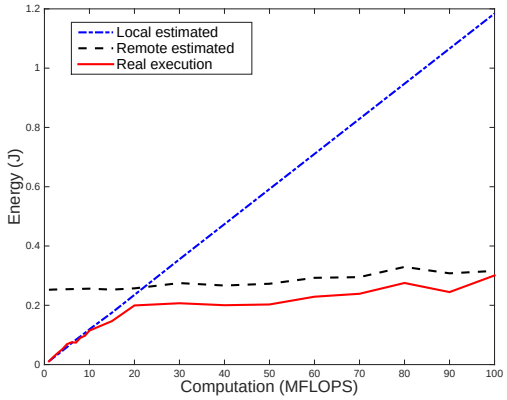
From Figs. 6.4-6.5 it can be observed that the real costs have a good match with the estimates produced by the offloading decision engine, when taking into account both the bandwidth and RTT. It is also of interest to observe the point where offloading starts being superior. For a small amount of data (100 KB), only about 17 MFLOPS are needed to reach this point, while the point arrives around 100 MFLOPS for a large volume of data (1 MB). As the data size grows, the RTT loses relevancy and the bandwidth plays the main role. When choosing the *energy saving* criterion, it needs even more amount of computation to reach the critical point. From Fig. 6.5, the fast device can not benefit so much from offloading as the slow device does. Offloading makes sense only if the amount of computation is also very large (GFLOPS).

6.2 Dynamic Transmission Scheduling

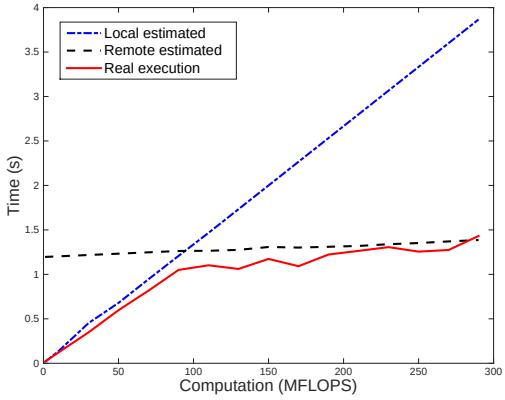
Using WiFi to offload large volumes of data from a mobile device to the cloud can be more energy-efficient than cellular radio. Since WiFi connections are not always available, we should decide



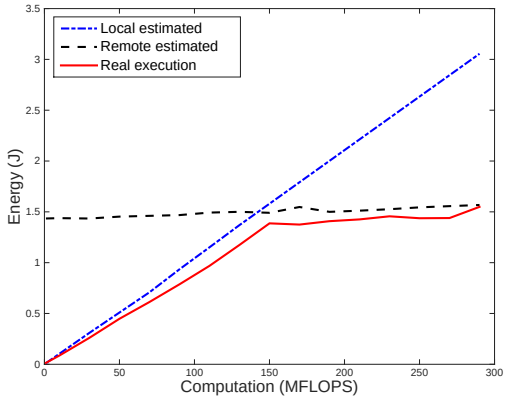
(a) Communication data 100 KB



(b) Communication data 100 KB

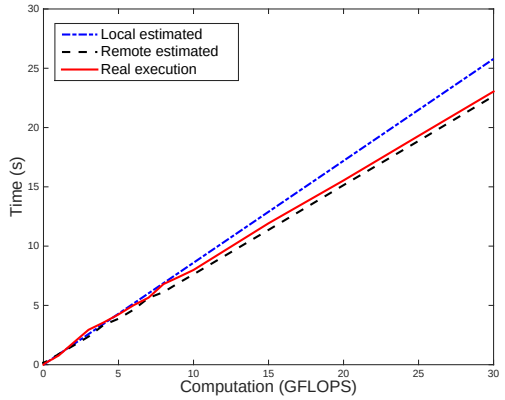


(c) Communication data 1 MB

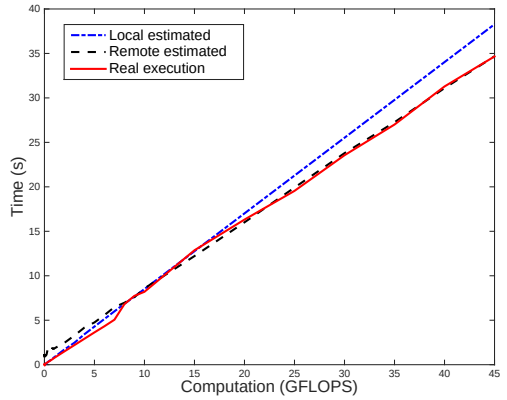


(d) Communication data 1 MB

Figure 6.4: Behavior of the slow device with different amounts of computation



(a) Communication data 100 KB



(b) Communication data 1 MB

Figure 6.5: Behavior of the fast device with different amounts of computation

when to transmit data and across which network interface.

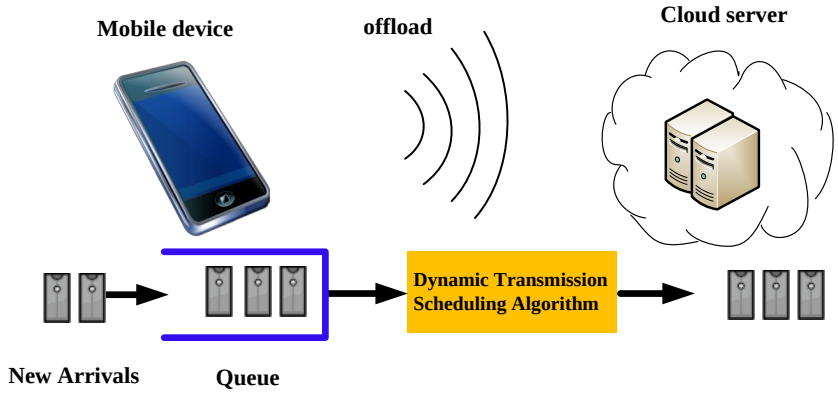


Figure 6.6: Framework of delayed offloading

As shown in Fig. 6.6, there is a queue of data to be offloaded from a mobile device to the remote cloud server. We propose a dynamic transmission scheduling algorithm based on the Lyapunov optimization. It uses the transmission energy cost as a penalty function and dynamically determines when offloading decisions are made in order to minimize the energy cost by accepting a small delay (queue length).

6.2.1 Adaptive Link Selection

As depicted in Fig. 6.7, the problem of when to offload and which interface to use can be formulated as an adaptive link selection problem. Given a set of available links with energy information, AP availability information as obtained from traces and data system queues, determine whether to use any of the available links (the appropriate network interface) to transfer data, while keeping the transmission delay bounded [103].

The mobile device selects the link with the best connection quality by running a series of probe-based tests to the cloud. Even after a particular link is selected, the connectivity can still be unstable as it is affected by user mobility, limited coverage of the WiFi APs and other factors. Because it sacrifices delay for energy, the problem of link selection and transmission scheduling for delay-tolerant applications can be naturally formulated using an optimization framework.

Suppose there are M channels available, let $B_j(t)$ denote the bandwidth between the mobile device and the cloud in time slot t when using channel j, where $j \in \{1, \cdots, M\}$. Let $b_j(t)$ or $\hat{b}_j(\alpha(t))$ denote the amount of data transmitted over channel j between the mobile device and the cloud in slot t. It is determined by a transmission decision $\alpha(t)$, which is the choice made in slot t,

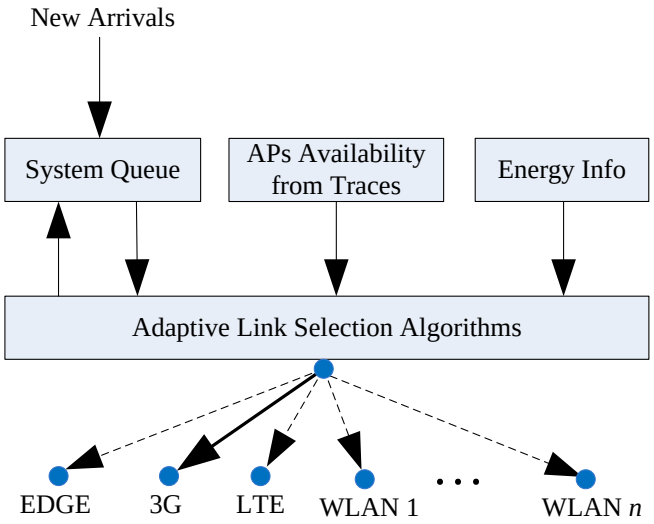


Figure 6.7: A mathematical model of adaptive link selection

either to transmit data over channel j or not to transfer, and can be expressed as:

$$b_j(t) = \hat{b}_j(\alpha(t)) = \begin{cases} B_j(t) \cdot \tau, & \text{if } \alpha(t)\text{="Transmit over channel } j\text{"}, \\ 0, & \text{if } \alpha(t)\text{="Idle"}, \end{cases} \tag{6.8}$$

where $\alpha(t)$="Idle" means that no transmission takes place in slot t and τ is the time duration that the interface is on. For convenience, τ is assumed to be a constant, which is based on the bandwidth estimation and should be neither too large to too small [107].

We denote the energy consumption caused by data transmission on the mobile device in time slot t as $E(t) = \hat{E}(\alpha(t))$, which depends on the current link bandwidth and the transmission decision $\alpha(t)$. Over a long time period T, the total amount of transmitted data is $\sum_{t=0}^{T-1}\sum_{j=1}^{M} b_j(t)$, correspondingly, the total energy consumption of the mobile device for transmitting such an amount of data can be denoted as $\sum_{t=0}^{T-1} E(t)$.

Suppose there are N queues of data to be sent from the mobile device to the cloud, and we define the vector of current queue backlogs by:

$$\boldsymbol{Q}(t) = \Big(Q_1(t), Q_2(t), \cdots, Q_N(t)\Big), \quad \forall t \in \{0, 1, \cdots, T-1\}, \tag{6.9}$$

where the queues are maintained in the mobile device's memory and for each queue i, $Q_i(t)$ represents its queue backlog of data to be transmitted from the mobile device to the cloud at the beginning of slot t.

Further, let $A_i(t)$ denote the amount of newly arriving data added to each queue i in time slot t. We assume that each random variable $A_i(t)$ is i.i.d. over time slots with expectation $\mathbb{E}\{A_i(t)\} = \lambda_i$. We call λ_i the arrival rate to queue i. Therefore, the queue length of queue i in time interval $t+1$, i.e. $Q_i(t+1)$ has the following dynamics:

$$Q_i(t+1) = \max\Big[Q_i(t) - b_i(t), 0\Big] + A_i(t), \forall i \in \{1, 2, \cdots, N\}, \forall t \in \{0, 1, \cdots, T-1\}. \quad (6.10)$$

Given this notation, we can formally state the queueing constraint that is imposed on our adaptive link selection algorithm. We require all the queues to be stable in the time average sense, i.e.

$$\bar{Q} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{N} \mathbb{E}\{Q_i(t)\} < \infty,$$

the stability constraint ensures that the average queue length is finite and we should not always defer the transmission.

While maintaining a stable queue we seek to design an adaptive link selection algorithm and dynamic transmission scheduling such that the average transmission energy is minimized [103]:

$$\min\left[\bar{E} \triangleq \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{E(t)\}\right], \quad (6.11)$$

where the required transmission energy $E(t)$ depends on the selected link during slot t.

6.2.2 Lyapunov-based Link Selection

To solve the adaptive link selection problem we employ a Lyapunov optimization framework, which enables us to derive a control algorithm that determines when and on which network to transmit our data such that the total energy-cost is minimized.

For each slot t, we define a Lyapunov function [83] as:

$$L(\boldsymbol{Q}(t)) = \frac{1}{2} \sum_{i=1}^{N} Q_i^2(t), \quad (6.12)$$

which represents a scalar measure of queue length in the network. We then define the Lyapunov

drift as the change in the Lyapunov function from one time slot to the next:

$$\begin{aligned} L(\boldsymbol{Q}(t+1)) - L(\boldsymbol{Q}(t)) &= \frac{1}{2}\sum_{i=1}^{N}\Big[Q_i^2(t+1) - Q_i^2(t)\Big] \\ &= \frac{1}{2}\sum_{i=1}^{N}\Big[\Big(\max[Q_i(t) - b_i(t), 0] + A_i(t)\Big)^2 - Q_i^2(t)\Big] \\ &\leq \sum_{i=1}^{N}\frac{A_i^2(t) + b_i^2(t)}{2} + \sum_{i=1}^{N} Q_i(t)[A_i(t) - b_i(t)]. \end{aligned} \tag{6.13}$$

The conditional Lyapunov drift $\Delta(\boldsymbol{Q}(t))$ is the expected change in the Lyapunov function over one time slot, given that the current state in time slot t is $\boldsymbol{Q}(t)$. That is:

$$\Delta(\boldsymbol{Q}(t)) = \mathbb{E}\Big\{L(\boldsymbol{Q}(t+1)) - L(\boldsymbol{Q}(t))|\boldsymbol{Q}(t)\Big\}. \tag{6.14}$$

From (6.13), we have that for a general control policy $\Delta(\boldsymbol{Q}(t))$ satisfies:

$$\Delta(\boldsymbol{Q}(t)) \leq \mathbb{E}\left\{\sum_{i=1}^{N}\frac{A_i^2(t) + b_i^2(t)}{2}|\boldsymbol{Q}(t)\right\} + \sum_{i=1}^{N} Q_i(t)\lambda_i - \mathbb{E}\left\{\sum_{i=1}^{N} Q_i(t)b_i(t)|\boldsymbol{Q}(t)\right\}, \tag{6.15}$$

where we have used the assumption that arrivals are i.i.d. over slots and hence independent of current queue backlogs, so that $\mathbb{E}\{A_i(t)|\boldsymbol{Q}(t)\} = \mathbb{E}\{A_i(t)\} = \lambda_i$.

Let C be a finite constant that bounds the first term on the right-hand-side of (6.15), so that for all t, all possible $\boldsymbol{Q}(t)$ and all possible transmission decisions we have:

$$\mathbb{E}\left\{\sum_{i=1}^{N}\frac{A_i^2(t) + b_i^2(t)}{2}|\boldsymbol{Q}(t)\right\} = \frac{1}{2}\mathbb{E}\left\{\sum_{i=1}^{N}A_i^2(t)\right\} + \frac{1}{2}\mathbb{E}\left\{\sum_{i=1}^{N}b_i^2(t)|\boldsymbol{Q}(t)\right\} \leq C. \tag{6.16}$$

There exist constants $A_{\max}^2$ and $b_{\max}^2$ that satisfy the following conditions:

$$\mathbb{E}\left\{\sum_{i=1}^{N}A_i^2(t)\right\} \leq A_{\max}^2 \quad \text{and} \quad \mathbb{E}\left\{\sum_{i=1}^{N}b_i^2(t)|\boldsymbol{Q}(t)\right\} \leq b_{\max}^2, \tag{6.17}$$

where $A_{\max} \geq A_i(t)$ represents the maximum amount of data that can arrive per time slot, and $b_{\max} \geq b_i(t)$ denotes the maximum amount of data that can be transmitted via the wireless network in a time slot. Hence, we have $C = (A_{\max}^2 + b_{\max}^2)/2$.

