Enabling and Optimizing Resource Constrained Ad-Hoc Mobile Clouds

by

Colin F. Funai

Submitted in Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Supervised by Professor Wendi Heinzelman

Department of Electrical and Computer Engineering

Arts, Sciences and Engineering

Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester

Rochester, New York

2017

To Yuka Miura, the love of my life.

Contents

Biographical Sketch

The author was born in Evanston, IL. He attended the University of Rochester and graduated with a Bachelor of Science degree in Electrical and Computer Engineering in 2012. He began graduate studies in the Department of Electrical and Computer Engineering at the University of Rochester in 2012 and received a Master of Science degree in 2013.

The following publications were a result of work conducted during doctoral study:

C. Funai, C. Tapparello, and W. Heinzelman, "*Mobile to Mobile Computational Offloading in Multi-hop Cooperative Networks*," In Preparation.

C. Funai, C. Tapparello, and W. Heinzelman, "*Enabling Multi-hop Ad Hoc Networks Through WiFi Direct Multi-group Networking*," Proc. of IEEE ICNC 2017.

C. Funai, C. Tapparello, and W. Heinzelman, "*Mobile to Mobile Computational Offloading in Multi-hop Cooperative Networks*," Proc. of IEEE GLOBECOM 2016.

C. Funai, C. Tapparello, H. Ba, B. Karaouglu, and W. Heinzelman, "*Extending Volunteer Computing through Mobile Ad Hoc Networking*," Proc. of IEEE GLOBECOM 2014.

C. Tapparello, **C. Funai**, and W. Heinzelman, "*Exploring the Impact of Extending Mobile Volunteer Computing through D2D Communications*," In Preparation.

C. Tapparello, **C. Funai**, S. Hijazi, A. Aquino, B. Karaouglu, H. Ba, J. Shi, and W. Heinzelman, "*Volunteer Computing on Mobile Devices: State of the Art and Future Research Directions*," Appears in Enabling Real-Time Mobile Cloud Computing through Emerging Technologies, IGI Global, 2015

Acknowledgments

I would like to thank my advisor, Professor Wendi Heinzelman, and Dr. Cristiano Tapparello for the opportunity to work with them, and their mentorship over the past few years. During my time in the Wireless Communication and Networking Group, both spent endless amounts of time proof-reading my papers and providing excellent suggestions and insights for my projects. In particular, I would like to thank Dr. Tapparello for his mentorship and patience when teaching me how to model and develop technical proofs. As well, Professors Mateos, Heinzelman from the department of Electrical and Computer Engineering, and Shen from the department of Computer Science, for acting as members of my committee.

I would like to thank all of my lab-mates and colleagues in the WCNG lab, especially Yizhe Cheng, Aaron Faulkenberry, Michael Nolan, Jeremy Warner, Abner Aquino, Shurouq Hajazi, Mohammed Ahmed, Jon Aho, Justin Fraumeni, Theodore Reiss, and Yukun Chen for their work and contributions to my projects. I would also like to thank all of my colleagues at the University of Rochester for their support, as well as Harris Corporation for funding this project.

Finally, I would like to thank my family and girlfriend for their love and support.

Abstract

Recent years have seen a rapid adoption of mobile devices, and an increased reliance on them, which has lead to increasingly computationally complex mobile applications. As a result, there have been several proposed systems that offload computationally intensive workloads from mobile devices to other computing resources, such as remote servers or local cloudlets. Although these proposed systems have been shown to provide benefits to the mobile applications, there are situations where the high latency communication to reach a remote server cannot be tolerated, or where there is no network connectivity to such resources. In these situations, offloading to other local devices is the only option. To this end, I have proposed a system that utilizes ad hoc communication protocols to create a local cloud that can be used for computational offloading.

By extending an existing mobile computing platform, I show the the viability of offloading computation to devices within one hop, and model the cost in terms of time and energy for this hybrid system. Additionally, I have designed and developed several approaches to enable multi-hop communication within a network of mobile devices utilizing the WiFi Direct communication protocol. By doing so, I have further enhanced mobile computing by enabling the necessary infrastructure to facilitate multi-hop ad hoc computational offloading. With an implemented system, I was able to model the performance of this multi-hop computational offloading system, as well as model the task distribution problem as a linear bottleneck assignment problem and thus provide a provably optimal task distribution.

In summary, by providing the infrastructure for enabling multi-hop ad hoc computational offloading with off the shelf devices, and providing a provably optimal task distribution scheme, I have enabled and optimized the performance of ad hoc mobile clouds.

Contributors and Funding Sources

This work was supported by a disseration committee consisting of Professor Wendi Heinzelman (advisor) from the department of Electrical and Computer Engineering, Professor Gonzalo Mateos from the department of Electrical and Computer Engineering, and Professor Kai Shen from the department of Computer Science. The GEMCloud architecture, described in this disseration, was developed by Dr. He Ba in conjunction with UCB Pharmaceutical. Additionally, the model presented in Chapter 3 was developed by Dr. Cristiano Tapparello. All other work in this dissertation was completed independently by the author. This work was funded by Harris RF and the University of Rochester Center for Emerging and Innovative Sciences (CEIS).

List of Tables

List of Figures

Chapter-1

Introduction

Mobile Cloud Computing has been used to describe a variety of techniques for offloading computation from mobile devices, including offloading to a remote cloud as well as offloading to local resources such as a cloudlet or other mobile devivces. The growing popularity of mobile devices, the increase in computing power of mobile devices, and their support for device to device communication protocols have made mobile to mobile offloading an increasingly viable option. According to the International Data Corporation (IDC) and Gartner Inc. [1, 2], in 2013 the worldwide smartphone market shipped one billion units in a single year for the first time, representing a 38.4% increase with respect to the 725.3 million units shipped in 2012. Moreover, [1, 2] recently reported that in the fourth quarter of 2016, 431.5 million smartphones were shipped. The progressive increase in worldwide shipments of smartphones, from the first quarter of 2011 to the fourth quarter of 2016, is presented in Figure 1.1. Additionally, some smartphones are being equipped with processors that have capabilities rivaling desktop processors, as shown in Figure 1.2. Finally, Google and Apple smartphones come equipped with device to device support through their developer APIs. This alone has enabled an estimated 99% of the mobile device market [1, 2] to participate in ad hoc networks.

As a result, with the sheer number of mobile devices and their ever advancing computational and device to device communication capabilities, ad hoc mobile clouds not only have the necessary resources needed to be viable, but it is now possible to consider mobile clouds that offload computation not just to a mobile device's nearest neighbor, but also to other devices in the network connected through multi-hop communication, creating a multi-hop ad hoc mobile cloud. As a result, one of the key advantages that ad hoc mobile clouds provide is an independence from infrastructure based

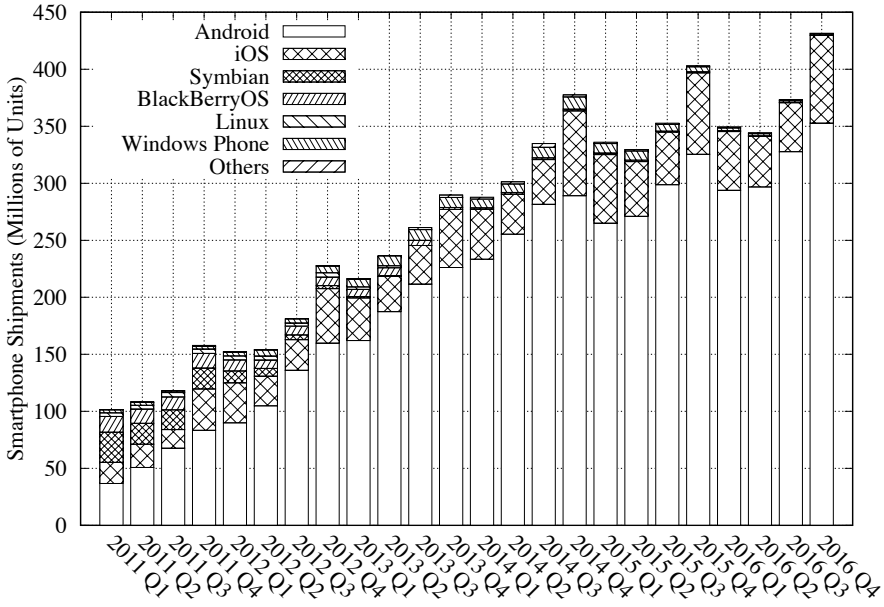


Figure 1.1: Quarterly worldwide shipments of smartphones and operating systems market share from the first quarter of 2011 to the second quarter of 2014. Data source: IDC worldwide quarterly mobile phone tracker and Gartner Inc. [1, 2].