To stabilize the data queue by making sure that there is a balance of arriving data and transmitted

data, while minimizing the mean energy $E(t)$, we incorporate the expected energy consumption over one slot t. It can be designed to make transmission decisions that greedily minimize a bound on the following drift-plus-penalty term in each slot t [83]:

$$\Delta(\boldsymbol{Q}(t)) + V\mathbb{E}\{E(t)|\boldsymbol{Q}(t)\}, \tag{6.18}$$

where $V \geq 0$ is a control parameter that represents an "importance weight" in deciding relative importance among queue backlog, transmission rate rate, and energy cost. In other words, V can be thought of as a threshold on the queue backlog beyond which the control algorithm decides to transmit, so V controls the energy-delay tradeoff [103]. From (6.15) and (6.16) we have:

$$\begin{aligned}\Delta(\boldsymbol{Q}(t)) + V\mathbb{E}\{E(t)|\boldsymbol{Q}(t)\} &\leq C + \sum_{i=1}^{N} Q_i(t)\lambda_i + V\mathbb{E}\{E(t)|\boldsymbol{Q}(t)\} \\ &\quad -\mathbb{E}\left\{\sum_{i=1}^{N} Q_i(t)\hat{b}_i(\alpha(t))|\boldsymbol{Q}(t)\right\} \\ &= C + \sum_{i=1}^{N} Q_i(t)\lambda_i + \mathbb{E}\left\{\left[VE(t) - \sum_{i=1}^{N} Q_i(t)\hat{b}_i(\alpha(t))\right] \mid \boldsymbol{Q}(t)\right\}.\end{aligned}$$

Using the concept of opportunistically minimizing an expectation, the optimization of the right-hand-side of the above inequality is accomplished by greedily minimizing the following term:

$$\arg\min_{\alpha(t)} \left[VE(t) - \sum_{i=1}^{N} Q_i(t)\hat{b}_i(\alpha(t))\right], \tag{6.19}$$

where we choose the transmission decision $\alpha(t)$ that will minimize (6.19).

We denote a decision function as:

$$d(t) = VE(t) - \sum_{i=1}^{N} Q_i(t)\hat{b}_i(\alpha(t)), \tag{6.20}$$

which depends on the current link bandwidth and the transmission decision $\alpha(t)$. In order to understand the intuition behind this decision, we would like to see when $d(t)$ can have a low value.

1) **Link with good quality**: $d(t)$ can be small when the link has a higher estimated transmission rate. It makes sense that we would like to use any high-quality link to transfer data over a low-quality link.
2) **Queue backlog is high**: $d(t)$ can achieve a low-value if the queue backlog $\boldsymbol{Q}(t)$ is high. This

is also intuitive: when data has been in the queue for a long time, there should be a higher incentive to transmit.

3) **Link with a low energy cost**: $d(t)$ is small when the energy cost $E(t)$ of a link is low (e.g. a WiFi link). Such a link should be preferred over a high-energy cellular link [103].
4) **Control parameter** V: if V is too large, first term of $d(t)$ becomes high, then a large queue backlog or a very high quality link may be required to trigger a transmission; if V is small, the first term becomes low, then delay can be reduced at the expense of increased energy cost [103]. In general, setting V large is more likely to defer the data offloading.

Therefore, over all available links, offloading is deferred until good-quality and low-energy links become available, or unless the queue backlog is too high. Further, when considering the decision $\alpha(t)$, the decision function $d(t)$ can be denoted as:

$$d(t) = \begin{cases} VE_i(t) - Q_i(t)b_i(t), & \text{if } \alpha(t)\text{="Transmit over channel } i\text{"}, \\ 0, & \text{if } \alpha(t)\text{="Idle"}. \end{cases} \tag{6.21}$$

Performance Bounds

For any $V > 0$, we assume that the data arrival rate λ_i is strictly within the network capacity region, which is defined as the region that can be achieved by the mobile device in communication networks [103]. We can achieve a mean energy consumption and queue backlog satisfying the following constraints [39]:

$$\bar{E} = \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{E(t)\} \leq E^* + \frac{C}{V}, \tag{6.22}$$

$$\bar{Q} = \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i=1}^{N} \mathbb{E}\{Q_i(t)\} \leq \frac{C + V(E^* - \bar{E})}{\varepsilon}, \tag{6.23}$$

where $\varepsilon > 0$ is a constant denoting the distance between arrival pattern and the capacity region boundary [103], E^* is a theoretical lower bound on the mean energy consumption using any control policy that achieves queue stability.

Discussion: (6.22) and (6.23) demonstrate the tradeoff between energy consumption and queue length (or delay). The upper bound of the average energy consumption is $O(1/V)$ and the upper bound of the average queue length is $O(Vp)$. We can achieve an average energy consumption $\bar{E}$ arbitrarily close to E^* while maintaining queue stability. However, this is achieved at the expense of a larger delay because the average queue backlog $\bar{Q}$ increases linearly with V. Choosing a large

value of V can thus push the average energy arbitrarily close to its optimal value. However, this comes at an expense in average queue backlog and delay that is $O(V)$ [83]. A good V value is one that achieves a good energy and delay tradeoff, where a unit increase in V yields a very small reduction in $\bar{E}$ with consistently growing delays [103]. In mathematical terms we can choose a $k < 0$ that satisfies:

$$\frac{d(E^* + C/V)}{dV} \geq k \Longrightarrow V \geq \sqrt{\frac{C}{-k}}, \tag{6.24}$$

where k is the slope of the curve (6.22).

Proof. Because the transmission decision $\alpha(t)$ minimizes the right-hand-side of the drift-plus-penalty in inequality (6.19), in every slot t (given the observed $\boldsymbol{Q}(t)$), we have:

$$\begin{aligned}\Delta(\boldsymbol{Q}(t)) + V\mathbb{E}\{E(t)|\boldsymbol{Q}(t)\} \quad &\leq \quad C + V\mathbb{E}\Big\{\hat{E}(\alpha^*(t))|\boldsymbol{Q}(t)\Big\} + \sum_{i=1}^{N} Q_i(t)\lambda_i \\ &\quad -\mathbb{E}\Bigg\{\sum_{i=1}^{N} Q_i(t)\hat{b}_i(\alpha^*(t))|\boldsymbol{Q}(t)\Bigg\},\end{aligned}$$

where $\alpha^*(t)$ is any other (possibly randomized) transmission decision that can be made in slot t. Fixing any value $\varepsilon > 0$ in the capacity region boundary further yields:

$$\begin{aligned}\Delta(\boldsymbol{Q}(t)) + V\mathbb{E}\{E(t)|\boldsymbol{Q}(t)\} \quad &\leq \quad C + V\mathbb{E}\Big\{\hat{E}(\alpha^*(t))|\boldsymbol{Q}(t)\Big\} + \sum_{i=1}^{N} Q_i(t)\lambda_i - \sum_{i=1}^{N} Q_i(t)(\lambda_i + \varepsilon) \\ &= \quad C + V\mathbb{E}\Big\{\hat{E}(\alpha^*(t))|\boldsymbol{Q}(t)\Big\} - \varepsilon\sum_{i=1}^{N} Q_i(t).\end{aligned}$$

Taking expectations with respect to $\boldsymbol{Q}(t)$ and using the law of iterated expectations, yields:

$$\mathbb{E}\{L(\boldsymbol{Q}(t+1))\} - \mathbb{E}\{L(\boldsymbol{Q}(t))\} + V\mathbb{E}\{E(t)\} \leq C + VE^* - \varepsilon\sum_{i=1}^{N}\mathbb{E}\{Q_i(t)\},$$

where $E^* \triangleq \mathbb{E}\Big\{\hat{E}(\alpha^*(t))\Big\}$, and summing the above inequality over $t \in \{0, 1, \cdots, T-1\}$ for some positive integer T, yields:

$$\mathbb{E}\{L(\boldsymbol{Q}(T))\} - \mathbb{E}\{L(\boldsymbol{Q}(0))\} + V\sum_{t=0}^{T-1}\mathbb{E}\{E(t)\} \leq CT + VTE^* - \varepsilon\sum_{t=0}^{T-1}\sum_{i=1}^{N}\mathbb{E}\{Q_i(t)\}. \tag{6.25}$$

Then, dividing (6.25) by VT and after a simple manipulation we obtain:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{E(t)\} \leq \frac{C}{V} + E^* - \frac{\varepsilon\sum_{t=0}^{T-1}\sum_{i=1}^{N}\mathbb{E}\{Q_i(t)\}}{VT} - \frac{\mathbb{E}\{L(\boldsymbol{Q}(T))\}}{VT} + \frac{\mathbb{E}\{L(\boldsymbol{Q}(0))\}}{VT}. \quad (6.26)$$

Since the Lyapunov function is non-negative by definition and so is E^*, neglecting that we subtract non-negative quantities in (6.26) yields:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{E(t)\} \leq P^* + \frac{C}{V} + \frac{\mathbb{E}\{L(\boldsymbol{Q}(0))\}}{VT}. \quad (6.27)$$

Similarly, dividing (6.25) by εT, and after rearranging terms we obtain:

$$\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{N}\mathbb{E}\{Q_i(t)\} \leq \frac{C + V(E^* - \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{E(t)\})}{\varepsilon} + \frac{\mathbb{E}\{L(\boldsymbol{Q}(0))\}}{\varepsilon T}. \quad (6.28)$$

Finally, taking a $\limsup$ as $T \to \infty$ in inequalities (6.27) and (6.28), we can derive (6.22) and (6.23), respectively. □

6.2.3 Transmission Schedulers

To understand this link selection algorithm, we consider the two most prominent networks: WiFi and 3G. Typically, the WiFi interface is much more energy-efficient, but its availability is limited while the 3G network is available almost everywhere. Besides, channel quality can be affected by environmental factors and interference. The channel bandwidth can be reduced due to competing users in the same cell. Therefore, for data-intensive but delay-tolerant applications, we can save energy by delaying transmissions until a good-quality or a low-energy interface such as WiFi becomes available, unless the queue backlog is too high.

Table 6.1: Energy Cost Models for 3G and WiFi Networks

Items	3G	WiFi
Ramp and transfer energy $R(x)$	$0.025x + 3.5$	$0.007x + 5.9$
Tail power P	0.62 W	N/A
Tail time T	12.5 s	N/A

Table 6.1 lists the measured energy consumption models [10]. The energy needed to transmit x bytes of data over the cellular network can be split into three components: ramp energy, transmission

energy and tail energy. The energy consumption depends on the type of selected interface. Instead of transitioning from high to low power state, the 3G interface spends substantial time in the high state, which incurs considerable energy, referred to as the tail energy, while for the WiFi interface, the tail energy is zero. Using WiFi, the data transfer itself is significantly more efficient than using the 3G connection for all transfer sizes. In addition to the transfer cost, the total energy also depends on the time that the interface is on. Therefore, the energy consumption for the 3G and WiFi interfaces in time slot t can be expressed as follows:

$$E_{3G}(t) = 0.025 \cdot b_{3G}(t) + 3.5 + 0.62 \cdot 12.5, \quad (6.29)$$

$$E_{WiFi}(t) = 0.007 \cdot b_{WiFi}(t) + 5.9. \quad (6.30)$$

Transmission Scheduler I ($N \neq M$)

The model of the transmission scheduler I for only one queue of arriving jobs is depicted in Fig. 6.8. The arrival vector $A(t)$ is assumed to be i.i.d over the time slot and $\mathbb{E}\{A(t)\} = \lambda$.

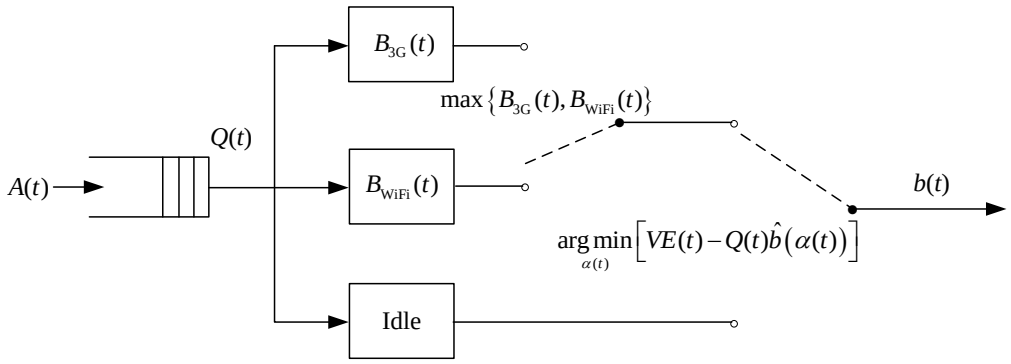


Figure 6.8: Model of transmission scheduler I

We take transmission decisions according to the estimate of the current network bandwidth. In Fig. 6.8 "$B_{3G}(t)$" represents the estimated 3G bandwidth in slot t, "$B_{WiFi}(t)$" represents the estimated WiFi bandwidth and "Idle" denotes that no transmission takes place in time slot t. If $B_{3G}(t)$ is larger than $B_{WiFi}(t)$, the mobile device will be linked to the 3G interface in time slot t to transmit data, otherwise it will be linked to the WiFi interface. The decision criterion can be denoted as $B(t) = \max\{B_{3G}(t), B_{WiFi}(t)\}$. According to the Lyapunov optimization, the minimization of the

average energy consumption is accomplished by greedily minimizing the following criterion:

$$\arg\min_{\alpha(t)} \left[VE(t) - Q(t)\hat{b}(\alpha(t)) \right]. \tag{6.31}$$

Denoting the decision function as $d(t) = VE(t) - Q(t)\hat{b}(\alpha(t))$, when considering the transmission decision $\alpha(t)$ we have:

$$d(t) = \begin{cases} VE(t) - Q(t)B(t) \cdot \tau, & \text{if } \alpha(t) = \text{“transmit”}, \\ 0, & \text{if } \alpha(t) = \text{“idle”}, \end{cases} \tag{6.32}$$

where $\alpha(t) \in \{\text{“transmit” and “idle”}\}$, taking on two possible values and

$$E(t) = \begin{cases} E_{\text{3G}}(t), & \text{if } \alpha(t) = \text{“transmit” and } B_{\text{3G}}(t) > B_{\text{WiFi}}(t), \\ E_{\text{WiFi}}(t), & \text{if } \alpha(t) = \text{“transmit” and } B_{\text{3G}}(t) \leq B_{\text{WiFi}}(t), \\ 0, & \text{if } \alpha(t) = \text{“idle”}. \end{cases}$$

If the transmission decision is $\alpha(t) =$ “transmit”, we choose to transfer data according to the current channel bandwidth. If $\alpha(t) =$ “idle”, no data is transmitted in slot t, so $E(t) = 0$ and $b(t) = 0$, and then we have $d(t) = 0$. Therefore, transmission takes place only if V satisfies: $VE(t) - Q(t)\hat{b}(\alpha(t)) < 0$. This happens when the bandwidth is high, making a large $\hat{b}(\alpha(t))$, or the queue $Q(t)$ is already congested in time slot t.

Over time, the queuing dynamic is given by:

$$Q(t+1) = \max[Q(t) - b(t), 0] + A(t), \quad \forall t \in \{0, 1, \cdots, T-1\}. \tag{6.33}$$

By Little’s Theorem [13], the average delay can be calculated as:

$$\bar{D} = \bar{Q}/\lambda. \tag{6.34}$$

The disadvantage of transmission scheduler I is that only the estimated bandwidth of 3G and WiFi in time slot t is considered and the energy usage of 3G and WiFi is not taken into account. For example, if $B_{\text{3G}}(t) = 50$ Kbps and $B_{\text{WiFi}}(t) = 49.99$ Kbps, since $B_{\text{3G}}(t)$ is larger than $B_{\text{WiFi}}(t)$ we choose the 3G interface to transmit data, even though it consumes much more energy than WiFi. In this situation we should also consider the energy demand of 3G and WiFi.

Transmission Scheduler II ($N \neq M$)

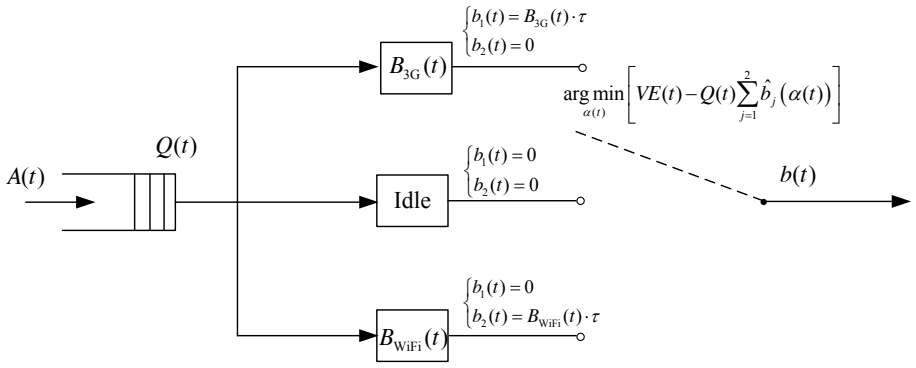


Figure 6.9: Model of optimal transmission scheduler II

The model of transmission scheduler II is as shown in Fig. 6.9. There are two links ($M = 2$) available for selection. We also use one queue ($N = 1$) to represent data transmission during each slot. Using the concept of opportunistically minimizing the expectation, the minimization of average energy consumption is accomplished by greedily minimizing:

$$\arg\min_{\alpha(t)} \left[VE(t) - Q(t) \sum_{j=1}^{M} \hat{b}_j(\alpha(t)) \right]. \tag{6.35}$$

Similarly, let $d(t) = VE(t) - Q(t) \sum_{j=1}^{M} \hat{b}_j(\alpha(t))$. Since $M = 2$, there are three possible results according to the transmission decision of $\alpha(t)$

$$d(t) = \begin{cases} VE_{3G}(t) - Q(t) B_{3G}(t) \cdot \tau, & \text{if } \alpha(t) = \text{“transmit via 3G"}, \\ VE_{WiFi}(t) - Q(t) B_{WiFi}(t) \cdot \tau, & \text{if } \alpha(t) = \text{“transmit via WiFi"}, \\ 0, & \text{if } \alpha(t) = \text{“idle"}, \end{cases} \tag{6.36}$$

where $\alpha(t) \in \{\text{“transmit via 3G”, “transmit via WiFi” and “idle”}\}$ is the transmission decision in slot t, taking on three possible values. We not only consider the estimated bandwidth but also take into account the energy usage of 3G and WiFi in time slot t. We thus compare the above values and choose the transmission decision corresponding to the smallest outcome.

If the 3G and WiFi interfaces can be used simultaneously, the transmission scheduler model in Fig. 6.9 can be further extended as in Fig. 6.10. Since the combined transmission works just like an extra channel, we have $M = 3$. Thus, there are four possible results in (6.35) according to the

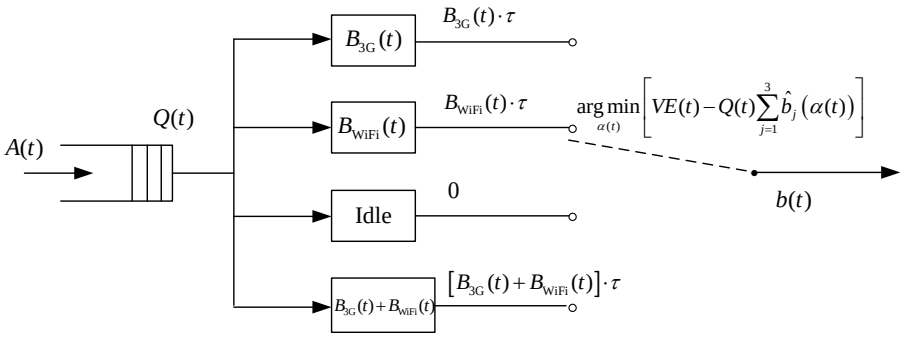


Figure 6.10: Model of transmission scheduler II for the combined scheme

transmission decision of $\alpha(t)$:

$$d(t)=\begin{cases} VE_{3G}(t)-Q(t)B_{3G}(t)\cdot\tau, & \text{if } \alpha(t)\text{="transmit via 3G",} \\ VE_{WiFi}(t)-Q(t)B_{WiFi}(t)\cdot\tau, & \text{if } \alpha(t)\text{="transmit via WiFi",} \\ V\Big[E_{3G}(t)+E_{WiFi}(t)\Big]-Q(t)\Big[B_{3G}(t)+B_{WiFi}(t)\Big]\cdot\tau, & \text{if } \alpha(t)\text{="transmit via 3G and WiFi",} \\ 0, & \text{if } \alpha(t)\text{="idle",} \end{cases}$$

where $\alpha(t)\in\{$"transmit via 3G", "transmit via WiFi", "transmit via 3G and WiFi", and "idle"$\}$ is the transmission decision in slot t, taking on four possible values.

Transmission Scheduler III ($N=M$)

The model of transmission scheduler III is depicted in Fig. 6.11. The number of channels is equal to the number of queues, that is $N=M=2$. $A_1(t)$ is only transmitted through the 3G interface while $A_2(t)$ is only transmitted through the WiFi interface. We assume that $A_1(t)$ and $A_2(t)$ take integer units of packets, the arrival vector $A(t)$ is i.i.d over slot and $\mathbb{E}\{A(t)\}=\lambda$. The question whether or not to allocate $A(t)$ to $A_1(t)$ and $A_2(t)$ in equal shares still remains. To analyze this problem, we simplify the model as shown in Fig. 6.12, such that it involves routing decisions besides scheduling decisions.

There are two separate queues depicted in Fig. 6.12, the arrival vectors $A_1(t)$ and $A_2(t)$ are i.i.d over all slots, $\mathbb{E}\{A_1(t)\}=\lambda_1$ and $\mathbb{E}\{A_2(t)\}=\lambda_2$. Since $A_1(t)+A_2(t)=A(t)$, according to the property of the Poisson distribution, we have: $\lambda_1+\lambda_2=\lambda$, where $\lambda_1=\rho\lambda$, $\lambda_2=(1-\rho)\lambda$, and $0\leq\rho\leq 1$ is defined as the dispatching ratio of arrival rate to queue 1. There are two extreme cases: when $\rho=0$, the mobile device only uses the WiFi interface to transmit data and when $\rho=1$, the mobile device only uses the 3G interface. Similarly, using the concept of opportunistic minimization

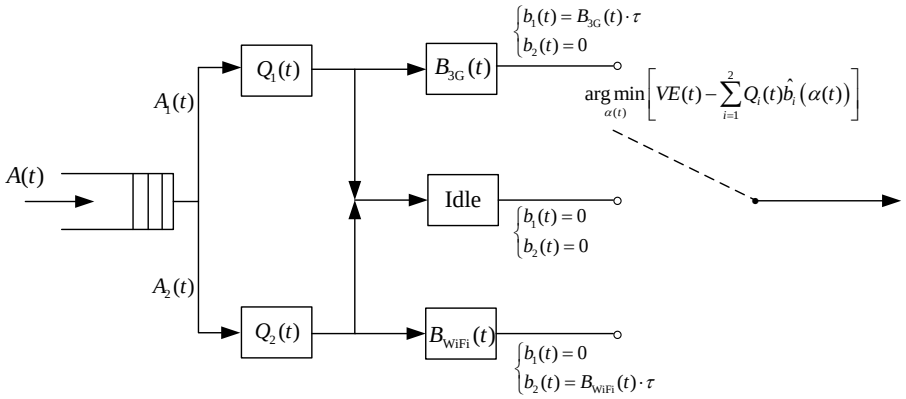


Figure 6.11: Model of transmission scheduler III

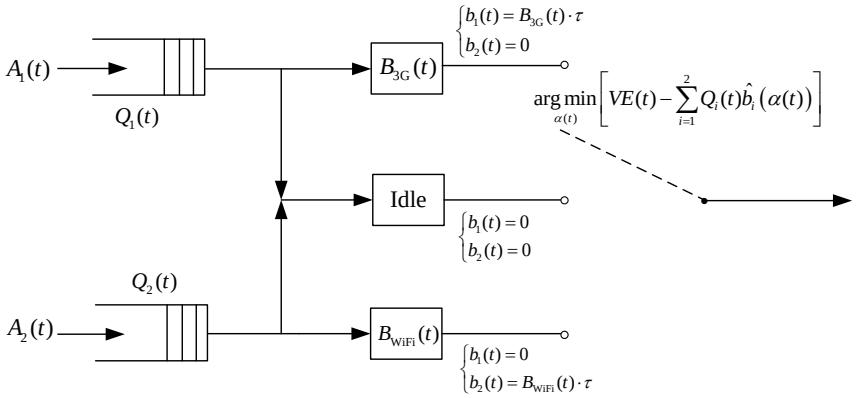


Figure 6.12: Equivalent model of transmission scheduler III

of the expectation, the minimization of the average energy consumption is accomplished by greedily minimizing:

$$\arg\min_{\alpha(t)} \left[VE(t) - \sum_{i=1}^{2} Q_i(t)\hat{b}_i(\alpha(t)) \right]. \tag{6.37}$$

Let $d(t) = VE(t) - \sum_{i=1}^{2} Q_i(t)\hat{b}_i(\alpha(t))$. Then there are three possible results according to the transmission decision of $\alpha(t)$ as given by:

$$d(t) = \begin{cases} VE_{\text{3G}}(t) - Q_1(t)b_1(t), & \text{if } \alpha(t)\text{="transmit via 3G",} \\ VE_{\text{WiFi}}(t) - Q_2(t)b_2(t), & \text{if } \alpha(t)\text{="transmit via WiFi",} \\ 0, & \text{if } \alpha(t)\text{="idle",} \end{cases} \tag{6.38}$$

where $\alpha(t) \in \{\text{“transmit via 3G”, “transmit via WiFi” and “idle”}\}$ is the transmission decision in

slot t, taking on the three possible values. The amount of data transmitted between the cloud and the mobile device in slot t is as follows:

$$\{b_1(t), b_2(t)\} = \begin{cases} \{B_{\text{3G}}(t) \cdot \tau, 0\}, & \text{if } \alpha(t)\text{="transmit via 3G"}, \\ \{0, B_{\text{WiFi}}(t) \cdot \tau\}, & \text{if } \alpha(t)\text{="transmit via WiFi"}, \\ \{0, 0\}, & \text{if } \alpha(t)\text{="idle"}, \end{cases}$$

and the queuing dynamics are given by:

$$Q_i(t+1) = \max\Big[Q_i(t) - b_i(t), 0\Big] + A_i(t),\ \forall i \in \{1, 2\},\ \forall t \in \{0, 1, \cdots, T-1\}. \tag{6.39}$$

Similarly, the average delay for this system is calculated as:

$$\bar{D} = \frac{\overline{Q_1 + Q_2}}{\lambda_1 + \lambda_2}. \tag{6.40}$$

Furthermore, the transmission scheduler III can be extended in the same way to more general scenarios as depicted in Fig. 6.7, where traffic queues can be concurrently distributed over several communication channels.

6.2.4 Simulation Results

We assume data arrivals follow a Poisson process with λ packets/minute and the size of each packet is 100 KB. The energy consumption models refer to (6.29) and (6.30) for the 3G and WiFi interfaces, respectively. Since data communication time between the mobile device and the cloud depends on the network bandwidth and the bandwidth of WLAN is remarkably higher than the bandwidth provided by radio access on a mobile device, the bandwidths for the 3G and WiFi interfaces follow uniform distributions on [1, 100] and [1, 300] KB/s, respectively. Suppose that the network bandwidths stay the same during each time slot, which is set as $\tau = 60$ s. Our algorithms are simulated in 1000 time slots for each of the V value ranging from 1 to 300.

From Fig. 6.13 (transmission scheduler I, refer to Fig. 6.8), the average energy consumption and transmit data fall quickly at the beginning and then tend to descend slowly while the average queue backlog grows linearly with V. This finding confirms the $[O(1/V),\ O(V)]$ tradeoff as captured in (6.22) and (6.23). According to different scenarios, we can adjust the value of V to control the energy-delay tradeoff. When the power constraint is stringent (e.g. the mobile device is running out of battery and no charger is available), choosing a larger V can save energy at the expense of longer average queue length and larger delay and instead, when the battery supply is not so limited (e.g. a

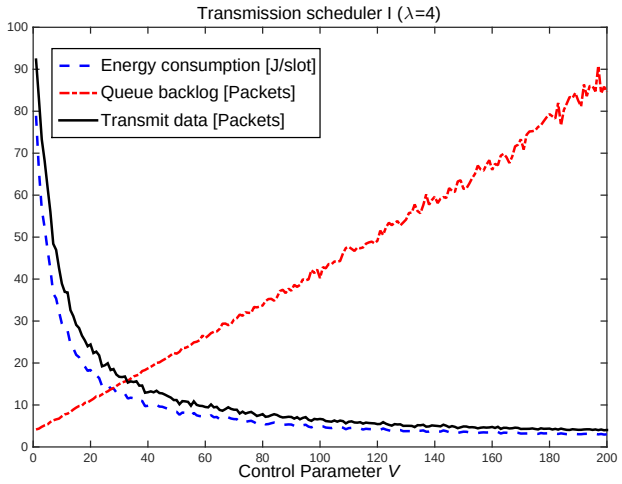


Figure 6.13: The impact of V on mean energy consumption, queue backlog and transmit data for transmission scheduler I