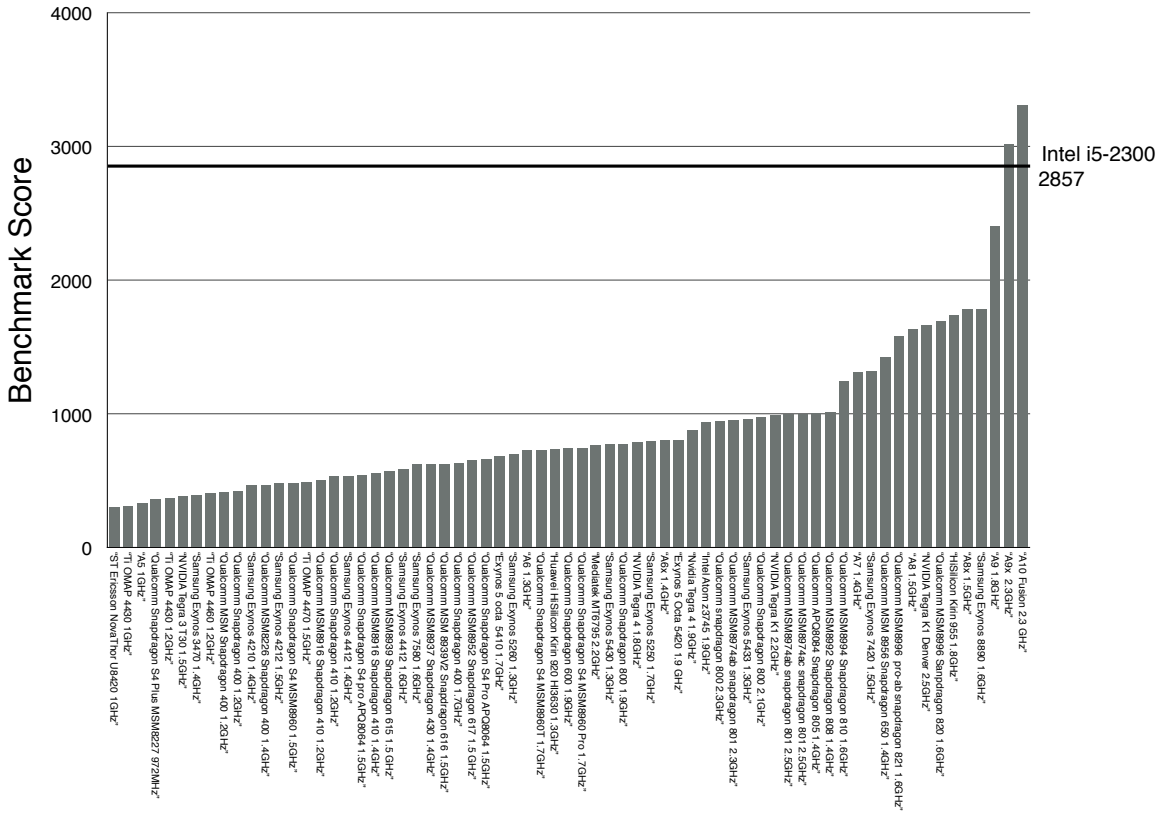


Figure 1.2: Benchmark scores of different mobile processors. Data source: Primate Labs Geekbench 4 [3].

communication. By using device to device $(D2D)$ communication protocols, ad hoc mobile clouds are able to quickly form a network. This is in contrast to remote cloud offloading systems, where a mobile device offloads computations to a remote server through an Internet connection. Take for instance CloneCloud [4], which relies on virtual machines, typically hosted by a cloud provider such as Amazon Web Service (AWS) [5], to receive computationally intensive tasks from a mobile device. While such remote clouds provide incredibly powerful servers for performing the computation, the only way this offloading can be achieved is through an Internet connection to the remote server. If such a connection to the Internet is not available, or if the latency to reach the server is too long, offloading to a powerful remote server is not an option.

In situations where there is no Internet connection, other approaches for offloading computation must be utilized. This is where ad hoc mobile clouds can provide the necessary resources to enable a mobile device to offload computations. The lack of fixed infrastructure for establishing Internet connectivity is common in military communication scenarios as well as in scenarios where the existing infrastructure has been damaged. In these cases, offloading computation to an ad hoc mobile cloud can be extrememly valuable to enable the completion of computationally-intensive tasks.

1.1 Challenges for Mobile Cloud Computing

In general, ad hoc mobile clouds have a few defining features. As the devices are mobile, maintaining reliable communication is paramount to the cloud's operation. Device mobility also implies that the devices are powered by battery, meaning that it is crucial to focus on techniques that are optimized for energy savings.

There are two cases to consider when designing a mobile cloud: 1) a cooperative network such that all mobile devices are cooperating to achieve a common goal, as is the case in military and disaster relief scenarios; and 2) a non-cooperative network such that the mobile devices are volunteering their services if they so choose, to help other devices in the network, possibly with the goal of gaining something in return, such as the ability to offload their own computation at a later time. Each of these cases provides additional challenges that must be considered in the design of an ad hoc mobile cloud.

For instance, when using ad hoc mobile clouds in a non-cooperative network, the biggest factor to consider is that the devices are personal mobile devices. This means that their availability to provide services for the mobile cloud can fluctuate. On the other hand, in cooperative networks, such as those designed for military and disaster relief scenarios, the devices are part of the network in order to achieve a certain goal. As a result, an ad hoc mobile cloud deployed in these scenarios can distribute computational tasks and develop communication schemes without having to consider the impact to the user.

In both of these cases, appropriately routing and assigning computation is important, meaning that the closest or most available node is not necessarily the best choice. In general, most nodes will try to route data over the shortest path; however, if every node participating in the network is trying to send data through the same node, this node becomes a bottleneck. These bottlenecks can adversely impact the overall performance of the network, by causing communication time-outs or forcing nodes to retransmit data. Both of these would put additional strain on each of the node's batteries.

In addition to the challenges pertaining to routing data through an ad hoc mobile cloud, the assignment of tasks must also be approached carefully. Although ad hoc mobile clouds already make efforts to balance between computational power and energy efficiency, the methods when doing so are not straight forward. Consider the above scenario where there are one or two nodes that are crucial for facilitating communication in the cloud, the impact of each task assigned to these crucial nodes arguably impacts the overall cloud more than if a leaf node were to perform said computation. Both of these issues can cause one of these intermediate nodes to die earlier than other nodes in the network. As a result, the network would become segmented, ultimately reducing the cloud's overall computational capabilities. Therefore, it is pertinent not only consider what paths are used to route data, but also what the status of the intermediate nodes are, as well as any other transient effects that different routing decisions have.

1.2 Contributions to Mobile Cloud Computing

This thesis aims to address the issues associated with enabling ad hoc mobile clouds by exploring the infrastructure used to facilitate communications, as well as optimizing the longevity of the cloud through the use of a variety of task distribution and communication schemes. The specific contributions of my work include the following:

- I developed the concept and explored the benefits of extending volunteer computing platforms through the use of ad hoc mobile clouds. In particular, I have developed and analyzed the trade-offs for two task distribution schemes named $Proxy$ and $Batch$. This work has been described in [6] and is detailed in Chapter 3.

- As a follow up to the work presented in [6], I developed a closed form model of our system. This models enables us to simulate larger networks as well as simulate the impact that device mobility has on network performance. This work is described in Chapter 3.

- I have designed and provided an in-depth analysis of different approaches for implementing multi-group WiFi Direct networks to enable multi-hop communication in ad hoc mobile clouds. My approaches include designs that will work with off-the-shelf mobile devices as well as optimized communication solutions that require modifying the Android operating system. This work is described in Chapter 4.

- I have developed a task distribution algorithm and a routing heuristic in order to investigate the effect that computational offloading has on a multi-hop wireless network. I have found and proved that this task distribution algorithm is optimal under certain conditions. This work is described in Chapter 5.

- I have developed a system to visualize basic parameters of an ad hoc mobile cloud. This system is able to control the network topology as well as track the location and movement of the participating nodes. By doing so, administrators are not only able to tailor routes and topologies, to avoid communication bottlenecks, but they can also anticipate and avoid network partitions, by monitoring the status of crucial relay nodes. This work is described in Chapter 6.

This thesis is organized as follows. Chapter 2 provides related work on mobile cloud computing, as well as related topics such as grid computing. Chapter 3 describes and models the implementation of a nearest neighbor ad hoc mobile cloud, using WiFi Direct and Bluetooth. In Chapter 4, techniques for enabling multi-group WiFi Direct networks are presented, and this is followed by Chapter 5, which presents a description of different task distribution techniques for multi-hop ad hoc mobile clouds. Chapter 6 presents a visualization system used to monitor the network throughput, device battery level, and device location in a user friendly manner. Finally, Chapter 7 presents an overview of my contributions to the development and optimization of ad hoc mobile clouds, as well as suggestions for future work.

Chapter-2

Related Work

2.1 Classification of Parallel Computing

High performance parallel computing has been an approach used to increase the speed of computation by dividing the computational problem into simultaneously computable sections and processing each section on different processing units. Traditionally, these independent processing units reside on the same device (multiprocessor computing), or even on the same chip (multicore computing). On the other hand, researchers have explored new computational architectures where the processors of multiple devices are connected by a communication network and cooperate in the computational job. These architectures can be classified according to the geographical distance between the devices that perform the computation: the parallel execution of computational jobs using a group of co-located computers is typically called *cluster computing*, while the cooperation among distant computers communicating over the Internet is typically referred to as *distributed computing*. While the former relies on a reliable local area network and can be used to solve distributed computing problems that require communication among the devices executing the tasks, the latter, due to the unpredictability of the Internet, typically deals only with what are termed "embarrassingly parallel problems," where there exists no dependency (or communication) between the parallel tasks. Both of these approaches consider that the computation is distributed across dedicated devices that either require direct management or the payment of a fee for accessing the processing power. As a result, the concept of *volunteer computing* has been proposed as an alternate parallel computing system that exploits computing resources donated by general-purpose computer owners.

In what follows, we first briefly describe these three classes of parallel computing, namely cluster computing, distributed computing, and volunteer computing, and then discuss how mobile devices can provide benefit to the distributed computation.

2.1.1 Cluster Computing

Computing clusters are built linking groups of computers through a high-bandwidth low latency local area network. These computers each run their own instance of an operating system, but work together to perform a common task so that they can be viewed as a single system. The computing clusters are developed for a variety of purposes such as load balancing on web servers, computationally intensive scientific calculations, and failure safe operation on critical commercial applications.

Attached Resource Computer (ARCNET) [7] was the first commercial computing cluster, developed in the late 70s, supporting both parallel computing as well as sharing file systems.

Beowulf clusters utilize standard commodity grade computers with specialized libraries and programs that allow job sharing among them. Beowulf clusters normally run Unix like operating systems, such as BSD, Linux, or Solaris and, potentially, any PC capable of running a Unix like operating system can be used in this configuration. The cluster is organized as multiple computers serving as the worker nodes and one or more computers taking the responsibility of the server. The server controls and coordinates the computing cluster and serves as a gateway between the computing cluster and the outside world. Stone Soupercomputer[1] [8] built by Oak Ridge National Laboratory was one of the large scale successful applications of the Beowulf concept.

Due to the dependency of the physical location of the hardware, computing clusters are built to serve a limited set of users located at a particular geographical region. Hence, the demand for computational resources on these systems have a high variance due to the correlation between usage patterns. Combined with the high cost of building computing clusters, this leads to both underutilization and outage of computational resources.