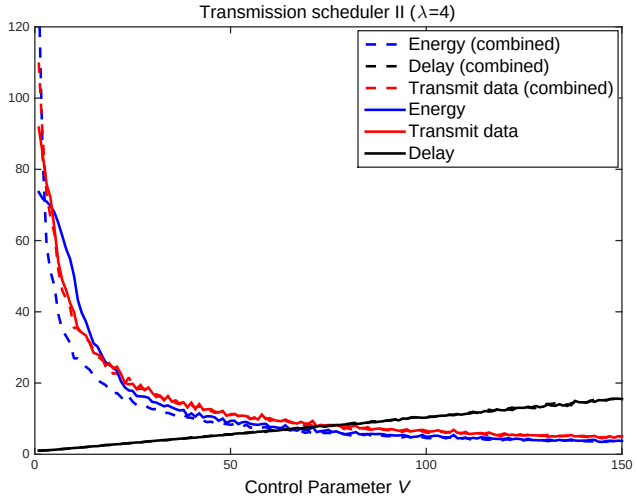
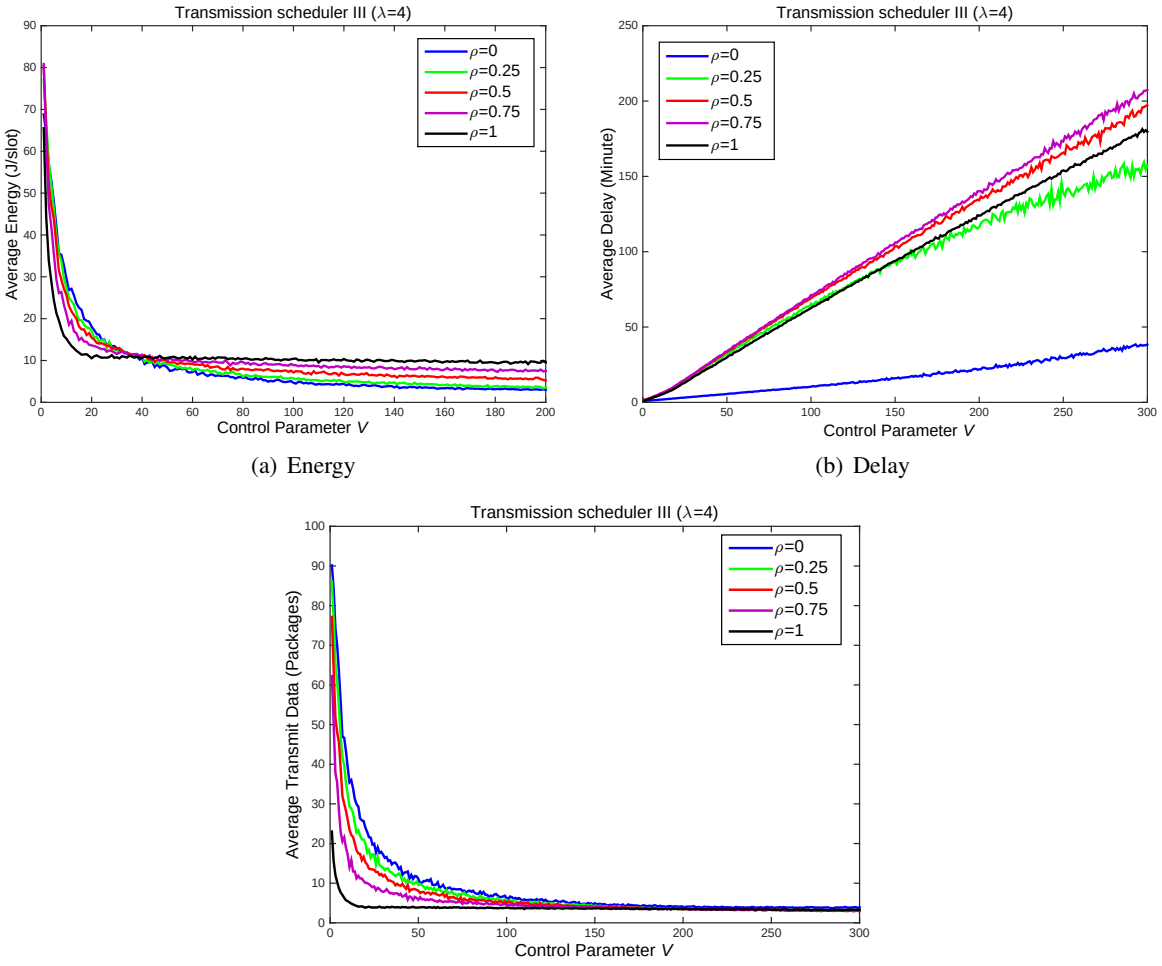


Figure 6.14: Comparison of different schemes for transmission scheduler II

charger is available), we can increase V to reduce the delay. Especially, there exists a sweet spot of V, and at this point, the marginal energy conservation is not worth the consistently growing delay with increasing of V. For example, when V increases from 100 to 200, the energy consumption has only a small decline while the delay increases significantly, thus we should trade energy with delay. Further, according to (6.24), the slope of the curve is $k \approx 0$ at this point.

The results of using transmission scheduler II are depicted in Fig. 6.14. We compare the scheme that combines 3G and WiFi (refer to Fig. 6.10) with the one that transmits separately (refer to Fig. 6.9). It can be seen that the average number of transmitted packets and average delay in both



(a) Energy

(b) Delay

(c) Data

Figure 6.15: The impact of V on mean energy consumption, delay and transmit data

schemes almost coincide with each other while the combined scheme achieves a lower average energy consumption than both individual schemes when the control factor V is small (e.g. $V \leq 75$).

The results of using transmission scheduler III for the scenario (refer to Fig. 6.11) are depicted in Fig. 6.15. In Fig. 6.15(a) when V is small, it has the minimum energy consumption when only using 3G ($\rho = 1$) for data transfer, while it has the maximum energy consumption when only using WiFi ($\rho = 0$). The mean energy increases with the increase of ρ when $V \leq 37$. However, when V arrives at a certain value ($V \approx 37$), the scheme that only uses 3G for data transfer has the minimum energy consumption while the one that only uses WiFi has the maximum value. The mean energy consumption increases with the increase of ρ when $V > 37$. Therefore, the energy consumption for

such a transmission scheduler closely depends on the value of ρ. The average delay in Fig. 6.15(b) is minimal when only using WiFi to transmit data. As ρ increases, the average delay first increases, but it then decreases after ρ arrives at some value, for example, the average delay is smaller for $\rho = 1$ than for $\rho = 0.75$. When V is small, the average transmit data depicted in Fig. 6.15(c) decreases with the increment of ρ, thus the mobile device can transfer the largest amount of data when only using the WiFi interface to transmit data due to its high bandwidth. However, when V is large, the average transmit data is almost the same and does not change when increasing V.

6.3 Delayed Offloading Model

In a heterogeneous wireless environment, the cellular interface can provide ubiquitous coverage for mobile devices in a wide area, but has lower data transmission rate and consumes more transmission energy than the WiFi interface. Therefore, by delaying transmission until WiFi is available, there are opportunities to reduce the transmission time (while bringing in extra waiting time). The reduced transmission times are directly translated into battery power saving for the mobile devices [64].

In delayed offloading, each data transfer is associated with a deadline, and the data transfer is resumed whenever getting in the coverage of WiFi until the transfer is completed [64, 123]. If the transfer does not finish within its deadline, the task will either be executed locally or cellular networks will finally complete the transfer. The delayed offloading model involves queueing with reneging and service interruptions. In queueing, reneging means that a job will leave the queue and join another queue after the deadline expires. Service interruption literally means unwilling discontinuity of service in the queue, and this models connection and disconnection periods of a mobile device to WiFi networks in the system [55].

According to the WiFi availability model in Fig. 5.10, we build two types of delayed offloading models as follows:

- **Partial Offloading Model**: we employ a single queue with two phases (the fast phase with WiFi network and the slow phase with cellular network) to offload jobs to the cloud server. When there is a WiFi connection available, all the offloadable jobs are sent over the WiFi network; otherwise, they are sent over the cellular interface as the cellular network is always available. We set a reneging deadline in the cellular network. If the deadline expires before the job switches over to some WiFi AP, then it is executed locally on the mobile device rather than remotely in the cloud [64]. By doing this, we have partial jobs offloaded to the cloud and the remaining ones processed locally.
- **Full Offloading Model**: when there is a WiFi connection available, all the offloadable jobs

are sent over the WiFi network; otherwise, they can be delayed up to a given deadline, or until WiFi becomes available. If the deadline expires before the job can be transmitted over some WiFi AP, then it is offloaded through cellular network. Therefore, we have all the offloadable jobs offloaded to the cloud via the cellular or WiFi network.

6.3.1 Partial Offloading Model

Figure 6.16 depicts a delayed offloading model based on the WiFi network's availability model. We consider an $M/M/1$ modulated queue in a two-phase (fast and slow) Markovian random environment, with impatient jobs. The jobs are offloaded either via a cellular connection or a WiFi network to the cloud. The single-server queuing system that oscillates between two feasible phases is denoted by f_{ON} and f_{OFF}. The persistence of the system at any phase is governed by a random mechanism [11]: if the system functions at phase f_{ON}, it switches to phase f_{OFF} after random time of mean duration $1/\xi$; if the system functions at phase f_{OFF}, it switches to the other phase after random time of mean duration $1/\eta$.

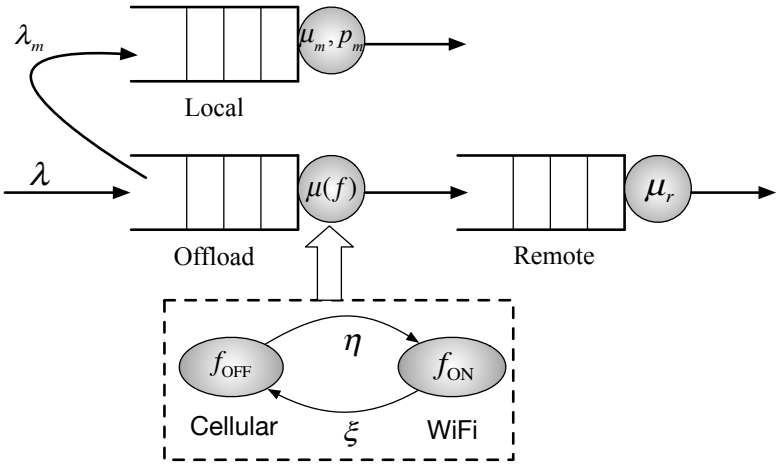


Figure 6.16: The partial offloading model

We assume that offloading jobs arrive at the system according to a Poisson process with rate λ arrive in a Poisson process with rate λ, and the modulating process $f \in \{f_{\text{ON}}, f_{\text{OFF}}\}$ determines the transition rates:

$$\mu(f) = \begin{cases} \mu_c, & \text{if } f = f_{\text{OFF}}, \\ \mu_w, & \text{if } f = f_{\text{ON}}. \end{cases} \tag{6.41}$$

We assume that the mean job size is $\mathbb{E}[X]$, the transmission speed of the fast phase (WiFi network) is s_w with service rate $\mu_w = s_w/\mathbb{E}[X]$, and its operating power is p_w when serving jobs and zero

whenever idle. Similarly, the corresponding speed for the slow phase (cellular network) is s_c with service rate $\mu_c = s_c/\mathbb{E}[X]$ ($\mu_c \leq \mu_w$), and its operating power is p_c. When in the slow phase, jobs become impatient. A reneging deadline, is associated with each job in this phase. That is, each job, upon arrival, activates an individual timer, exponentially distributed with an reneging rate r. 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 queue and is assumed to be executed locally on the mobile device rather than being offloaded to the cloud [96].

Therefore, Figure 6.16 demonstrates that the delayed offloading model consists of a transmission queue (with two alternating states of cellular and WiFi), a local processing queue and an exponential queue representing the cloud. However, when the job waits in the cellular network for too long time, we choose to process it locally on the mobile device. If the job in the offload queue is completely transmitted before the assigned deadline has expired, we say that the job is successfully offloaded. If offloading fails, the job leaves the offload queue and joins the local queue for immediate local processing. We call such an event a reneging event.

Queueing Analysis

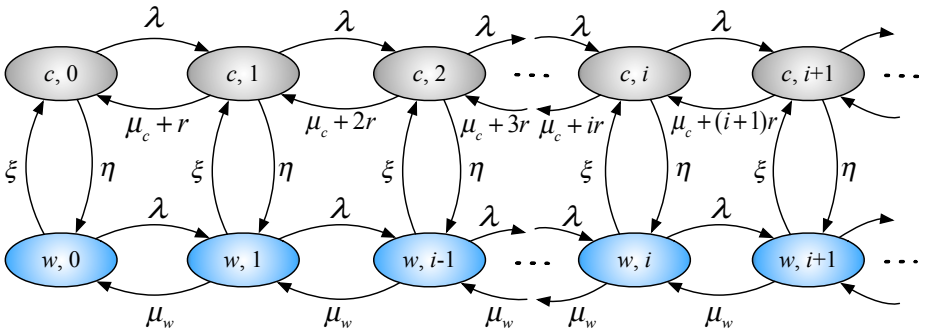


Figure 6.17: The Markov chain for the partial offloading model

Given the previously stated assumptions, the delayed offloading model can be modelled with a 2D Markov chain, as shown in Fig. 6.17. The states with cellular network are denoted $\{c, i\}$, and the states with WiFi connectivity are denoted $\{w, i\}$. The parameter i corresponds to the number of jobs in the system (queuing and in service). Writing the balance equations for the cellular and WiFi

states gives:

$$
\begin{aligned}
(\lambda+\eta)\pi_{c,0} &= (\mu_c+r)\pi_{c,1}+\xi\pi_{w,0} && (6.42a)\\
(\lambda+\eta+\mu_c+ir)\pi_{c,i} &= \lambda\pi_{c,i-1}+(\mu_c+(i+1)r)\pi_{c,i+1}+\xi\pi_{w,i}\ \ (i>0) && (6.42b)\\
(\lambda+\xi)\pi_{w,0} &= \mu_w\pi_{w,1}+\eta\pi_{c,0} && (6.42c)\\
(\lambda+\xi+\mu_w)\pi_{w,i} &= \lambda\pi_{w,i-1}+\mu_w\pi_{w,i+1}+\eta\pi_{c,i}\ \ (i>0) && (6.42d)
\end{aligned}
$$

Let μ be defined as: $\mu = \pi_c \cdot \mu_c + \pi_w \cdot \mu_w$.

$$
\begin{aligned}
\pi_c &= \frac{\mathbb{E}[T_{\text{OFF}}]}{\mathbb{E}[T_{\text{ON}}]+\mathbb{E}[T_{\text{OFF}}]} = \frac{\xi}{\eta+\xi}, && (6.43)\\
\pi_w &= \frac{\mathbb{E}[T_{\text{ON}}]}{\mathbb{E}[T_{\text{ON}}]+\mathbb{E}[T_{\text{OFF}}]} = \frac{\eta}{\eta+\xi} = AR, && (6.44)
\end{aligned}
$$

where π_c and π_w are the steady-state probabilities of finding the offloading system in some region with only cellular access and with WiFi availability, respectively. We define the probability generating functions for both cellular and WiFi states as:

$$
G_c(z) = \sum_{i=0}^{\infty} \pi_{c,i} z^i \quad \text{and} \quad G_w(z) = \sum_{i=0}^{\infty} \pi_{w,i} z^i, \quad |z| \le 1. \tag{6.45}
$$

Using (6.42) we obtain:

$$
G_w(z)\beta(z) = \eta z G_c(z) - \mu_w(1-z)\pi_{w,0},
$$

where $\beta(z) = (\lambda z - \mu_w)(1-z) + \xi z = -\lambda(z-z_1)(z-z_2)$. The roots z_1, z_2 of the quadratic polynomial $\beta(z)$ are obtained as:

$$
z_{1,2} = \frac{\lambda+\mu_w+\xi \mp \sqrt{(\lambda+\mu_w+\xi)^2 - 4\lambda\mu_w}}{2\lambda}.
$$

General Case

Assume $r \neq 0$, we have the partial offloading model. Let us define:

$$
\begin{aligned}
\kappa_1(z) &= e^{-\frac{\lambda z}{r}} z^{\frac{\mu_c}{r}} (z_1-z)^{\frac{\eta}{r}\frac{z_1(z_2-1)}{z_2-z_1}} (z_2-z)^{-\frac{\eta}{r}\frac{z_2(z_1-1)}{z_2-z_1}}, \qquad z \le z_1,\\
\kappa_2(z) &= e^{-\frac{\lambda z}{r}} z^{\frac{\mu_c}{r}} (z-z_1)^{\frac{\eta}{r}\frac{z_1(z_2-1)}{z_2-z_1}} (z_2-z)^{-\frac{\eta}{r}\frac{z_2(z_1-1)}{z_2-z_1}}, \qquad z \ge z_1.
\end{aligned}
$$

According to [96], we obtain:

$$\pi_{c,0} = \frac{r\xi\kappa_2(1)S}{\mu_c(\xi+\eta)(SV-TU)}, \tag{6.46}$$

$$\pi_{w,0} = -\frac{r\kappa_2(1)T}{\mu_w(\xi+\eta)(SV-TU)}, \tag{6.47}$$

where S, T, U, V are defined as follows:

$$S = \int_0^{z_1} \frac{\kappa_1(x)}{\beta(x)}dx, \quad T = \int_0^{z_1} \frac{\kappa_1(x)}{x}dx, \quad U = \int_{z_1}^{1} \frac{\kappa_2(x)}{\beta(x)}dx \quad \text{and} \quad V = \int_{z_1}^{1} \frac{\kappa_2(x)}{x}d,$$

by the definitions of $\kappa_1(z)$, $\kappa_2(z)$ and $\beta(z)$, we have $T, U, V > 0$ and $S < 0$. Therefore, $\pi_{c,0}$ and $\pi_{w,0}$ are positive. One can show formally that the system is ergodic. Intuitively, we indicate that the system is always stable since, with any set of parameters $\lambda \geq 0$, $\mu_c \geq 0$, $\mu_w > 0$, $\xi > 0$, $\eta > 0$ and $r > 0$, the abandonment process, whose overall rate increases with the number of jobs, prevents explosion [96]. Alternatively, the system is stable if and only if $\pi_{c,0}$ and $\pi_{w,0}$ are positive, which always holds for the above set of parameters. According to [96], we obtain:

$$\mathbb{E}[N_c] = \frac{\lambda - \mu + \mu_c\pi_{c,0} + \mu_w\pi_{w,0}}{r}, \tag{6.48}$$

$$\mathbb{E}[N_w] = \frac{\eta(\lambda-\mu) + r(\lambda-\mu_w)\pi_w + \eta\mu_c\pi_{c,0} + \mu_w(\eta+r)\pi_{w,0}}{\xi r}. \tag{6.49}$$

As shown in Fig. 6.17, the expected number of jobs served per unit of time in the slow phase and fast phase are $\mu_c(\pi_c - \pi_{c,0})$ and $\mu_w(\pi_w - \pi_{w,0})$, respectively [141]. Therefore, the rate of abandonment due to impatience in the slow phase, λ_a, is given by

$$\begin{aligned} \lambda_a &= \lambda - \mu_c(\pi_c - \pi_{c,0}) - \mu_w(\pi_w - \pi_{w,0}) \\ &= \lambda - \mu + \mu_c\pi_{c,0} + \mu_w\pi_{w,0} \\ &= r \cdot \mathbb{E}[N_c]. \end{aligned} \tag{6.50}$$

The rate of jobs executed locally on the mobile device λ_m must be equal to λ_a, i.e. $\lambda_m = \lambda_a$. The probability that an arbitrary job arriving to the offload queue will renege and join to the local queue, i.e. it will be executed locally and will never offload, is defined as:

$$p_r = \frac{\lambda_a}{\lambda} = \frac{\lambda - \mu + \mu_c\pi_{c,0} + \mu_w\pi_{w,0}}{\lambda}. \tag{6.51}$$

We recollect that the busy fraction of the service station is $\rho = 1 - (\pi_{c,0} + \pi_{w,0})$.

Extreme Case

Assume $r \to 0$, the partial offloading model in Fig. 6.16 reduces to the non-delayed offloading model, which is depicted in Fig. 6.18, since the reneging rate is zero, there will be no local queue in this model, and it is analyzed in comparison with the delayed offloading models.

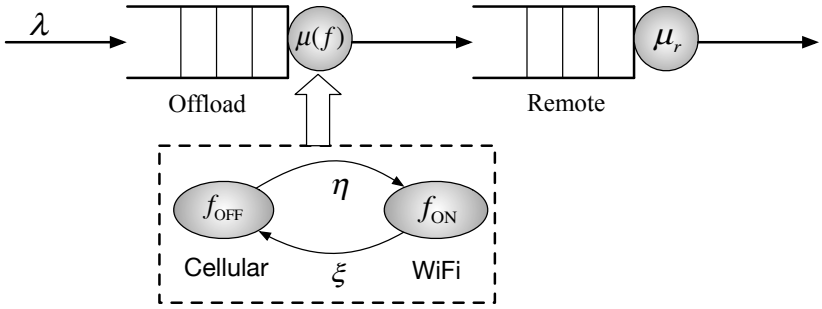


Figure 6.18: The non-delayed offloading model

After solving the equations of (6.42) when setting $r = 0$, we have [142]:

$$g(z)G_c(z) = \pi_{w,0}\xi\mu_w z + \pi_{c,0}\mu_c\Big[\xi z + \lambda z(1-z) - \mu_w(1-z)\Big],$$

where $g(z) = \lambda^2 z^3 - \lambda(\eta + \xi + \lambda + \mu_c + \mu_w)z^2 + (\eta\mu_w + \xi\mu_c + \mu_c\mu_w + \lambda(\mu_c + \mu_w))z - \mu_c\mu_w$, and it is proven that $g(z)$ has only one root z_0 in the interval $(0, 1)$.

After some algebraic manipulations, we obtain:

$$\pi_{c,0} = \frac{\xi(\mu - \lambda)z_0}{\mu_c(1 - z_0)(\mu_w - \lambda z_0)}, \tag{6.52}$$

$$\pi_{w,0} = \frac{\eta(\mu - \lambda)z_0}{\mu_w(1 - z_0)(\mu_c - \lambda z_0)}. \tag{6.53}$$

Once the values of $\pi_{c,0}$ and $\pi_{w,0}$ have been established, the probability generating functions can be calculated as:

$$G_c(z) = \frac{\xi(\mu - \lambda)z + \pi_{c,0}\mu_c(1 - z)(\lambda z - \mu_w)}{g(z)}, \tag{6.54}$$

$$G_w(z) = \frac{\eta(\mu - \lambda)z + \pi_{w,0}\mu_w(1 - z)(\lambda z - \mu_w)}{g(z)}. \tag{6.55}$$

By using $\mathbb{E}[N_i] = \sum_{n=0}^{\infty} n\pi_{i,n} = \mathrm{d}G_i(z)/\mathrm{d}z|_{z=1}$, we get the average number of jobs in the system [142]:

$$\begin{aligned}\mathbb{E}[N] &= \mathbb{E}[N_c] + \mathbb{E}[N_w] \\ &= \frac{\lambda}{\mu-\lambda} + \frac{\mu_c(\mu_w-\lambda)\pi_{c,0} + \mu_w(\mu_c-\lambda)\pi_{w,0} - (\mu_c-\lambda)(\mu_w-\lambda)}{(\xi+\eta)(\mu-\lambda)}.\end{aligned} \tag{6.56}$$

Mean Response Time

The total cost for offloading a job is composed of the cost for sending the job to the cloud and idly waiting for the cloud to complete the job. By Little's Law, $\mathbb{E}[N] = \lambda\mathbb{E}[T]$, the mean response time of the partial offloading model can be calculated as:

$$\mathbb{E}[T] = \mathbb{E}\Big[\mathbb{E}[T_i]\Big] = \sum_{i\in\{c,w,m,r\}} \frac{\lambda_i}{\lambda}\mathbb{E}[T_i] = \frac{1}{\lambda}\sum_{i\in\{c,w,m,r\}} \mathbb{E}[N_i], \tag{6.57}$$

where $\mathbb{E}[N_c]$ and $\mathbb{E}[N_w]$ are the average number of jobs in the cellular network and WiFi network as obtained in (6.48) and (6.49), respectively.

For the local processing, since the arrival rate equals to the reneging rate, we have $\lambda_m = r \cdot \mathbb{E}[N_c]$. For an ordinary $M/M/1$-FCFS queue, the average number of jobs on the mobile device is $\mathbb{E}[N_m] = \rho_m/(1-\rho_m)$, where $\rho_m = \lambda_m/\mu_m$ is the utilization.

Since there is no waiting time before entering into remote service in the cloud, for an $M/M/\infty$ queue, the average number of jobs on the cloud server can be calculated as: $\mathbb{E}[N_r] = \lambda_r/\mu_r$, where $\lambda_r = \lambda - \lambda_m$.

Mean Energy Consumption

A key assumption in our work is that each service operates at a constant power p_i, ($i \in \{c, w, m\}$) whenever it is busy, i.e. the mobile device consumes energy only when there are jobs in the system. Since $\mathbb{E}[P] = \lambda\mathbb{E}[\mathcal{E}]$ is the mean power consumption, we can calculate the mean energy consumption for the partial offloading model as:

$$\mathbb{E}[\mathcal{E}] = \mathbb{E}\Big[\mathbb{E}[\mathcal{E}_i]\Big] = \sum_{i\in\{c,w,m\}} \frac{\lambda_i}{\lambda}\mathbb{E}[\mathcal{E}_i] = \frac{1}{\lambda}\sum_{i\in\{c,w,m\}} \mathbb{E}[P_i]. \tag{6.58}$$

Since the application jobs are remotely executed on the cloud server rather than on the mobile device, we do not need to calculate this energy consumption. Since the utilization of the queue is

the probability that the server is busy, we have $\Pr\{N_i > 0\} = \rho_i$ [116], i.e. the energy cost is only incurred during the fraction of the time the server is busy. The corresponding average power consumption can be calculated as: $\mathbb{E}[P_i] = p_i \cdot \Pr\{N_i > 0\} = p_i \cdot \rho_i$.

The energy consumed due to local execution depends on the processing speed of the mobile device and its specific characteristics. Since the service on the mobile device is always available, we have:

$$\mathbb{E}[P_m] = p_m \cdot \Pr\{N_m > 0\} = p_m \cdot \rho_m. \tag{6.59}$$

Since the energy consumed due to offloading depends on the transmission rate, we have:

$$\mathbb{E}[P_c] = p_c \cdot \Pr\{N_c > 0\} = p_c \cdot \rho_c, \tag{6.60}$$

$$\mathbb{E}[P_w] = p_w \cdot \Pr\{N_w > 0\} = p_w \cdot \rho_w. \tag{6.61}$$

According to Fig. 6.17, the utilization of the cellular and WiFi network are the probability that the corresponding network is busy, we have: $\rho_c = \pi_c - \pi_{c,0}$ and $\rho_w = \pi_w - \pi_{w,0}$.

Further, by substituting (6.57) and (6.58), into (2.4), we can formulate the explicit expressions of the ERWP metric for the delayed offloading model.

6.3.2 Full Offloading Model

For the full offloading model, in which all jobs are offloaded, there are two coupled queues used for offloading at the side of a mobile device, called WiFi queue and cellular queue, and both queues are served by a FCFS discipline.

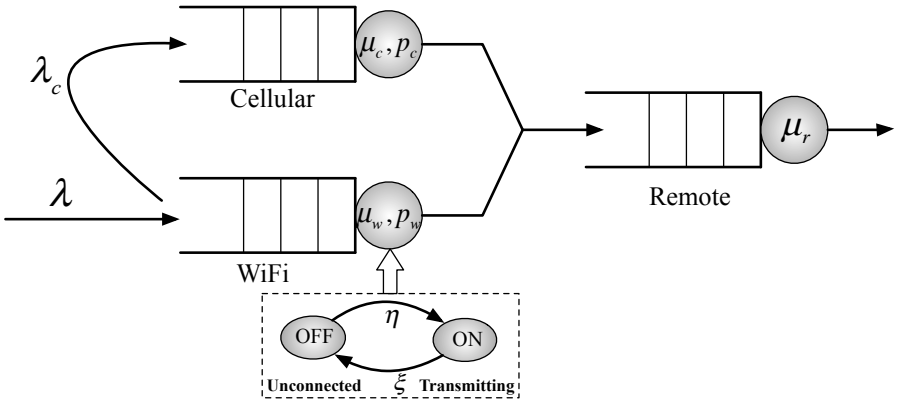


Figure 6.19: The full offloading model

Figure 6.19 depicts a delayed offloading model based on the WiFi network's availability model.

All jobs arriving to the system are by default sent to the WiFi interface for offloading. When a job is offloaded to the cloud via a WLAN network, there is queueing due to the transmission speed of the WLAN link. We model the intermittent availability of hotspots as a queue with occasional server break-down. The server availability is governed by an IPP with exponentially distributed ON-OFF periods. Specifically, the server is either in ON-state processing the existing jobs, or in OFF-state during which no job receives service. We assume the jobs will abandon the queue during periods without WiFi connectivity.

We assign a reneging deadline for each job (drawn from an exponential distribution). Jobs are served in the FCFS order depending on their remaining deadlines (either while queued or while at the head of the queue, but waiting for WiFi). A job can be served only via WiFi before its deadline. As the queueing system is continuous, it handles transmission at the bit level so that assigning a deadline to a job is equivalent to assigning the same deadline to each bit of the job. When the channel is in the OFF-state, jobs become impatient. That is, each job, upon arrival, activates an individual timer, exponentially distributed with an reneging rate r. If the network does not recover from the OFF-state to the ON-state before the deadline expires, the job abandons the WiFi queue, instead, to be offloaded via cellular network.

If the job in the WiFi queue is completely transmitted through WiFi networks before the assigned deadline has expired, we say that the job is successfully offloaded. If offloading fails, the job leaves the WiFi queue and joins the cellular queue for immediate transmission through a 3G connection. We call such an event a reneging event. When the job is offloaded to the cloud via a cellular network, there is queueing due to the transmission speed of the cellular link. Costs arise in terms of transmission delays (queueing and actual transmission time) and transmission energy consumption. The service is always available since the cellular connection is always on.

Similarly, the cloud queue is a pure delay station at which jobs spend an exponentially distributed amount of time with mean equal to $1/\mu_r$ time units.