[1]The Stone Soup is an old folk story in which hungry strangers persuade local people of a town to give them food. It is usually told as a lesson in cooperation, especially in situations of resource scarcity.

2.1.2 Distributed Computing

Distributed computing overcomes the geographical limitation of cluster computing by allowing distant computers to cooperate in the execution of computational tasks. By integrating geographically diverse multiple computing clusters or individual computers, distributed computing architectures can serve a larger group of consumers with less correlated usage patterns. Although distribution and scheduling of the computing jobs across the distributed computing resources adds another layer of complexity, with the introduction of the Internet, distributed computing systems provide a fairly low cost and high performance solution to large computing problems.

Through distributed computing, computational capabilities can be offered to users as a service. In this new model of computing, also referred to as utility computing, customers can acquire large computing capabilities as needed. The computational tasks are offloaded to the service providers' computing platform, and the results are downloaded back after completion of the tasks. Many commercial instantiations of distributed computing exist today, including Amazon Elastic Compute Cloud [5]. One intrinsic drawback of this approach is that the users' performance is negatively affected by the network delay, since the entire user data and the result of the computation need to be exchanged back and forth with the distributed computing system.

More recently, a new subclass of distributed computing named cloud computing has also been proposed, and it is receiving considerable attention. Distributed computing architectures have evolved into cloud computing systems that not only undertake computational tasks but also serve as data storage systems and provide online access to computer services or resources. These resources are shared by multiple users but are usually dynamically reallocated per demand, thus maximizing the effectiveness of the shared infrastructure. Microsoft's OneDrive [9] and IBM Cloud [10] are two of the many commercial examples of this paradigm.

2.1.3 Volunteer Computing

Although distributed computing systems increase the efficiency of parallel computing, they still require a large investment for both hardware and software as well as incurring significant operational

costs (i.e., maintenance, direct power consumption and cooling infrastructure). Several studies have shown that many computing devices (i.e., personal computers, tablets and mobile devices) under utilize their processing capabilities for the majority of their operational time. The potential of these resources exceeds any centralized computing system. This is the basis for volunteer computing.

The first volunteer computing project, Great Internet Marsenne Prime Search [11], was started in 1996 with the objective of using freely available software on volunteers' computers working in parallel to find prime numbers. Starting from this project, volunteer computing emerged as a result of the wide spread adoption of personal computers and the Internet. With volunteer computing, volunteers can dedicate the unused computer cycles on their personal computers to the distributed computation. This is made possible by middleware systems such as JXTA [12], XtremeWeb [13], and Berkeley Open Infrastructure for Network Computing (BOINC) [14]. BOINC was originally developed to provide support and increase security for the SETI project [15] and later extended as a platform for other distributed applications. It is now one of the most popular volunteer computing platforms with over 1,000,000 active participants [16].

2.1.4 Parallel Computing on Mobile Devices

The idea of connecting mobile devices into a parallel computing system was proposed in 2002 [17], when both their computational capabilities and diffusion where still highly limited. With the increase in mobile device computational capabilities, different system architectures have been proposed to exploit their resources for parallel computing. The classification of parallel computing presented in this section can be extended to the case in which the mobile devices are performing the actual computations. In this regard, solutions that group nearby mobile devices using a device to device communication technology such as Bluetooth [18] and WiFi Direct [19], and distributed systems that link together distant devices through an Internet connection have both been investigated. Many traditional distributed computing architectures have recognized the widespread usage, significant computing capabilities and energy efficiency of mobile devices and have attempted to extend their operation over mobile computing platforms. For example, Hyrax [20] ports Hadoop Apache, an open-source im-

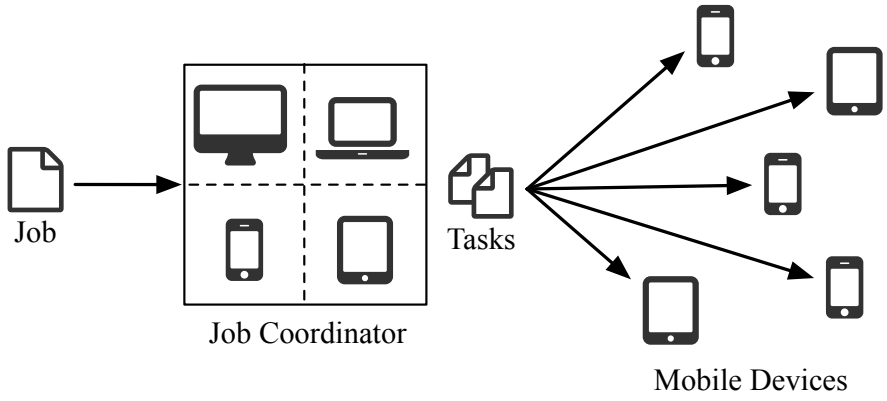


Figure 2.1: Example of a mobile distributed computing architecture where the job coordinator assigns tasks to the participating mobile devices.

plementation of MapReduce, to execute jobs on networked Android smartphones. A client version of BOINC was ported to an ARM/Linux platform [21] to evaluate the processing power of mobile devices, and an Android client [16] to include mobile devices in the distributed computations has also been released by the BOINC project.

2.2 Mobile Distributed Computing Architectures

Modern mobile devices, such as smartphones and tablets, have become powerful and energy efficient computing architectures, that are both widely available and underutilized for long periods of time. As a result, multiple approaches for integrating mobile devices into a parallel computing infrastructure have been proposed and are currently receiving considerable attention. These studies can be broadly classified in two main categories, depending on the particular entity that is responsible for the management of tasks that need to be executed by the participating devices. This element can either be a remote specialized server that communicates with the mobile devices through an Internet connection, or it can be a mobile device itself, that exploits the presence of other geographically close devices to solve computationally intensive tasks. According to this division, we refer to the first scenario as "Server Driven Mobile Distributed Computing," while we call the latter scenario "User Driven Mobile Distributed Computing." We note that the server driven approach is, in fact, an extension of a traditional distributed computing architecture, where mobile devices can participate in the distributed

computation alongside standard PCs. The user driven case, instead, represents a viable way to perform intensive computing when an Internet connection is not available or it is undesirable because of communication delay. For example, many applications in the area of tactical military communications, search and rescue operations, and sensor network operations require computing intensive algorithms, such as image and signal processing, at remote or isolated locations that frequently neither have direct access to the Internet nor are in the vicinity of other devices with Internet access. In both cases, the device that "drives" the computation is considered to be the job coordinator, since it is in charge of the task distribution process and is responsible for the reception and organization of the results of the tasks' execution. An example of such a mobile computing architecture is presented in Figure 2.1.

In what follows, we provide a literature review of the different frameworks that have been proposed to use mobile devices for executing tasks in distributed computing. All of these systems follow a similar architecture and communication protocol: a mobile device is connected through a suitable radio communication technology to the job coordinator, it receives the tasks to be computed and, after the execution is completed, the mobile device sends the result of the computation back to the job coordinator. It is important to note that the idea of enabling the execution of rich applications on mobile devices by offloading the computation (or part of it) and storage to a distributed architecture composed of traditional computers has also been proposed. This type of service is typically referred to as mobile cloud computing [22].

2.2.1 Server Driven Mobile Distributed Computing

Mobile OGSI.NET [23]

Mobile OSGI.NET was one of the first attempts to connect mobile devices to a distributed computing architecture. The goal of this framework is to provide a way for mobile and non-mobile devices to collaborate in the execution of resource-demanding applications. The OGSI architecture consists of three components: the Mobile Web Server that handles the message exchanges between the device and the remote server, the Grid Services Module that implements the core processing necessary to execute applications, and the Grid Services, that represent the particular applications that will be

executed by the device. In addition, the Grid Services Module monitors the resources of the mobile devices (like, e.g., the battery level) and decides if the mobile device is able to perform the computation, or if it is better to pass the task to another device. Experimental results show that the total time required to compute a set of tasks decreases as the number of devices increases. Moreover, they show that the energy consumption can be efficiently distributed between the devices.

Hyrax [20]

Hyrax is a platform derived from Hadoop Apache, an open source implementation of MapReduce[2], that supports distributed computing on Android devices. The basic idea behind Hyrax is to allow a heterogeneous network of smartphones and servers to cooperate in the execution of computing jobs. The framework has been designed to provide an abstraction of the available resources, thus being able to scale with the number of devices and tolerating node connection and departure. The performance of Hyrax in terms of both execution times and resource usage was evaluated with a testbed of 12 Android smartphones. Although the performance of Hyrax is poor for CPU-intensive tasks, it demonstrates the feasibility and scalability of the proposed framework. In addition, the advantages of using Hyrax as an infrastructure for applications that use mobile data have been investigated through the implementation of a distributed multimedia search and sharing application. The authors of [20] stated that Hyrax had not been optimized for battery efficiency, but in several tests it was shown to use significantly less power than a video recording and downloading application.

Computing While Charging [24]

Computing While Charging (CWC) describes and evaluates a scenario in which a company uses the mobile devices provided to its employees for the execution of parallelizable tasks. The main idea behind CWC is that using mobile devices for work-related computing can potentially reduce not only the capital investment in servers but also the cost of energy, since a smartphone can be up to 20x more efficient than a standard server. Thus, the authors of [24] propose a framework where the

[2]MapReduce is a programming framework for data-intensive cloud computing on commodity clusters developed by Google.

phones are used for the computation only while being charged, so that the user is not disturbed by the computations. Moreover, while charging the phone has a high probability of being connected to the Internet through a WiFi Access Point. The application monitors the user interactions with the phone and, if a user uses the phone while it is computing a task, the task is interrupted and migrated to a different phone so that the task computation does not have any impact on the user. Moreover, the application incorporates an algorithm that monitors the charging patterns: a test performed on 15 volunteers showed that the users charge their phones predominantly during the night and for an uninterrupted period of several hours. They also ran other experiments to evaluate the impact of the network connectivity on the task completion time, showing that simply accounting for the CPU clock speed results in poor task completion times. CWC also includes an algorithm to predict the time it takes for the tasks to be completed and three task distribution methods with different complexity. The main experiment involved 18 Android phones with different CPU clock speeds and different network connectivity technologies, and it showed that a greedy scheduler is approximately 1.6 times faster than the other tested schedulers.

jUniGrid [25]

jUniGrid is a lightweight framework that allows the integration of mobile devices into heterogeneous desktop distributed computing systems to solve high complexity computational problems. jUniGrid introduces two separate applications, that correspond to two functional roles: the Task-Submitter (TS) and the Node-Application (NA). The node application is installed on the devices that execute the tasks, while the task submitter runs on the device that creates the tasks, stores them in a task queue and assigns them to the nodes according to a First In First Out policy. Moreover, jUniGrid works based on a split/merge algorithm. It provides the user with the flexibility to split the job and merge the results according to the requirement of the particular job. This split and merge is accomplished by TS, that also implements all the functionalities for job allocation, monitoring and result aggregation. Thus, the TS is installed on the device that creates task queues and sends them to the nodes in a FIFO fashion, where the first device to be connected receives the first task output, while the node application is installed on the devices that execute the tasks. The paper shows the gains

in term of job execution times that can be achieved by allowing mobile devices to participate in the distributed computation. However, the focus of the paper is to provide a basic generalized grid mechanism for cooperative multi-platform processing and does not provide any detail about the specific implementation.