Queueing Analysis

The WiFi Queue refers to offloading jobs from the mobile device to the cloud via a WLAN network, which is modelled as an $M/M/1$-FCFS queue with intermittently available service. When a server recovers, it continues to serve the customer whose service has been interrupted, i.e. the work already completed is not lost (cf. data transfers with resume).

We assume that the service station fails from time to time and resumes its operation after a random time. The Markov chain for the delayed offloading model is depicted in Fig. 6.20, which is equivalent to assuming that $\mu_c = 0$, $\pi_{\text{ON}} = \pi_w$ and $\pi_{\text{OFF}} = \pi_c$ in Fig. 6.17.

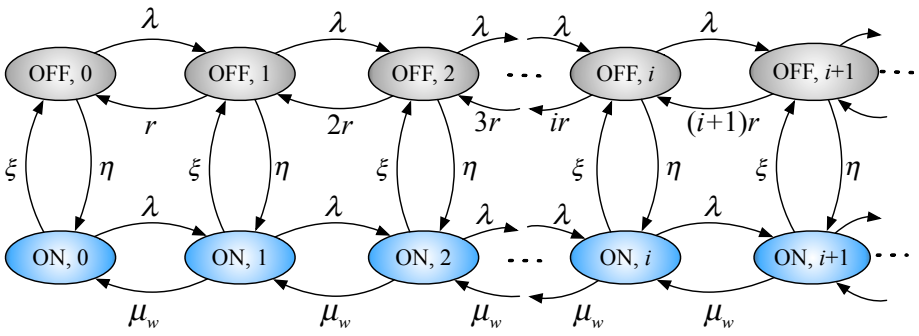


Figure 6.20: The 2D Markov chain for the WiFi queue

Writing the balance equations for this chain gives:

$$(\lambda+\eta)\pi_{\text{OFF},0} = \xi\pi_{\text{ON},0} + r\pi_{\text{OFF},1}, \tag{6.62a}$$

$$(\lambda+\eta+ir)\pi_{\text{OFF},i} = \lambda\pi_{\text{OFF},i-1} + (i+1)r\pi_{\text{OFF},i+1} + \xi\pi_{\text{ON},i} \quad (i>0), \tag{6.62b}$$

$$(\lambda+\xi)\pi_{\text{ON},0} = \eta\pi_{\text{OFF},0} + \mu_w\pi_{\text{ON},1}, \tag{6.62c}$$

$$(\lambda+\xi+\mu_w)\pi_{\text{ON},i} = \lambda\pi_{\text{ON},i-1} + \mu_w\pi_{\text{ON},i+1} + \eta\pi_{\text{OFF},i} \quad (i>0). \tag{6.62d}$$

According to [77], we obtain:

$$\pi_{\text{OFF},0} = -\frac{\xi\kappa_2(1)S}{(\xi+\eta)\kappa_1(0)U}, \tag{6.63}$$

$$\pi_{\text{ON},0} = \frac{r\kappa_2(1)}{\mu_w(\xi+\eta)U}. \tag{6.64}$$

We have $\mu = \pi_c \cdot \mu_c + \pi_w \cdot \mu_w = \pi_{\text{ON}} \cdot \mu_w$. After substituting the above values into (6.48) and (6.49), we derive the mean number of jobs in the WiFi Queue as:

$$\mathbb{E}[N_{\text{OFF}}] = \frac{\lambda - \mu_w(\pi_{\text{ON}} - \pi_{\text{ON},0})}{r}, \tag{6.65}$$

$$\mathbb{E}[N_{\text{ON}}] = \frac{\eta\lambda - \mu_w(\eta+r)(\pi_{\text{ON}} - \pi_{\text{ON},0}) + \lambda r\pi_{\text{ON}}}{\xi r}. \tag{6.66}$$

Therefore, the average number of jobs in the WiFi queue is formulated as:

$$\mathbb{E}[N_w] = \mathbb{E}[N_{\text{OFF}}] + \mathbb{E}[N_{\text{ON}}]. \tag{6.67}$$

In Fig. 6.20 the expected number of jobs served per unit of time in the WiFi queue is $\mu_{\text{ON}}(\pi_{\text{ON}} -$

$\pi_{\text{ON},0}$). Therefore, the rate of abandonment due to impatience in the OFF periods, λ_a, is given by:

$$\lambda_a = \lambda - \mu_{\text{ON}}(\pi_{\text{ON}} - \pi_{\text{ON},0}) = r \cdot \mathbb{E}[N_{\text{OFF}}]. \tag{6.68}$$

The rate of jobs sent back to the cellular network λ_c must be equal to the abandonment rate, i.e. $\lambda_c = \lambda_a$. The probability that an arbitrary job arriving to the WiFi queue will renege, i.e. it will be offloaded over a cellular queue, is defined as:

$$p_r = \frac{\lambda_a}{\lambda} = \frac{\lambda - \mu_w(\pi_{\text{ON}} - \pi_{\text{ON},0})}{\lambda}. \tag{6.69}$$

Mean Response Time

By Little's Law, $\mathbb{E}[N] = \lambda\mathbb{E}[T]$, the mean response time can be calculated as:

$$\mathbb{E}[T] = \mathbb{E}\Big[\mathbb{E}[T_i]\Big] = \sum_{i\in\{c,w,r\}} \frac{\lambda_i}{\lambda}\mathbb{E}[T_i] = \frac{1}{\lambda}\sum_{i\in\{c,w,r\}} \mathbb{E}[N_i], \tag{6.70}$$

where $\mathbb{E}[N_w]$ is the average number of jobs in the WiFi queue as obtained in (6.67).

The cellular queue refers to offloading jobs from the mobile device to the cloud via a cellular network, which is modelled as an $M/M/1$-FCFS queue. The average number of jobs is $\mathbb{E}[N_c] = \rho_c/(1-\rho_c)$, where $\rho_c = \lambda_c/\mu_c$ is the probability that the cellular queue is busy.

Since all the jobs are offloaded to the remote cloud server, for an $M/M/\infty$ queue, the average number of jobs in the cloud server is $\mathbb{E}[N_r] = \lambda/\mu_r$.

Mean Energy Consumption

The mean energy consumption can be calculated as:

$$\mathbb{E}[\mathcal{E}] = \mathbb{E}\Big[\mathbb{E}[\mathcal{E}_i|i]\Big] = \sum_{i\in\{w,c\}} \frac{1}{\lambda}\mathbb{E}[P_i] = \frac{1}{\lambda}\sum_{i\in\{w,c\}} p_i \cdot \Pr\{N_i > 0\} = \frac{1}{\lambda}\sum_{i\in\{w,c\}} p_i \cdot \rho_i, \tag{6.71}$$

where $\rho_w = \pi_{\text{ON}} - \pi_{\text{ON},0}$ is the fraction of time that WiFi is available to process jobs, and as the recovery rate $\eta \to \infty$, the availability ratio of WiFi $\pi_{\text{ON}} = \frac{\eta}{\xi+\eta}$ tends to be 1.

Further, by substituting (6.70) and (6.71) into (2.4), we can formulate the optimization of the ERWP metric for the offloading assignment as:

$$r^* = \arg\min_r \ ERWP, \tag{6.72}$$

we seek to find the reneging rate r^* such that $ERWP$ is minimized.

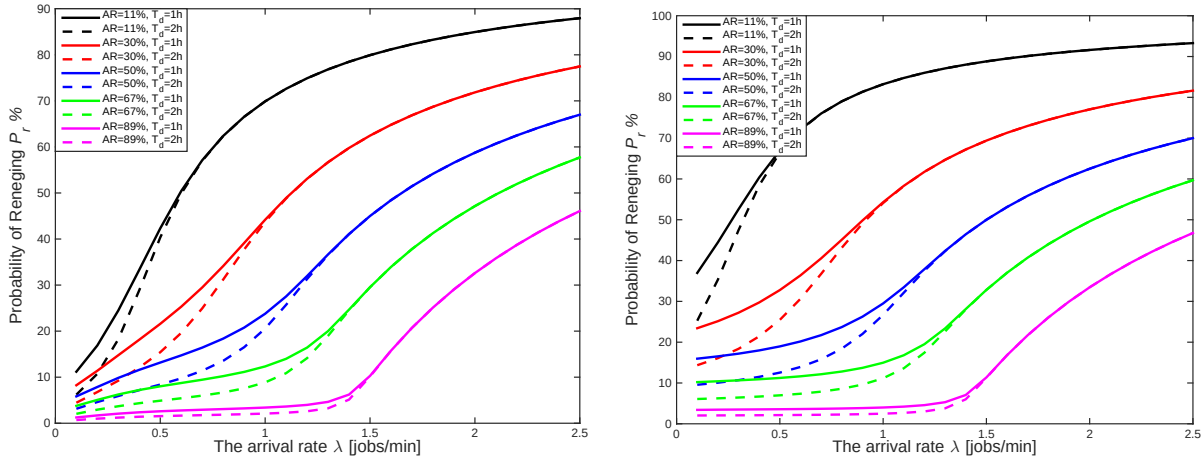
6.3.3 Analytical Evaluation

Using measurements from real traces in [64], the average data rates of the cellular and WiFi networks are set as $s_c = 200$ Kbps and $s_w = 2$ Mbps, respectively. The average duration of WiFi availability period is 52 min ($\xi = 1/52$ min^{-1}), while the average duration with only cellular network coverage is 25.4 min ($\eta = 1/25.4$ min^{-1}). The availability ratio is thus 67%. The mean job size is assumed to be 10 MB. According to the power models developed in [10], we set the power coefficients $p_c = 2.5$ W, $p_w = 0.7$ W and $p_m = 2$ W, respectively. Besides, suppose that the total job arrival rate is $\lambda = 0.5$ packet/min, the mobile service rate $\mu_m = 0.2$ and the cloud service rate $\mu_r = 1$.

We first analyze the respective probabilities of reneging for the two delayed offloading models. An availability ratio $AR = 11\%$ has been reported in [9]. As shown in Fig. 6.21, as AR of the WiFi network increases, the percentage of jobs abandon the offload queue (for the partial offloading model, seen as in Fig. 6.21(a)) or the WiFi queue (for the full offloading model, refer to Fig. 6.21(b)) drops. However, the full offloading model has the higher reneging probability than the partial offloading model under the same deadline. That is because the partial offloading model can use the cellular network to transmit data, and thus the number of jobs waiting in the offload queue is reduced. On the other hand, as the reneging deadline increases from 60 min to 120 min, the jobs have a higher chance to be offloaded via the WiFi network, and therefore the reneging probability decreases at the lower level of arrival rates. However, for high arrival rates, the reneging probability stays the same under different deadlines.

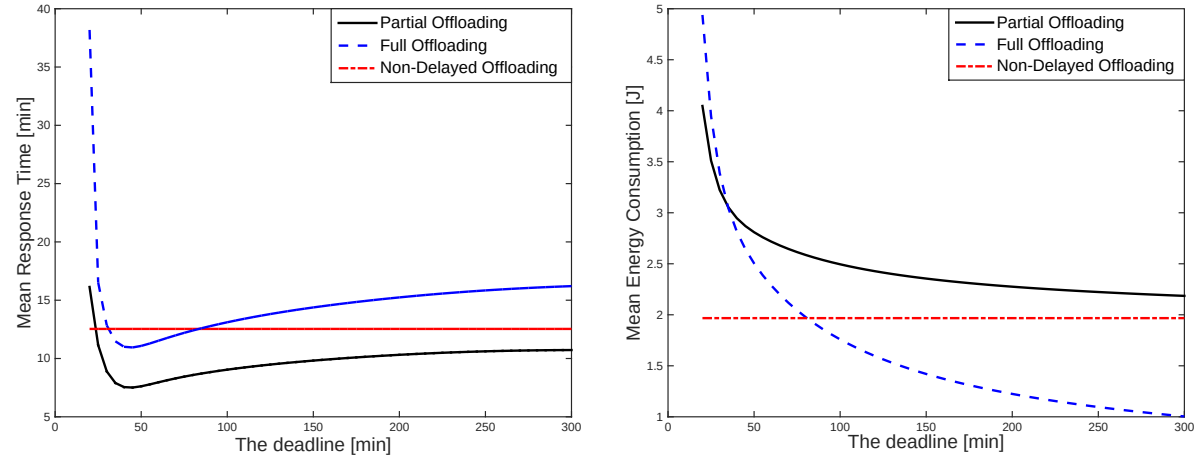
The mean response time includes the queueing and service time. From Fig. 6.22(a), it can be seen that the partial offloading model has the lowest average response time, since it takes full use of the slow phase of the cellular network when the WiFi is in the unavailable period. For the lower deadlines ($T_d < 40$ min), the mean response time decreases as the deadline rises, since jobs with higher deadlines a higher chance to transmit with the fast WiFi network, leading to smaller response time. However, the mean response time increases for longer deadlines, since jobs with lower deadlines leave the queue earlier, leading to lower queueing delays. In Fig. 6.22(b) when the reneging deadline is small, the non-delayed offloading model achieves the lowest mean energy consumption, but as the deadline increases, the full offloading model is much better. This is due to the fact that the WiFi network is much more energy efficient than a cellular network like 3G.

We then fix the reneging deadline to 120 min under different arrival rates. In Fig. 6.23(a) the mean response time increases when increasing the arrival rate due to queueing effects. The partial offloading model performs much better than the other two models since it fully uses the unavailable



(a) Partial offloading model (b) Full offloading model

Figure 6.21: The reneging probabilities for the delayed offloading models

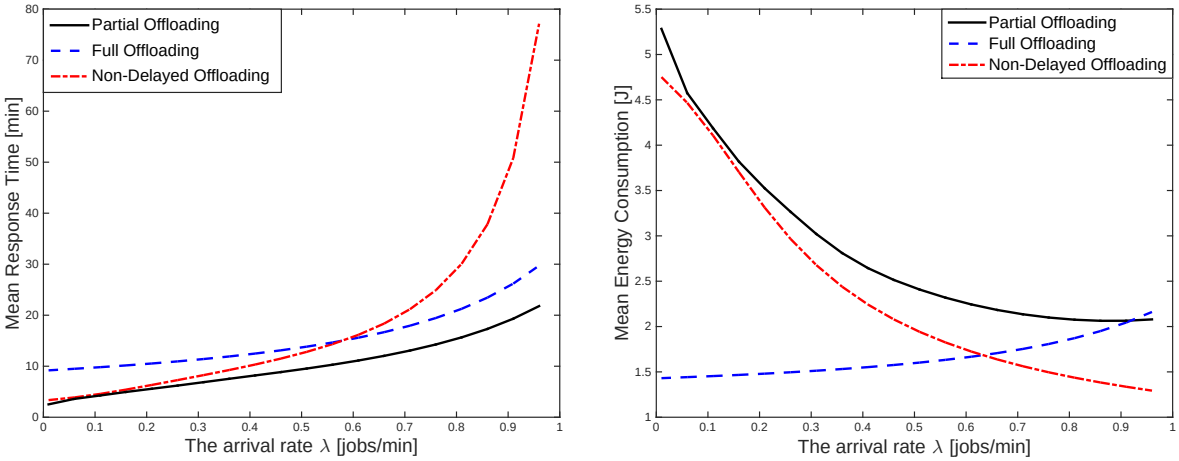


(a) Mean Response Time (b) Mean Energy Consumption

Figure 6.22: Comparison for the offloading models under different deadlines

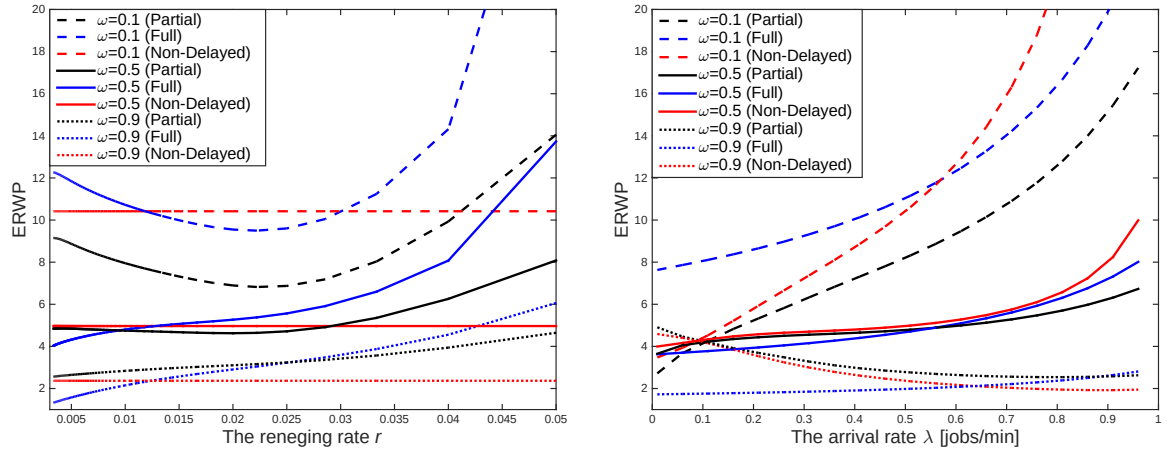
periods of WiFi by offloading jobs with a cellular network, which in turn brings tremendous energy consumption as shown in Fig. 6.23(b).

We then use the ERWP metric to compare three offloading models. In Fig. 6.24(a) when ω is small, the partial offloading model can achieve the smallest ERWP value by optimizing the reneging rate r, which indicates that when considering response time more important (for delay-sensitive applications), it is better to use the partial offloading model. Otherwise, when considering energy consumption more important than response time (for delay-tolerant applications), the full offloading



(a) Mean Response Time (b) Mean Energy Consumption

Figure 6.23: Comparison for the offloading models under different arrival rates



(a) Reneging rate (b) Arrival rate

Figure 6.24: Comparison of ERWP for the offloading models under different reneging and arrival rates

model is much better, which translates the reduced transmission time from the fast WiFi network into battery power saving for the mobile device. In Fig. 6.24(b) when ω is small, as the arrival rate of the offloadable jobs λ increases, all the three offloading models perform worse. However, the non-delayed offloading model is more sensitive to the job arrival rates. The partial offloading model can always achieve the smallest ERWP value, which means that when considering response time more important, it is better to use the partial offloading model. Otherwise, when considering

energy consumption more important than response time, the full offloading model is much better for smaller λ. While at higher job arrival rate, the non-delayed offloading model is preferred.

6.4 Summary

We have presented a fundamental approach for designing an online algorithm for the energy-delay tradeoff in “delayed" offloading through the Lyapunov optimization framework. Three types of transmission schedulers were proposed and compared for making offloading decisions, which consider several factors: data backlog, channel quality and energy consumption of the wireless interface. They will decide to transmit data when the connectivity is good enough or when the queues in the mobile device are congested. A significant amount of energy can be saved without sacrificing on the transmission delay too much.

The jobs will abandon the queue very often especially when the availability ratio of the WiFi network is very small. We can optimally choose the reneging deadline to achieve a different energy-performance tradeoff by optimizing the ERWP metric. In general one can say that for delay-sensitive applications, the partial offloading model is preferred when setting an intermediate deadline; while for delay-tolerant applications, the full model shows very good results and outperforms the other offloading models when setting a large deadline.

Chapter 7

Concluding Remarks

7.1 Conclusions

In this thesis, the main goal has been to make offloading decisions based on time and energy saving, which can be divided into four sub-targets, namely, *what*, *where*, *how* and *when* to offload. We discuss contributions in those four aspects as follows:

- *What to Offload*: through a proposed partitioning algorithm designed for arbitrary topology of applications, we are able to determine which portions of the application to run on a mobile device and what to execute on a cloud server according to different cost models. The algorithm provides a stable low time complexity method and can significantly reduce execution time and energy consumption by optimally distributing tasks between the mobile device and the cloud. At the same time, it can well adapted to dynamic changes of context.
- *Where to Offload*: through static (combing AHP and fuzzy TOPSIS methods) and dynamic (Lyapunov- and LARAC-based methods) offloading decisions, we are able to choose the most appropriate offloading target (e.g. local, cloudlet and cloud) to which the code has to be off-loaded. The former takes multiple criteria into consideration, while the latter has low complexity and can significantly reduce the energy consumed while satisfying the response time constraint imposed by the mobile application.
- *How to Offload*: through queueing analysis for offloading operations with multiple network interfaces in heterogeneous wireless environments, we are able to capture the tradeoff between the energy consumption and the response time and to determine how to leverage the complementary strength of WiFi and cellular networks by choosing heterogeneous wireless interfaces for offloading.
- *When to Offload*: through dynamic transmission scheduling and link selection based on Lya-

punov optimization for data offloading between the mobile device and the cloud, we are able to extend the device's battery life for transferring large volumes of data. If a delay-tolerant job is deferred up to a given deadline, or until a fast and energy-efficient network becomes available, the transmission time will be reduced, which could lead to saving energy. However, if the reduced service time can not cover the extra waiting time, this policy may not be suitable for delay-sensitive applications.

We try to figure out how effective and efficient mobile offloading systems are and what factors influence their performance, by determining the optimal partitioning scheme for splitting a specific application into local and remote parts (*what to offload*), the type of surrogate in which to be offloaded (*where to offload*), offloading plans that enable the device to schedule mobile offloading operations (*how to offload*) and the right time to offload (*when to offload*) under different conditions of the device, such as available bandwidth, data to transmit, and tasks to execute.

7.2 Suggestions

Several directions have been suggested for future work in the previous chapters. The main ideas of them are summarized as follows:

- Parameters for decision making such as bandwidth, security, response time and failure rate are sometimes hard to measure or acquire in a timely manner in practical systems. Therefore, the way how to estimate and measure these parameters need to be further investigated.
- Assumptions that the data rate and power consumption keep constant in all the regions within network coverage is somehow unrealistic. Therefore, it is worth considering scenarios where they might be different at each connected access point.
- Derivation based on Poisson processes and exponentially distributed of the job sizes should be extended to other arrival processes with arbitrary distributions of the job sizes.
- Validation based on real workloads and more realistic application examples will be provided in the future to gain insights about efficiency of the proposed algorithms for offloading decision making.
- Integrating the proposed offloading decision algorithms in an actual software deployment framework to automatically distribute software components on a cloud infrastructure.

Bibliography

[1] E. H. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] E. Abebe and C. Ryan. Adaptive application offloading using distributed abstract class graphs in mobile environments. *Journal of Systems and Software*, 85(12):2755–2769, 2012.

[3] O. Alexandru-Corneliu. *Extending the capabilities of mobile devices through cloud offloading*. PhD thesis, University Politehnica of Bucharest, Romania, 2013.

[4] K. Ali and O. Lhoták. Application-only call graph construction. In *ECOOP 2012–Object-Oriented Programming*, pages 688–712. Springer, 2012.

[5] Amazon. EC2. http://aws.amazon.com/ec2/.

[6] M. Amoretti, A. Grazioli, F. Zanichelli, V. Senni, and F. Tiezzi. Towards a formal approach to mobile cloud computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 743–750. IEEE, 2014.

[7] L. L. Andrew, M. Lin, and A. Wierman. Optimality, fairness, and robustness in speed scaling designs. *ACM SIGMETRICS Performance Evaluation Review*, 38(1):37–48, 2010.

[8] Apple. iCloud. http://www.apple.com/icloud/.

[9] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3G using WiFi. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 209–222. ACM, 2010.

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

[11] S. Balsamo, G.-L. dei Rossi, and A. Marin. Queueing networks and conditional product-forms. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, pages 204–213. ICST, 2013.

[12] R. Beraldi, K. Massri, M. Abderrahmen, and H. Alnuweiri. Towards automating mobile cloud computing offloading decisions: An experimental approach. In *ICSNC 2013: The Eighth International Conference on Systems and Networks Communications*, 2013.

[13] D. P. Bertsekas, R. G. Gallager, and P. Humblet. *Data networks*, volume 2. Prentice-hall Englewood Cliffs, NJ, 1987.

[14] W. Binder and J. Hulaas. Using bytecode instruction counting as portable CPU consumption metric. *Electronic Notes in Theoretical Computer Science*, 153(2):57–77, 2006.

[15] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

[16] V. Cardellini, V. D. N. Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli. A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*, pages 1–29, 2013.

[17] X. Chen. Decentralized computation offloading game for mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 26(4):974–983, 2015.

[18] X. Chen and M. R. Lyu. Performance and effectiveness analysis of checkpointing in mobile environments. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 131–140. IEEE, 2003.

[19] M. H. Cheung and J. Huang. Dawn: Delay-aware Wi-Fi offloading and network selection. *IEEE Journal on Selected Areas in Communications*, 33(6):1214–1223, 2015.

[20] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[21] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010.

[22] C. S. M. I. Consortium. Service measurement index version 1.0, 2011.

[23] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[24] M. Dağdeviren, S. Yavuz, and N. Kılınç. Weapon selection using the AHP and TOPSIS methods under fuzzy environment. *Expert Systems with Applications*, 36(4):8143–8151, 2009.

[25] S. Deng, L. Huang, J. Taheri, and A. Zomaya. Computation offloading for service workflow in mobile cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1–1, 2014.

[26] P. Di Lorenzo, S. Barbarossa, and S. Sardellitti. Joint optimization of radio resources and code partitioning in mobile cloud computing. *arXiv preprint arXiv:1307.3835*, 2013.

[27] J.-F. Ding and G.-S. Liang. Using fuzzy MCDM to select partners of strategic alliances for liner shipping. *Information Sciences*, 173(1):197–225, 2005.

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

[29] Y. Fang. Modeling and performance analysis for wireless mobile networks: a new analytical approach. *Networking, IEEE/ACM Transactions on*, 13(5):989–1002, 2005.

[30] K. Fekete, K. Csorba, B. Forstner, T. Vajk, M. Feher, and I. Albert. Analyzing computation offloading energy-efficiency measurements. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 301–305. IEEE, 2013.

[31] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya. Mobile code offloading: from concept to practice and beyond. *Communications Magazine, IEEE*, 53(3):80–88, 2015.

[32] H. Flores and S. N. Srirama. Mobile cloud middleware. *Journal of Systems and Software*, 92:82–94, 2014.

[33] H. Flores, S. N. Srirama, and C. Paniagua. Towards mobile cloud applications: Offloading resource-intensive tasks to hybrid clouds. *International Journal of Pervasive Computing and Communications*, 8(4):344–367, 2012.

[34] J.-M. Fourneau, B. Plateau, and W. Stewart. Product form for stochastic automata networks. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*. ICST, 2007.

[35] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13–, 2009.

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

[37] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[38] A. Gember, C. Dragga, and A. Akella. Ecos: leveraging software-defined networks to support mobile application offloading. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 199–210. ACM, 2012.

[39] L. Georgiadis, M. J. Neely, and L. Tassiulas. *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

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

[41] Google. Google app engine. https://appengine.google.com/.

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

[43] M. Griera Jorba. Improving the reliability of an offloading engine for android mobile devices and testing its performance with interactive applications. Master's thesis, Freie Universität Berlin, 2013.

[44] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading for pervasive computing. *Pervasive Computing, IEEE*, 3(3):66–73, 2004.

[45] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel computing*, 26(12):1519–1534, 2000.

[46] K. Henricksen, J. Indulska, and A. Rakotonirainy. Infrastructure for pervasive computing: Challenges. In *GI Jahrestagung (1)*, pages 214–222, 2001.

[47] J.-A. Hong, S. Seo, N. Kim, and B.-D. Lee. A study of secure data transmissions in mobile cloud computing from the energy consumption side. In *Information Networking (ICOIN), 2013 International Conference on*, pages 250–255. IEEE, 2013.

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

[49] E. Hyytiä, S. Aalto, A. Penttinen, and J. Virtamo. On the value function of the M/G/1 FCFS and LCFS queues. *Journal of Applied Probability Evaluation*, 49(4):1052–1071, 2012.

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

[51] E. Hyytiä, J. Virtamo, S. Aalto, and A. Penttinen. M/M/1-PS queue and size-aware task assignment. *Performance Evaluation*, 68(11):1136–1148, 2011.

[52] M. Jia, J. Cao, and L. Yang. Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 352–357. IEEE, 2014.

[53] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó. Lagrange relaxation based method for the

QoS routing problem. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 859–868. IEEE, 2001.

[54] B. Y.-H. Kao and B. Krishnamachari. Optimizing mobile computational offloading with delay constraints. In *Proc. of Global Communication Conference (Globecom 14)*, pages 8–12, 2014.

[55] Y. Kim, K. Lee, and N. B. Shroff. An analytical framework to characterize the efficiency and delay in a mobile data offloading system. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 267–276. ACM, 2014.

[56] A. Klein, C. Mannweiler, J. Schneider, and H. D. Schotten. Access schemes for mobile cloud computing. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on*, pages 387–392. IEEE, 2010.

[57] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Unleashing the power of mobile cloud computing using thinkair. *arXiv preprint arXiv:1105.3232*, 2011.

[58] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[59] D. Kovachev and R. Klamma. Framework for computation offloading in mobile cloud computing. *International Journal of Interactive Multimedia & Artificial Intelligence*, 1(7):6–15, 2012.

[60] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[61] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[62] K. Kumar, Y. Nimmagadda, and Y.-H. Lu. Ranking servers based on energy savings for computation offloading. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 267–272. ACM, 2009.

[63] Y.-W. Kwon and E. Tilevich. Energy-efficient and fault-tolerant distributed mobile execution. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 586–595. IEEE, 2012.

[64] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. Mobile data offloading: How much can WiFi deliver? *Networking, IEEE/ACM Transactions on*, 21(2):536–550, 2013.

[65] K. Lee and I. Shin. User mobility-aware decision making for mobile computation offloading. In *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*, pages 116–119. IEEE, 2013.

[66] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng. Challenges on wireless heterogeneous networks for mobile cloud computing. *Wireless Communications, IEEE*, 20(3):34–44, 2013.

[67] O. Lhoták and L. Hendren. Scaling java points-to analysis using spark. In *Compiler Construction*, pages 153–169. Springer, 2003.

[68] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246. ACM, 2001.

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

[70] Y.-D. Lin, E.-H. Chu, Y.-C. Lai, and T.-J. Huang. Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds. *Systems Journal, IEEE*, 9(2):393–405, 2013.