Ocelot [26]

Ocelot is a distributed mobile computing platform that leverages mobile devices to execute lightweight computational tasks generated from a Wireless Sensor Network (WSN). Ocelot is modeled after the Berkley Open Infrastructure for Network Computing (BOINC) with the exception of employing smartphones and tablets rather than workstations and server machines, as they reduce the maintenance costs and the power usage. For testing purposes, Ocelot was integrated with a WSN that is deployed to monitor indoor environmental conditions in a building. More specifically, Ocelot was used to monitor and analyze the electrical power consumption and the environmental emissions within the building. In this setting, Ocelot's clients (the mobile devices) are attached to sensors that exchange data through WiFi Direct and/or Bluetooth. Although the clients themselves cannot gather sensory data, they can partition and efficiently process the data through parallel computing. Ocelot's clients serve as nodes that request and receive tasks from a server through XML files. One of Ocelot's main features is having multiple servers to insure an efficient task distribution, as one of the servers is consistently storing the battery status of the nodes to make sure that the scheduling server sends tasks to only those nodes with sufficient power. Once a node receives a task, it will execute the code, which is usually written in Java or C. Currently, Ocelot allows the mobile devices to act only as clients that execute tasks, but allowing the mobile devices to also become servers for the task distribution is considered as future work.

To prove its advantage in reducing energy consumption, Ocelot has been compared against traditional computers. Results show that, while laptops and desktops were 5 times faster than mobile devices, mobile devices consume up to 86% less energy. In addition, adding more devices to the client pool dramatically lowers the total task completion time. As a result, Ocelot proves that it is feasible to distribute tasks among mobile devices and provides considerable energy and cost savings with respect

to a system that uses standard computers.

CANDIS [27]

CANDIS is a framework that distributes computing tasks to mobile devices as well as normal desktop or server hardware and provides an efficient method of reducing costs by taking advantage of the fluctuating energy market prices. The authors in [27] recognized that distributing computing tasks to mobile devices has been extensively studied. Thus, they focus on devising a computational infrastructure that is able to further reduce the computation costs and energy consumption. CANDIS is a Java-based framework and is thus able to run existing Java code, and desktop, server and Android mobile devices that support Java can be easily connected in the cloud. In such a hybrid cloud, the server compiles the tasks and constructs a scheduler to allocate and distribute the tasks. The paper shows that, while equally dividing the tasks among the available devices might be easier to implement, distributing much smaller tasks results in faster execution times but increases the communication overhead. Thus, CANDIS implements a more efficient allocation method where the server, before assigning the actual tasks, estimates the capability of each client by assigning a benchmarking task and using the results to improve the task allocation scheme. In addition, CANDIS uses information about the price of electricity, which fluctuates dramatically throughout the day (on average, prices are usually much lower late at night and early in the morning). The authors in [27] conclude that using CANDIS on a large scale not only allow to save money but can also stabilize the electric energy consumptions.

ANGELS [28]

ANGELS is a framework that allows mobile devices and computers to cooperatively participate in the computation of analytical data. This framework allows the parallel execution of jobs on a set of nodes that can be either mobile devices or standard PCs. For the evaluation of the framework, a “text search” application, in which mobile devices and servers had to find a specific word in a large text file and an algorithm to estimate the value of π have been considered. Experimental results show that the tasks’ latency can be substantially reduced when the tasks are distributed among the mobile devices.

The framework does not consider the impact of the computation on the user experience. Moreover, the task distribution process considers a simple distribution scheme, in which tasks are assigned as soon as a device becomes available.

2.2.2 User Driven Mobile Distributed Computing

Serendipity [29]

Serendipity enables a mobile computation initiator to use the computational resources of nearby mobile devices to speed up the computation and preserve some energy. Serendipity improves the mobile device's computational experience by applying optimizing algorithms that minimize local power consumption and/or decrease the computation completion time, while taking into account the constraints of the intermittent communication links such as limited contact duration, limited transfer bandwidth, and completion-time unpredictability. Serendipity follows what is called a "PNP block paradigm": the job is pre-processed and divided into n parallel task programs, and the results of the execution of the tasks are finally merged by a post-process algorithm. The goal of this design is to have an initiator disseminate a task (pre-process) to the computational nodes (task programs) it encounters based on the estimated completion time or energy consumption, and finally coalesce the data it receives (post-process).

To do so, three algorithms have been presented: 1) a WaterFilling method where the initiator knows when to contact the nodes and has access to the nodes' profiles in order to predict the number of tasks a node can execute and the time required to process them; 2) a Computing on Dissemination (COD) method, where the initiator does not know the contact time but has access to the nodes' profiles; and 3) the Unpredictable Computing on Dissemination (upCOD), where both the contact time and the nodes' profiles are unknown.

Serendipity has been implemented on Android and showed substantial performance gains when compared to executing tasks locally on the initiator's mobile device. While in all the experiments, the WaterFilling method performed better than COD and upCOD, experimental results show the clear benefit of disseminating tasks on Serendipity rather than executing them locally, especially when

Name	Year	Contributions	Task Distribution	Operating System	Applications
Mobile OSGI.NET	2004	Porting of OGSI.NET to mobile devices	Homogeneous tasks. FIFO queue	Microsoft PocketPC 2003	Prime numbers search
Hyrax	2009	Porting of Hadoop to Android devices	Homogeneous tasks. FIFO queue	Android	Distributed multimedia search; Content sharing
Computing While Charging	2012	Profiling charging behaviors, scheduling algorithm, migration of tasks across phones	Heterogeneous tasks. Greedy algorithm based on the Minimum Makespan Scheduling problem with task migration.	Android	Prime numbers search; Word searching; Photo pixels blurring
JUniGrid	2013	Generic framework API for developing grid applications	Homogeneous tasks. FIFO queue	JAVA	DNA sequence matching
Ocelot	2013	Distributed computing system that uses mobile devices as computing resources.	Homogeneous tasks. FIFO queue.	Android, iOS	Dynamic life cycle assessment of a building
CANDIS	2013	Distributed computing system that uses mobile devices and traditional computer as computing resources.	Heterogeneous tasks. Scheduler based on the device computational capability and the energy market prices.	Android	Distributed brute force hash-cracking; XML to JSON conversion
ANGELS	2014	Framework that allows the remote execution of programs within mobile devices. The focus is on the processing of IoT analytical data.	Heterogeneous tasks. Tasks are assigned according to the device computational capability.	Android	Text search; π value estimation

Table 2.1: Server driven mobile distributed computing implementations.

the number of tasks exceeds 100. Moreover, Serendipity was able to speed up computation up to 3 times compared to local conventional computing. In addition, Serendipity increases the battery life of mobile devices and allows the saving of a significant amount of energy by distributing the computation between different devices.

Honeybee [30]

Honeybee is a framework that deals with both human and machine computation, where human computation represents a set of operations that require human interaction, like filling out a personal survey form, while machine computation is a generic computer algorithm, like word searching or number sorting. The Honeybee framework distributes a computational intensive task, like a face detection algorithm, among several mobile devices. Honeybee's focus is on keeping the smartphone busy, in the sense that, as soon as one computation is completed, the device is allowed to steal tasks from another slower device. The proposed implementation also focuses more on getting as many tasks done as possible and does not provide a customizable, user friendly interface. Moreover, to each task is assigned a deadline that the mobile device has to satisfy in order to continue getting tasks. If the deadline is not met, the task is passed to another device. Honeybee has been implemented on Android, using Bluetooth as a local communication technology. Experimental results show that managing local connections severely impacts the delegator throughput. As future work, the authors are planning to support different types of D2D communication technologies, e.g., WiFi Direct.

Unity [31]

Unity represents a system architecture that allows a group of mobile devices to share the workload required to download a data file from the Internet. With this approach, each device downloads small parts of the file and then shares those parts with the other members of the group so that every device will eventually get the complete content. Leveraging short-range technologies such as WiFi and Bluetooth, Unity allows a coordinator to communicate with its peers to split the download as well as restarting it from the point where it stopped in case of a failure.

Unity has been implemented on Android smartphones that have either WiFi or Bluetooth capabil-

ities. Unity employs WiFi HotSpot, an Android utility that uses 802.11 infrastructure mode to allow the coordinator to act as a WiFi AP and all other peers to be connected as clients. As a result, the WiFi HotSpot functionality allows the coordinator to stay awake for the entire duration, while peers are in power saving mode, consuming a negligible amount of energy. After an initial connection phase, where the devices connect to the coordinator, the coordinator determines the size of the file through an HTTP request, divides the load, and then sends a control message containing the file URL to its peers. Subsequently, each peer starts to download its share of the file using its own data connection and sends the data blocks to the coordinator, which collects all the blocks, reconstructs the entire file and distributes it to all the peers. Unity also includes a task distribution scheme that takes into account the variability of the peers' cellular network conditions for better distributing the workload between the mobile devices. Experimental results show an improvement in download speeds up to 27%. In addition, a variations of Unity called Unity-Cloud has also been presented. In Unity-Cloud, a remote server coordinates the formation of the peer to peer group and assigns to each device the part to be downloaded according the relative cellular conditions. As soon as the peers are geographically close, the cloud coordinates the local blocks sharing for reconstructing the original file.