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

[72] F. Liu, P. Shu, and J. C. Lui. Appatp: An energy conserving adaptive mobile-cloud transmission protocol. *IEEE Transactions on Computers*, 64(11):3051–3063, 2015.

[73] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99–117, 2015.

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

[75] X. Lu. Energy-aware performance analysis of queueing systems. Master's thesis, Aalto university, 2013.

[76] J. Martínez Ripoll. Improving the performance and usability of an offloading engine for android mobile devices with application to a chess game. Master's thesis, Technische Universität Berlin, 2013.

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

[78] F. Mehmeti and T. Spyropoulos. Is it worth to be patient? Analysis and optimization of delayed mobile data offloading. In *INFOCOM, 2014 Proceedings IEEE*, pages 2364–2372.

IEEE, 2014.

[79] F. Mehmeti and T. Spyropoulos. Performance analysis of mobile data offloading in heterogeneous networks. 2014.

[80] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. J. Giuli, and X. Gu. Towards a distributed platform for resource-constrained devices. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 43–51. IEEE, 2002.

[81] Microsoft. Microsoft windows azure. https://windows.azure.com/.

[82] A. P. Miettinen and J. K. Nurminen. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 4–4. USENIX Association, 2010.

[83] M. J. Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.

[84] M. P. S. Nir. *Scalable resource augmentation for mobile devices*. PhD thesis, Carleton University Ottawa, 2014.

[85] J. Niu, W. Song, and M. Atiquzzaman. Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications. *Journal of Network and Computer Applications*, 37:334–347, 2014.

[86] R. Niu, W. Song, and Y. Liu. An energy-efficient multisite offloading algorithm for mobile devices. *International Journal of Distributed Sensor Networks*, 2013.

[87] M. Nkosi and F. Mekuria. Cloud computing for enhanced mobile health applications. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 629–633. IEEE, 2010.

[88] D. L. Olson. Comparison of weights in topsis models. *Mathematical and Computer Modelling*, 40(7):721–727, 2004.

[89] A.-C. Olteanu and N. Tapus. Tools for empirical and operational analysis of mobile offloading in loop-based applications. *Informatica Economica*, 17(4):5–17, 2013.

[90] S. Ou, Y. Wu, K. Yang, and B. Zhou. Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 1856–1860. IEEE, 2008.

[91] S. Ou, K. Yang, and A. Liotta. An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 116–125. IEEE Computer Society, 2006.

[92] S. Ou, K. Yang, A. Liotta, and L. Hu. Performance analysis of offloading systems in mobile wireless environments. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1821–1826. IEEE, 2007.

[93] V. Pandey, S. Singh, and S. Tapaswi. Energy and time efficient algorithm for cloud offloading using dynamic profiling. *Wireless Personal Communications*, pages 1–15, 2014.

[94] A. Papoulis and S. U. Pillai. *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 2002.

[95] A. Penttinen, E. Hyytiä, and S. Aalto. Energy-aware dispatching in parallel queues with on-off energy consumption. In *Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International*, pages 1–8. IEEE, 2011.

[96] N. Perel and U. Yechiali. Queues with slow servers and impatient customers. *European Journal of Operational Research*, 201(1):247–258, 2010.

[97] H. Qian and D. Andresen. Jade: An efficient energy-aware computation offloading system with heterogeneous network interface bonding for ad-hoc networked mobile devices. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*, pages 1–8. IEEE, 2014.

[98] H. Qian and D. Andresen. Emerald: Enhance scientific workflow performance with computation offloading to the cloud. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on*, pages 443–448. IEEE, 2015.

[99] H. Qian and D. Andresen. An energy-saving task scheduler for mobile devices. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on*, pages 423–430. IEEE, 2015.

[100] H. Qian and D. Andresen. Extending mobile device's battery life by offloading computation to cloud. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, pages 150–151. IEEE Press, 2015.

[101] H. Qian and D. Andresen. Jade: Reducing energy consumption of android app. *the International Journal of Networked and Distributed Computing (IJNDC), Atlantis press*, 3(3):150–158, 2015.

[102] H. Qian and D. Andresen. Reducing mobile device energy consumption with computation offloading. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on*, pages 1–8. IEEE, 2015.

[103] M.-R. Ra, J. Paek, A. B. Sharma, R. Govindan, M. H. Krieger, and M. J. Neely. Energy-delay tradeoffs in smartphone applications. In *Proceedings of the 8th international conference on*

Mobile systems, applications, and services, pages 255–270. ACM, 2010.

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

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

[106] T. Shi. An energy-efficient, time-constrained scheduling scheme in local mobile cloud. Master's thesis, University of Nevada, Las Vegas, 2014.

[107] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li. eTime: energy-efficient transmission between cloud and mobile devices. In *INFOCOM, 2013 Proceedings IEEE*, pages 195–199. IEEE, 2013.

[108] K. Sinha and M. Kulkarni. Techniques for fine-grained, multi-site computation offloading. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 184–194. IEEE Computer Society, 2011.

[109] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

[110] V. X. Tran, H. Tsuji, and R. Masuda. A new QoS ontology and its QoS-based ranking algorithm for web services. *Simulation Modelling Practice and Theory*, 17(8):1378–1398, 2009.

[111] I. Tsimashenka and W. J. Knottenbelt. Trading off subtask dispersion and response time in split-merge systems. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 431–442. Springer, 2013.

[112] T. Verbelen, T. Stevens, F. De Turck, and B. Dhoedt. Graph partitioning algorithms for optimizing software deployment in mobile cloud computing. *Future Generation Computer Systems*, 29(2):451–459, 2013.

[113] C. Wang and Z. Li. Parametric analysis for adaptive computation offloading. *ACM SIGPLAN Notices*, 39(6):119–130, 2004.

[114] X. Wang, J. Wang, X. Wang, and X. Chen. Energy and delay tradeoff for application offloading in mobile cloud computing. *IEEE Systems Journal*, PP:1–10, 2015.

[115] M. Whaiduzzaman, A. Gani, N. B. Anuar, M. Shiraz, M. N. Haque, and I. T. Haque. Cloud service selection using multicriteria decision analysis. *The Scientific World Journal*, 2014.

[116] A. Wierman, L. L. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM 2009, IEEE*, pages 2007–2015. IEEE, 2009.

[117] Wikipedia. Branch and bound. http://en.wikipedia.org/wiki/Branch_and_bound.

[118] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. Using bandwidth data to make computation offloading decisions. In *Parallel and Distributed Processing (IPDPS), 2008 IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[119] H. Wu. Analysis of mhealth systems with multi-cloud computing offloading. In *Mobile Health*, pages 589–608. Springer, 2015.

[120] H. Wu and D. Huang. Modeling multi-factor multi-site risk-based offloading for mobile cloud computing. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 230–235. IEEE, 2014.

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

[122] H. Wu, X. Lin, X. Liu, and Y. Zhang. Application-level scheduling with deadline constraints. In *INFOCOM, 2014 Proceedings IEEE*, pages 2436–2444. IEEE, 2014.

[123] H. Wu, X. Lin, X. Liu, and Y. Zhang. Application-level scheduling with probabilistic deadline constraints. *Networking, IEEE/ACM Transactions on*, 24(6), 2015.

[124] H. Wu, D. Seidenstücker, Y. Sun, C. M. Nieto, W. Knottenbelt, and K. Wolter. A novel offloading partitioning algorithm in mobile cloud computing. 2015.

[125] H. Wu, Y. Sun, and K. Wolter. Analysis of the energy-response time tradeoff for delayed mobile cloud offloading. *ACM SIGMETRICS Performance Evaluation Review*, 43(2):33–35, Sept. 2015.

[126] H. Wu, Y. Sun, and K. Wolter. Energy-efficient decision making for mobile cloud offloading. 2015.

[127] H. Wu, Q. Wang, and K. Wolter. Methods of cloud-path selection for offloading in mobile cloud computing systems. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 443–448, 2012.

[128] H. Wu, Q. Wang, and K. Wolter. Mobile healthcare systems with multi-cloud offloading. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, volume 2, pages 188–193. IEEE, 2013.

[129] H. Wu, Q. Wang, and K. Wolter. Optimal cloud-path selection in mobile cloud offloading systems based on QoS criteria. *International Journal of Grid and High Performance Computing (IJGHPC)*, 5(4):30–47, 2013.

[130] H. Wu, Q. Wang, and K. Wolter. Tradeoff between performance improvement and energy saving in mobile cloud offloading systems. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 728–732. IEEE, 2013.

[131] H. Wu and K. Wolter. Dynamic transmission scheduling and link selection in mobile cloud computing. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 61–79. Springer, 2014.

[132] H. Wu and K. Wolter. Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric. In *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2014 8th International Conference on*, pages 90–97. ICST, 2014.

[133] H. Wu and K. Wolter. Analysis of the energy-performance tradeoff for delayed mobile offloading. In *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2015 9th International Conference on*. ICST, 2015.

[134] H. Wu and K. Wolter. Software aging in mobile devices: Partial computation offloading as a solution. In *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*. IEEE, 2015.

[135] H. Wu, K. Wolter, and A. Grazioli. Cloudlet-based mobile offloading systems: a performance analysis. In *IFIP WG 7.3 Performance 2013 31 st International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2013 Student Poster Abstracts September 24-26, Vienna, Austria*, 2013.

[136] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma. Phone2Cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1):95–111, 2014.

[137] K. Yang, S. Ou, and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *Communications Magazine, IEEE*, 46(1):56–63, 2008.

[138] L. Yang and J. Cao. Computation partitioning in mobile cloud computing: A survey. *ZTE Communications*, 4:08–17, 2013.

[139] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):23–32, 2013.

[140] U. Yechiali. A queuing-type birth-and-death process defined on a continuous-time Markov chain. *Operations Research*, 21(2):604–609, 1973.

[141] U. Yechiali. Queues with system disasters and impatient customers when system is down. *Queueing Systems*, 56(3-4):195–202, 2007.

[142] U. Yechiali and P. Naor. Queuing problems with heterogeneous arrivals and service. *Operations Research*, 19(3):722–734, 1971.

[143] W. Zhang, Y. Wen, and D. O. Wu. Energy-efficient scheduling policy for collaborative exe-

cution in mobile cloud computing. In *INFOCOM, 2013 Proceedings IEEE*, pages 190–194. IEEE, 2013.

[144] Y. Zhang, H. Liu, L. Jiao, and X. Fu. To offload or not to offload: an efficient code partition algorithm for mobile cloud computing. In *Cloud Networking (CLOUDNET), 2012 IEEE 1st International Conference on*, pages 80–86. IEEE, 2012.

List of Figures

List of Tables

Glossary

AHP analytic hierarchy process
AP access point
AR availability ratio

CI consistency index
CR consistency ratio

ERP Energy-Response time Product
ERWP Energy-Response time Weighted Product
ERWS Energy-Response time Weighted Sum

FCFS first come first served
FLOPS floating-point operations per second

GFLOPS 10^9 FLOPS
GUI graphical user interface

IPP interrupted Poisson process

LARAC Lagrangian relaxation based aggregated cost
LOP load-balanced offloading policy
LP linear-programming

MCC mobile cloud computing
MCO mobile cloud offloading
MCOP min-cost offloading partitioning
MDP Markov decision process
MFLOPS 10^6 FLOPS

PQWS Power-Queue length Weighted Sum

QoS quality of service

RI random index
ROP random offloading policy
RTT round-trip time

SLA service level agreement

TOP tradeoff offloading policy
TOPSIS technique for order preference by similarity to ideal solution

VM virtual machine

WLAN wireless local area network

List of Publications

Wu, H., & Wolter, K. Analysis of the Energy-Performance Tradeoff for Delayed Mobile Offloading. In *Proceedings of the 9th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '15)*. ACM, 2015.

Wu, H. & Wolter, K. Software Aging in Mobile Devices: Partial Computation Offloading as a Solution. In *Software Reliability Engineering Workshops (ISSREW), 2015 IEEE International Symposium on*. IEEE, 2015.

Wu, H., Knottenbelt, W. J. & Wolter, K. Analysis of the Energy-Response Time Tradeoff for Mobile Cloud Offloading using Combined Metrics, In *Teletraffic Congress (ITC), 2015 27th International*. IEEE, 2015.

Wu, H., Sun, Y., & Wolter, K. Analysis of the Energy-Response Time Tradeoff for Delayed Mobile Cloud Offloading. *ACM SIGMETRICS Performance Evaluation Review*. 43(2),33-35. ACM, 2015.

Sun, Y., **Wu, H.**, & Schiller, J. A Step Length Estimation Model for Position Tracking. In *Localization and GNSS (ICL-GNSS), 2015 International Conference on*. IEEE, 2015.

Sun, Y., **Wu, H.**, & Schiller, J. A Running Step Length Estimation Model for Position Tracking. In *Positioning, Navigation and Communication (WPNC), 2015 12th Workshop on*. IEEE, 2015.

Wu, H. & Wolter, K. Tradeoff Analysis for Mobile Cloud Offloading Based on an Additive Energy-Performance Metric. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS '14)*. ACM, 2014.

Wu, H., & Wolter, K. Dynamic Transmission Scheduling and Link Selection in Mobile Cloud Computing. In *Analytical and Stochastic Modeling Techniques and Applications* (pp. 61-79). Springer International Publishing, 2014.

Wu, H., Wang, Q., & Wolter, K. Optimal Cloud-Path Selection in Mobile Cloud Offloading Systems Based on QoS Criteria. *International Journal of Grid and High Performance Computing (IJGHPC)*, 5(4), 30-47. 2013.

Wu, H., Wang, Q., & Wolter, K. Tradeoff Between Performance Improvement and Energy Saving in Mobile Cloud Offloading Systems. In *Communications Workshops (ICC), 2013 IEEE International Conference on* (pp. 728-732). IEEE, 2013.

Wu, H., Wolter, K., & Grazioli, A. Cloudlet-based Mobile Offloading Systems: a Performance Analysis. In *IFIP WG 7.3 Performance 2013 31st International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2013 Student Poster Abstracts September 24-26, Vienna, Austria*. 2013.

Wu, H., Wang, Q., & Wolter, K. Mobile Healthcare Systems with Multi-cloud Offloading. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on* (pp. 188-193). IEEE, 2013.

Wang, Q., **Wu, H.**, & Wolter, K. Model-based Performance Analysis of Local Re-execution Scheme in Offloading System. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on* (pp. 1-6). IEEE, 2013.

Wu, H., Wang, Q., & Wolter, K. Methods of Cloud-Path Selection for Offloading in Mobile Cloud Computing Systems. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on* (pp. 443-448). IEEE, 2012.

Wu, H., Seidenstücker, D., Sun, Y., Nieto, C.M., Knottenbelt, W. J., & Wolter, K. A Novel Offloading Partitioning Algorithm in Mobile Cloud Computing. *Unpublished*, 2015.

Wu, H., Sun, Y., & Wolter, K. Energy-Efficient Decision Making for Mobile Cloud Offloading. *Unpublished*, 2015.

About the Author

For reasons of data protection, the curriculum vitae is not included in the online version.