DRAP [32]

DRAP proposes a mechanism to group volunteer mobile devices into high performance decentralized computing systems. The idea behind this work is to create an infrastructure where mobile devices in close geographical proximity can form a cloudlet, and share resources with each other and with other nearby devices. The concept of a cloudlet has been introduced in [33], and represents a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices. In DRAP, the cloudlet is represented by a cluster of mobile devices that provides storage capability and computational resources to the other devices in the network. The framework monitors the movement of the participating nodes and implements all the functionalities required to connect the nodes in the network and enable the communications. For routing the communications between the devices, DRAP uses a modified version of the Ad Hoc Distance Vector (AODV) routing protocol. DRAP also includes an algorithm that uses the mobile devices'

resources to determine the subset of nodes that should be selected to become part of the cloudlet. Computer simulations prove the feasibility and performance gain of the proposed architecture, for different types of applications. The implementation of DRAP in real life mobile devices is considered as future work.

2.2.3 Mobile Volunteer Computing

The architectures presented in the previous sections can all be adapted to allow volunteer users to participate in the parallel computation. However, architectures explicitly designed with the objective of realizing a volunteer computing system by interconnecting personal mobile devices through the Internet have also been developed.

BOINC on Mobile Devices

The first attempt to extend the participation in volunteer computing to mobile devices dates back to 2007 [21], with researchers working on getting the application SETI@home [15] (and other scientific programs) to run efficiently on ARM processors [34]. Starting from this feasibility study, different Android applications [16, 35, 36, 37] and a prototype implementation for iOS [38] have been proposed to extend the participation in BOINC projects to mobile devices. In February 2014, the integration of Android devices into the BOINC system has been extensively promoted with the campaign HTC Power to Give [39], an initiative that aims to create a supercomputer by harnessing the collective processing power of Android smartphones. As of September 2014, the HTC Power to Give application has been installed on 1.7 million devices [40].

CrowdLab [41]

The idea of creating testbeds by interconnecting volunteer mobile devices has also received considerable attention [41, 42, 43]. In particular, CrowdLab [41] is a testbed architecture that utilizes volunteer mobile resources to offer features common to infrastructure-based testbeds. CrowdLab allows the execution of guest code on participating mobile devices through hardware virtualization, it

Name	Year	Contributions	Task Distribution	Operating System	Applications
Serendipity	2012	Distributed computing system that exploits nearby mobile devices	Heterogeneous tasks. Three schemes with different complexity.	Simulations on Emulab	Speech-to-text application
Honeybee	2013	Framework API to support job sharing and crowd-sourcing among mobile devices. Work stealing to achieve load balancing	Heterogeneous tasks. Tasks assigned at random. Scheduler that attempts to minimize the idle time	Android	Face detection; Mandelbrot set generation, Collaborative photography
Unity	2013	System architecture that enables collaborative downloading across co-located mobile devices.	Heterogeneous tasks. Tasks (data to be downloaded) are assigned according to the cellular network conditions	Android	Collaborative file download
DRAP	2014	Cluster formation of volunteer mobile devices for distributed computation. The focus is on how to best group the devices based on their capabilities.	High level description of a Cloudlet Manager that handles task distribution. The details about the task distribution process are not provided	Simulations on ns-3	Testing of the cloudlet formation algorithm

Table 2.2: User driven mobile volunteer computing implementations.

supports low-level access to the radio device and the concurrent scheduling of co-located applications. Volunteers contribute resources to CrowdLab in the same way that users contribute spare resources to BOINC. The CrowdLab architecture uses a centralized remote server for tracking the experiments and the current available volunteer resource contributors, and a decentralized local task coordinator that is responsible for the scheduling and task distribution to nearby devices. CrowdLab includes an algorithm to limit the amount of energy that each application can consume in a certain time period. According to this scheduling scheme, a device will not participate in the distributed computation if the owner is actively using it, and it allows the user to set a daily resource budget for running experiments as a percent of battery capacity or as a period of participation.

Seattle [44]

Seattle [44] has been proposed as a distributed computing platform that exploits heterogeneous volunteer devices for educational and research purposes, that supports different operating systems and architectures. Seattle is a general purpose learning platform based on the Python programming language that allows users to develop and test different types of applications ranging from networking to cloud computing. The objective of this platform is to provide researchers and educators the ability to create application prototypes and to evaluate their performance on a wide range of devices distributed around the world. Seattle follows an open source philosophy, and it embraces the heterogeneity of today's end user environment, thus providing a unique environment that is not available on other testbeds. A recent study showed that more than 20,000 devices are currently contributing their resources to Seattle, with more than 500 being mobile devices [45].

GEMCloud [46]

More recently, GEMCloud (Green Energy Mobile Cloud) [46] has been proposed as a distributed system that utilizes energy efficient personal mobile devices as computing resources instead of desktop computers. Mobile devices are considered to be particularly appealing because of their increasing computing capabilities, great popularity and diffusion as well as for the fact that they can potentially provide energy savings with respect to standard computers. The vision of GEMCloud is to adapt

the traditional distributed computing infrastructure by shifting the load of the computation to mobile devices. GEMCloud follows a traditional volunteer computing architecture, where a remote server distributes the tasks to participating devices through an Internet connection. The task distribution is based on the device characteristics and customizable user preferences. In GEMCloud, the user experience is particularly important, and the user is allowed to finely control how much and when to contribute to the distributed computation, including settings on battery level, charging vs. not charging, WiFi vs. 3G/4G communication, and device temperature. Experimental results show a comparison between the completion time and relative energy consumption of different types of tasks and different computing devices. While the mobile devices are always slower than a high performance workstation, GEMCloud shows that some mobile devices have performance comparable to a standard computer, while always consuming much less energy. The GEMCloud application is available for download in the Google Play store.

Starting from the GEMCloud implementation, the authors in [6] presented a computational infrastructure that extends the ability of mobile devices to participate in volunteer computing through ad hoc networking. The architecture presented in [6] overcomes the intrinsic requirement of Internet connectivity to participate in volunteer computing by introducing decentralized job coordinators. These job coordinators, referred to as task distribution points, are mobile devices directly connected to the Internet that are able to invite other devices to join the computation via device to device communication. Experimental results show that allowing for additional devices without Internet connectivity to participate in the computation reduces significantly the overall time required for the execution of the tasks, with only minor additional energy consumption at the decentralized job coordinators.

Name	Year	Contributions	Task Distribution	Operating System	Applications
Seattle	2009	Distributed computing and general purpose learning platform	Application specific tasks	Maemo Linux	Multiple applications
BOINC	2011	Porting of the BOINC client to Android devices	Project specific tasks	Android	Multiple scientific research projects
CrowdLab	2011	Testbed architecture based on volunteer mobile resources	Application specific tasks	Android	Multiple applications
GEMCloud	2013	Distributed computing system that uses mobile devices as computing resources. Evaluation of computing power and energy efficiency of mobile devices	Heterogeneous tasks assigned at random. FIFO queue subject to user preferences.	Android	Protein structure predictions

Table 2.3: Mobile volunteer computing implementations.

Chapter-3

Extending Mobile Cloud Computing with Device-to-Device Communications

3.1 Introduction

While the aforementioned systems provide a variety of viable and energy efficient mobile computing architectures, there exist scenarios where a mobile device's connectivity to the Internet is limited or non-existent, resulting in devices being unable to utilize these aforementioned systems. As a result, we have proposed and implemented an architecture that extends the existing mobile cloud computing systems by offloading computation to a device's nearest neighbors. In particular, we present an extension to a volunteer computing system in which a device with Internet capabilities, either WiFi or 3G/4G, can elect itself as a local task distribution point, inviting other users to join the computation via existing Device to Device (D2D) communication methods.

Starting from the existing implementation of GEMCloud [46], we design and develop an experimental system where local task distribution is performed using different D2D technologies. Besides the implementation, our focus is on the evaluation of the computing capabilities and energy efficiency of the system, as well as to prove the feasibility of the system for an existing D2D technology. In this regard, WiFi Direct [19] is considered. For the local task distribution points, two methods for distributing the computation tasks to the devices connected through D2D communication, *Batch* and *Proxy*, are also proposed. Using *Batch* mode, a set of N tasks are cached at the task distribution point and are then distributed to the ad-hoc network as they are requested. In *Proxy* mode, instead, the task distribution point acts as a gateway to the Internet-based source of the data. The remainder of

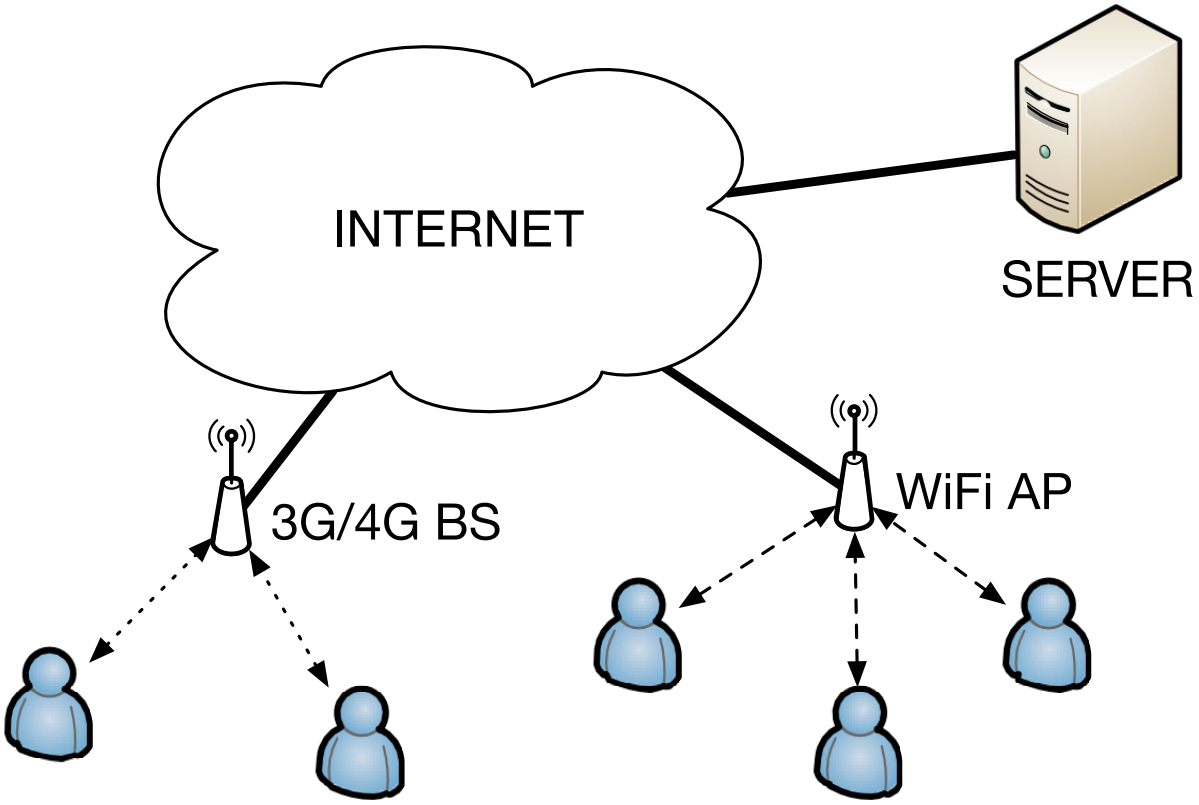


Figure 3.1: A traditional mobile computing topology. Users are connected to a remote computing server through a 3G/4G Base Station (BS) or a WiFi Access Point (AP).

the chapter highlights the energy consumption and evalutates the performance of the task distribution methods over a single hop.

3.2 Application Scenarios

Traditional mobile computing systems use the Internet as the communication architecture, as shown in 3.1. This limits the effective use of mobile computing due to (i) the fact that some devices may not have access to the Internet for some time periods, and (ii) the cost of accessing the Internet might deter some participants from fully sharing the idle cycles of their devices. In this section, we present some reference scenarios in which having an ad hoc task distribution method will extend the total resource utilization, enabling more users to contribute their spare cycles.

3.2.1 Server Driven Mobile Computing

Most mobile devices use WiFi as the primary method of accessing the Internet due to the fact that it provides an affordable and fast connection. However, considering a user's typical mobility pattern, mobile devices do not have access to a WiFi access point during a significant portion of the day, such as time spent on transportation (e.g., buses, subways); and shopping centers as well as during time spent in remote areas away from the coverage of WiFi hot spots. Although having a cellular data connection does allow a mobile device to participate in a mobile computing platform such as GEMCloud, the cost associated with this connection type often times deters the end-users from participating.

On the other hand, during periods of non-regular mobility patterns such as vacation times, the connectivity of the devices becomes the bottleneck factor limiting the effective use of mobile computing. Although WiFi is quite widespread, aside from some hotels and coffee shops that offer it as a complementary service, Internet connectivity through WiFi hot spots generally requires a subscription fee. In addition, the fact that the connection to these hot spots is not seamless and often requires the user intervention, further limits the connectivity of mobile devices, thus decreasing their utilization in the mobile computing architecture.

A standard mobile computing system, like GEMCloud and BOINC, would not be able to utilize users who do not have an active data connection or who are not willing to use their data connection. These users would then be forced to wait until they are back "on the grid" to resume their participation.

In order to address these shortcomings, we propose the introduction of two functional roles for each client of the system: (i) task execution points (TEPs), and (ii) task distribution points (TDPs). TEPs are devices that have compatible computational platforms and are willing to participate in the computing platform. TEPs compute the tasks assigned to them and send the results back to the device that requested the computation. TDPs, instead, are responsible for receiving sets of tasks from the cloud and then distributing them to the TEPs through peer to peer connections, e.g., Bluetooth or WiFi-Direct. In our proposal, a TDP could be a dedicated device that only distributes the tasks, or it could be a generous user that is willing to share its Internet connection, established either via a

cellular network or through a WiFi hot spot. It is important to note that these roles can both be taken simultaneously, and in such case, we refer to this client as task distribution and execution point (TDEP). Therefore, rather than being fixed, the TDP and the TEP are roles that can change over time. After the computation, the task results are returned either directly to the server that requested the computation if a preferred connection becomes available, or to a distribution point that will be in charge of sending them back to the requesting server. We note that the results do not have to be returned to the same distribution point that assigned the tasks.

We want to further emphasize that the proposed system could be enhanced with further economical incentives. For example, some reward could be offered to the clients acting as TDP in exchange for distributing the tasks to the TEPs that cannot receive the tasks directly. These economical incentives could form the foundation of a new business model and enhance the efficient use of computational devices. A particular scenario is the utilization of ad hoc mobile computing in public transportation. The introduction of a task distribution point in a bus would allow the passengers with mobile devices to participate in computing. While dedicated TDPs could be installed on public transportation, users who are willing to use their data connection can also take the role of a TDP and connect the mobile devices on the bus to the cloud. In this case, participation could be encouraged with economic incentives such as discounted bus tickets.

3.2.2 User Driven Mobile Computing

Often times the devices running distributed computing algorithms reside outside of traditional networks and away from nodes that are capable of intensive computing. For instance, sparsely populated remote locations oftentimes neither have direct access to the Internet nor are in the vicinity of other devices with access. For example, ad-hoc distributed computing can help many applications in the area of tactical military communications, first responder networks, search and rescue operations, and sensor network operations. In these application areas, there are many computing intensive algorithms ranging from image processing to target detection that can be parallelized. Using a cooperative distributed computational architecture, these algorithms can be computed in a shorter amount

of time using a higher accuracy and with less impact on the energy resources of critical nodes, thereby increasing the network lifetime.

Some central solutions have been proposed for this purpose, in which the computing jobs are offloaded to a central server [47]. However, due to the high latency of the communication architecture, such central solutions are infeasible for jobs with tight bandwidth limitations. One solution includes deploying a *Cloudlet* [48] to help accelerate battlefield applications; however, this approach does not completely eliminate the latency problem and creates a single point of failure where the system could collapse if the *Cloudlet* were to go down. Distributed computing addresses these problems by dividing the jobs and distributing them to the nodes in the network. The results are collected and fused for a final decision. This approach minimizes the latency by decreasing the distance between the point of the job request and the point of computation and eliminates the dependency on a specific node in the network.

In our proposed system, the node that creates the jobs also takes the role of a TDP and distributes the tasks to the nodes in the ad hoc network. This system can be optimized to maximize the computational power available to the nodes in need, and to maximize the lifetime of the network by letting jobs be executed where energy resources are abundant and limiting the energy consumption on bottleneck devices.

Finally, the system can be extended to work on multi-hop networks by letting the recipients of the tasks take the role of a TDP and further distribute the tasks to the nodes around themselves. However, we limit the scope of this chapter and only consider single hop ad hoc networks over which TDPs are responsible for distributing the jobs.

3.3 Background on Device to Device Communication

The distributed computing architecture proposed in this work is based on the availability of Device to Device (D2D) communication, which is an emerging paradigm that addresses end-to-end communication between devices, without any human intervention.

Although many protocols have been proposed over the years, IEEE 802.11 DCF, IEEE 802.11s,

IEEE 802.11z, Zigbee, SMAC, WiFi Direct, and Bluetooth, are a few that we have chosen to list and provide a brief description below. It is also worth mentioning that each of the aforementioned have their individual application areas.

IEEE 802.11 DCF has a basic method that provides ad hoc D2D communication and is widely available. However, it has been shown that this protocol suffers from low performance in real life implementations [49] and has a high energy consumption due to the requirement of maintaining the listening state on the radios. WiFi Direct, on the other hand, has recently been proposed to address the shortcomings of IEEE 802.11 DCF and has been designed with energy saving mechanisms leading to higher energy efficiency. IEEE 802.11s and IEEE 802.11z add mesh networking and direct communication on top of the IEEE 802.11 family. These additions are only relevant in a multi-hop network. SMAC and ZigBee are energy efficient protocols that are designed for sensor networks. The data rates supported by these protocols are very low and thus not suitable for large tasks. Moreover, their availability are limited to low power sensor nodes that have very low computational capacity. Bluetooth is a widely available technology designed to support low power communications between nearby devices (*Bluetooth low energy*), but with the possibility of reaching high data rates (*Bluetooth high speed*) [18].

Given the above, we focus our attention on two D2D protocols, namely WiFi Direct and Bluetooth. This is because we believe that WiFi Direct is the most promising technology, while Bluetooth is an incumbent technology widely deployed. Moreover, both protocols are commonly available off the shelf in several mobile devices. In the remainder of this section, we present a brief overview of these two protocols.

3.3.1 Bluetooth

Bluetooth is a popular standard primarily used for Wireless Personal Area Networks (WPAN) [50]. Bluetooth uses the 2.4 GHz band for transmission, and it uses the Frequency Hop Spread Spectrum (FHSS) technique for channel access. Devices inside a WPAN are divided into *master* and *slaves*, where the master node is essentially a node that is in charge of advertising, authenticating and allowing

other nodes to join the network. Bluetooth authentication primary steps are *Inquiry* and *Paging* [50]. The inquiry procedure is done by sending requests, and responses are sent from nodes that have discovery enabled [50]. This is done completely at the physical layer and it is during this step that QoS commitments are made [50]. Additionally, there is an extended *Inquiry* phase in which information such as data type, local name, and services can be made exchanged between devices [50]. Connection requests are made on a special channel, and each node can only handle one connection request at a time.

The *master* is responsible for coordinating the transmissions inside the group, and each master can manage up to seven slaves. The *master* can transmit during the even slots, and all the *slaves* are allowed to transmit during the odd slots. A message can last for several slots, and during transmission, the hopping sequence does not advance.

3.3.2 WiFi Direct

WiFi Direct [19] is a standard released by the WiFi alliance that enables D2D communication without requiring a wireless access point. During D2D communication, devices forms a group were one of them is the Group Owner (GO) and all the others are considered Group Members (GMs). It is important to note that these roles are not fixed and can change dynamically.

Additionally, WiFi Direct groups can also consist of nodes that do not support WiFi Direct, but do support the IEEE 802.11 standard that the group is operating. WiFi Direct utilizes IEEE 802.11 a/b/g/n infrastructure mode, and can transmit either at 2.4GHz or 5GHz.

Since WiFi Direct utilizes the IEEE 802.11's *infrastructure* mode, all of the QoS, power saving, and security protocols of IEEE 802.11 are also inherited. Similar to a typical IEEE 802.11 network, where nodes find and connect to APs, the GO also acts as a soft AP, advertising and allowing nodes to join the group. Group advertisement is performed using beacon packets, just like a typical IEEE 802.11 AP, and the GO is responsible for giving control of the channel to nodes in its network as well as routing data through its group. Nodes that have WiFi support but do not support WiFi Direct can still connect to a WiFi Direct group [19, 51] and are referred to as *legacy clients*. GMs are the nodes

that support WiFi Direct and hence can capitalize on the WiFi Direct power saving options.

The nodes that support WiFi Direct go through a group formation process in order to determine the roles of the GO and the GMs. There are three group formation processes: standard, persistant and autonomous [19, 51]. In the standard group formation, typically done when nodes meet for the first time, nodes listen on channels 1, 6, and 11 in the 2.4 GHz band [19, 51]. After finding another device, the devices negotiate as to which will act as the *group owner* (GO) [19, 51]. This is done in the handshake process, where the devices exchange an *intent value*, and the device with the highest value become the GO [19, 51]. Persistent group formation occurs if the persistent flag has been set in the first encounter, this allows for the nodes to take up the roles that they had previously [19, 51]. This method also enables a quicker handshake, allowing the devices to skip Phase 1 of WiFi Protected Setup (WPS) [19, 51]. In autonomous group formation, a node assigns itself the role of GO and creates its own group [19, 51].

After finding a group, by responding to probe requests, or after the negotiation phase of group formation, nodes will establish a secure connection using WPS and receive an IP address using Dynamic Host Configuration Protocol (DHCP) exchange [19, 51].

3.4 System Architecture

The traditional mobile computing platform, as considered in [20, 46, 47], is presented in Figure 3.2.

Our goal is to extend the range of computation devices of a traditional mobile computing system via ad hoc communications. In Figure 3.3, we show an example of this type of system, where certain users are allowed to become local Task Distribution Points (TDPs and TDEPs in Figure 3.3) and provide access to the distributed computing system to other clients or task execution points, using an ad hoc communication method (e.g., D2D communication). Thus, each TDP is in charge of requesting tasks from the server, and then it distributes these tasks to its clients. Moreover, the TDPs are responsible for organizing the results, resolving any failures, and returning the results back to the remote server.

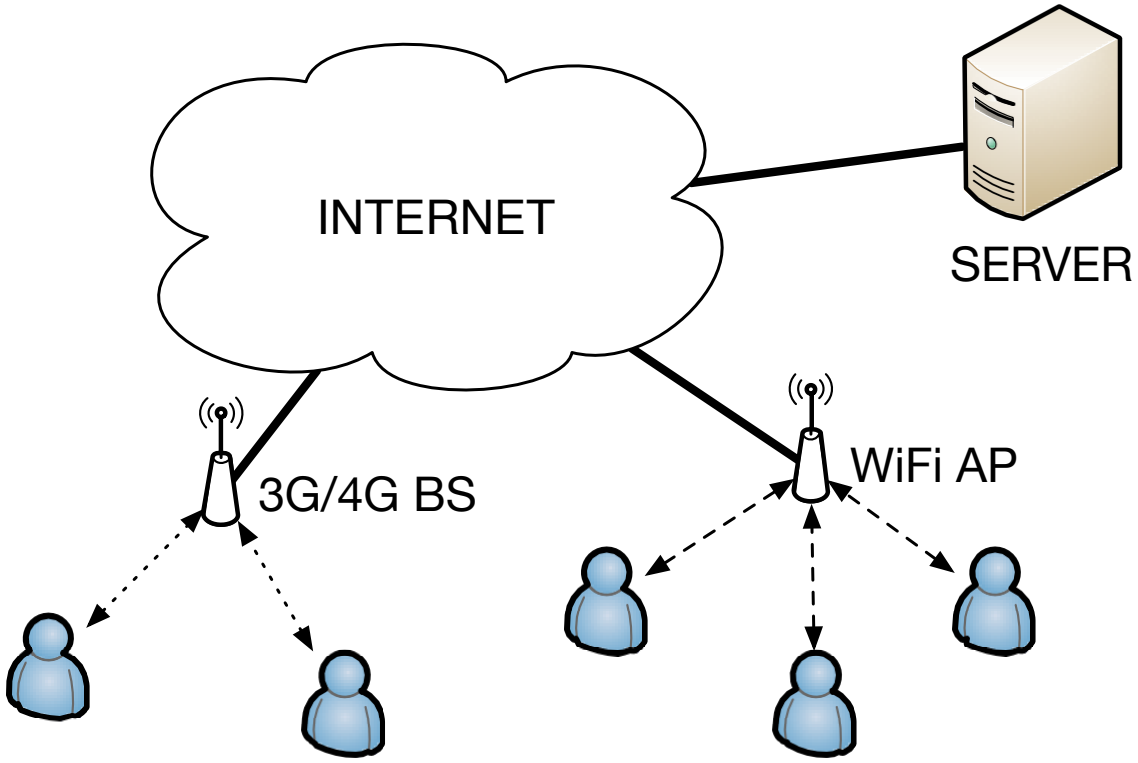


Figure 3.2: Example of a traditional mobile volunteer computing architecture where users are connected to the remote server through a WiFi Access Point (AP) or a 3G/4G Base Station (BS).

The general idea behind our approach is similar to the one proposed in [23], that considers a manager/worker model, in which the manager receives requests from the workers and responds to these requests by assigning tasks. The system presented in [23] considers the manager to be a remote server that the workers (either traditional computers or mobile devices) can reach through the Internet.

3.4.1 Task Distribution Point

In our system, a central role is played by the devices that act as TDPs and, for this reason, the way in which the TDP operates will affect the performance of the complete system. In particular, the TDP is required to receive tasks from a remote server, distribute the tasks to the TEPs through D2D communication, receive the results of the computations from the TEPs and then send the results back to the remote server.

The TDPs can receive the tasks from the server following different approaches. In general, the different methods can be classified based on when the actual request happens: 1) following a proactive

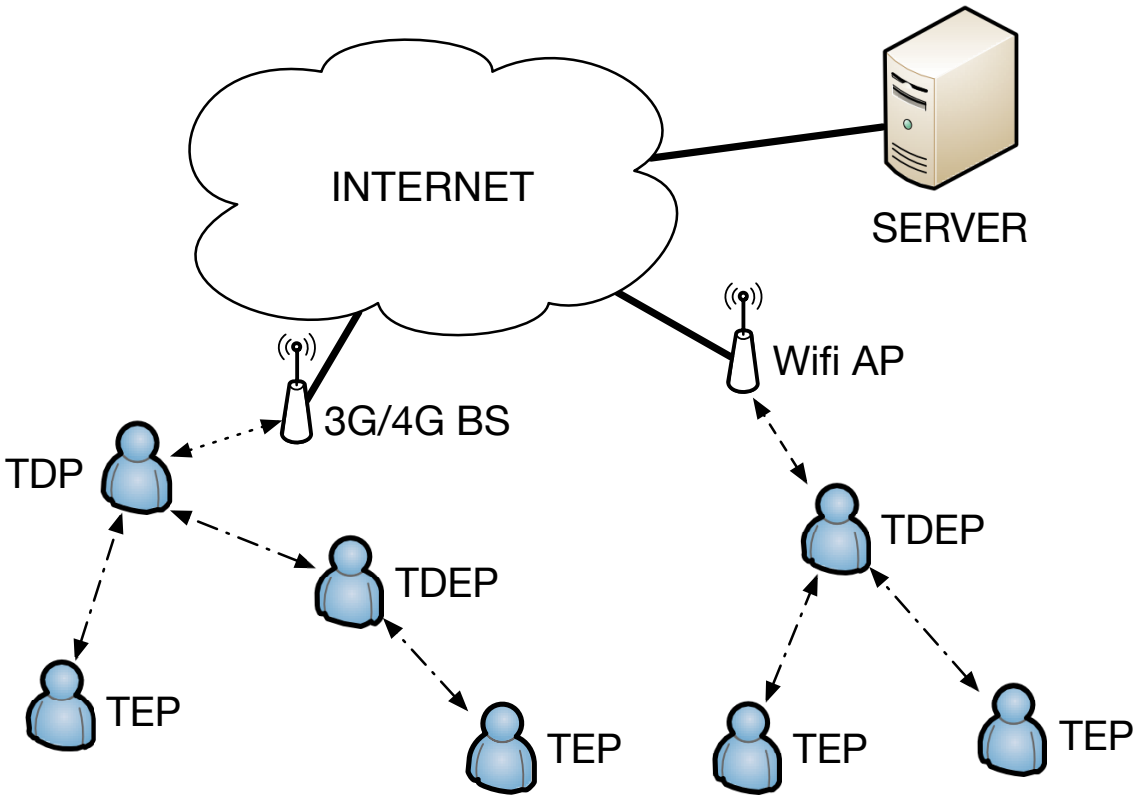
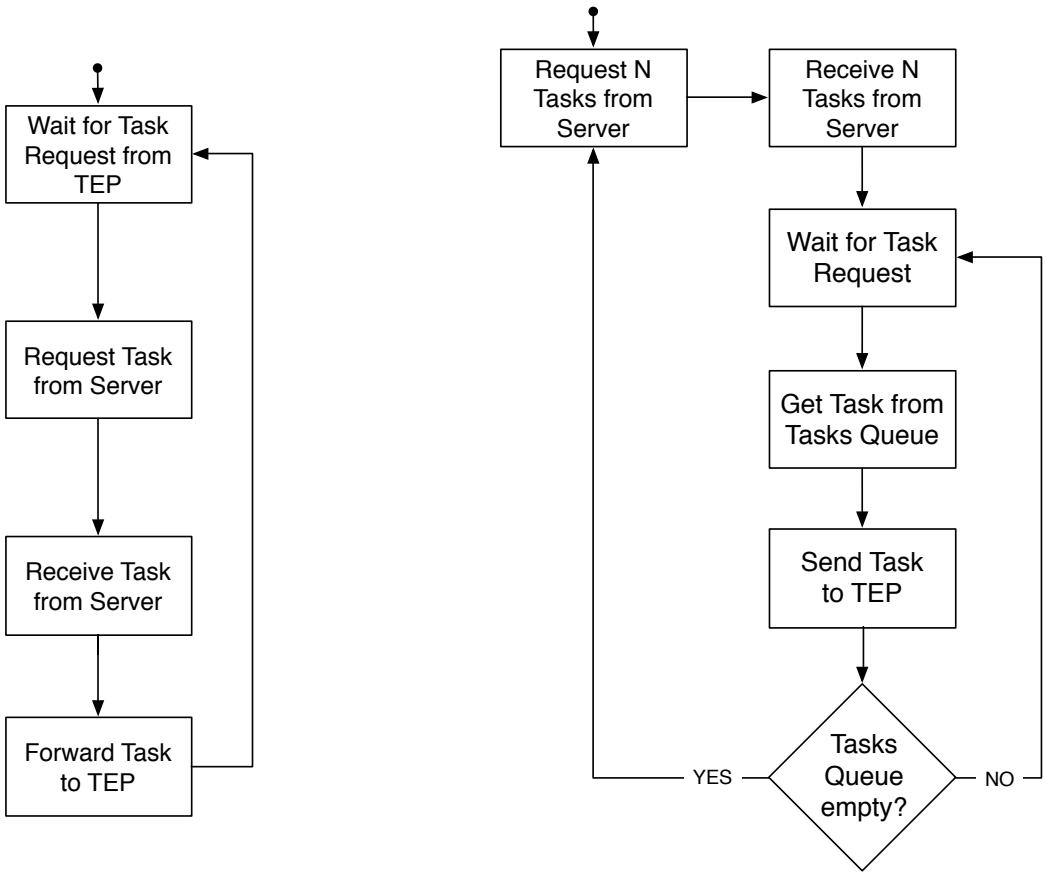


Figure 3.3: Example of ad hoc mobile computing topology with corresponding functional role assigned to each of the clients. Task distribution point (TDP), task execution point (TEP), and task distribution and execution point (TDEP).



(a) (b)

Figure 3.4: Task distribution flowcharts for the TDP operating according to the *Proxy* (a) and *Batch* (b) methods.

approach, meaning that the task requests to the server are made before the actual requests from the TEPs, or 2) following a reactive approach, thus requesting the task from the server at the time of the actual TEP requests. Following a similar idea, the TDP can send the results back to the remote server either immediately, at the time of the reception from the TEP, or delayed, after collecting multiple results. To this end, we propose two different TDP modes of operation: a simple *Proxy* method, that considers a reactive task distribution technique and an immediate forwarding of the results, and a more involved *Batch* approach that, instead, implements a combination of proactive task requests with a delayed results transmission.

Proxy: This method represents a basic mode of operation and requires little intelligence to be added to the device that will act as TDP. In particular, a *Proxy* TDP acts as a gateway and forwards all requests and responses directly between the remote server and its clients. Figure 3.4(a) shows the

flow diagram of the *Proxy* task distribution procedures, while Figure 3.5(a) presents the flow diagram for the collection of the results.

Batch: In this method, the TDP proactively requests a set of N tasks from the server and caches them so that, when it receives a request from the TEPs, the TDP will promptly respond with one of the cached tasks. After completing the task, the TEP returns the result to the TDP, and the TDP stores these results before sending them back to the server. Figure 3.4(b) shows the flow diagram of the *Batch* task distribution procedures, while Figure 3.5(b) presents the flow diagram for the collection of the results.

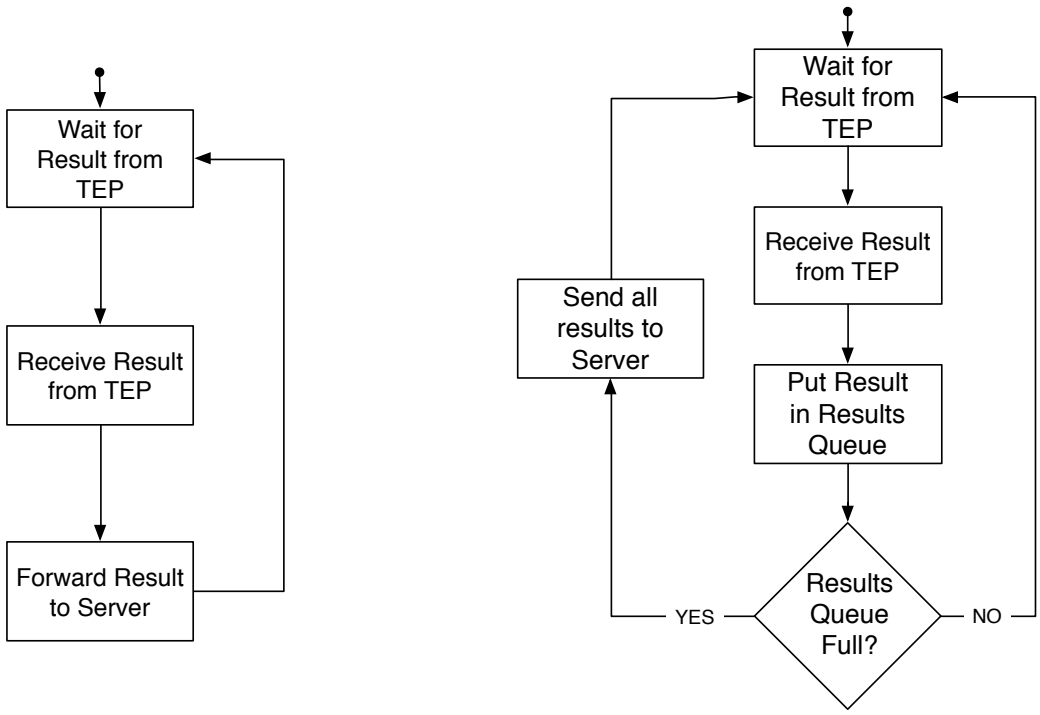
3.4.2 Task Execution Point

In the proposed system, we assign the role of task execution point (TEP) to all the devices that participate in the distributed computation, regardless of the connection used for receiving the task and sending the results of the computation. Thus, a traditional client of the mobile computing system that receives the tasks directly from the server and a client connected through D2D communications are both considered TEPs. Moreover, the way in which the TEPs receive tasks and send results back to the server are completely transparent, and the protocol used for the communication is exactly the same in both cases.

3.4.3 Complexity Considerations

The objective of our work is to extend the mobile computing architectures by introducing local TDPs, whose operation is completely transparent to both a standard client (i.e., TEP) and the remote server. Thus, the additional complexity of our system when compared to an existing mobile computing architecture, resides on the requirements of the D2D communication technology, on the ability to advertise and discover local TDPs, and on the intelligence necessary for the local task distribution procedure.

In particular, the *Proxy* implementation simply forwards all the requests to and from the server to the ad hoc network clients. Thus, a *Proxy* TDP acts as a local gateway, and it performs a translation



(a) (b)

Figure 3.5: Result collection flowcharts for the TDP operating according to the *Proxy* (a) and *Batch* (b) methods.

between the Internet interface and the D2D domain. We consider the *Proxy* method to be the simplest mode of operation that can be implemented in a TDP. It simply forwards every communication to and from the server, thus introducing additional delay due to the additional communication hop. However, it is considered as a reference implementation for evaluating the performance of other task distribution techniques. The *Batch* task distribution approach, instead, overcomes the inefficiency in the network utilization of the *Proxy* method by caching a set of N tasks at the TDPs for faster distribution to the local TEPs. Moreover, a *Batch* TDP stores a set of M results before sending them to the server. Thus, some additional computation and data storage is required in order to handle the task requests and the caching operations.

In section 5.6.1 we provide some experimental results that show the impact in terms of time delay and energy consumption of the proposed task distribution methods.

3.4.4 Implementation

Given that the idea behind our work is to enhance the performance of a mobile computing system, we extend the GEMCloud [46] architecture to support TDPs that can distribute tasks to TEPs connected through D2D communication. GEMCloud provides us with a platform, and an Android application, to evaluate the benefits of extending volunteer computing through ad hoc networking.

In GEMCloud, the server acts as a central point by coordinating and distributing tasks to the connected clients. The client connects to the server using either 3G/4G or WiFi. After connecting, the server checks to see if the client requires an updated version of GEMCloud [46]. The server then authenticates the client using a set of parameters that are exchanged and, after that, it assigns tasks to the client. Due to the unpredictability of the Internet, the typical job executed in an architecture like GEMCloud are termed "embarrassingly parallel problems," where there exists no dependency (or communication) between the parallel tasks. Thus, the tasks are stored in a list, and they are assigned in a first in first out order. Each of these tasks are independent and do not need to be all returned to the server for the overall job to complete. However, the quality of the final results will depend on the number of tasks that are actually returned after the distributed computation.

In our implementation of the *Proxy* and *Batch* distribution methods, the server is the same as in GEMCloud, and the client takes the role of TEP. Using our system, we can extend the devices that can participate in the GEMCloud computations by connecting additional TEPs through the TDPs. The *Proxy* task distribution method provides operation very close to GEMCloud: the TDP behaves as a proxy, and it forwards any requests between the TEP and the server. In the *Batch* implementation, instead, the TDP requests and caches a set of tasks from the server. The TDP still goes through the same process as in GEMCloud, where the server checks for updates and authenticates before sending a task; however, the TEPs that connect to a TDP running our *Batch* implementation will have their version checked against the local version that the TDP is running. The TDP then will send tasks from its tasks queue. It is important to note that in both task distribution methods, in addition to the role of a TDP, the device may also act as a TEP. In the *Proxy* method, devices that serve as TDPs connect and request tasks from the server also for themselves, similar to the original GEMCloud operation. In the

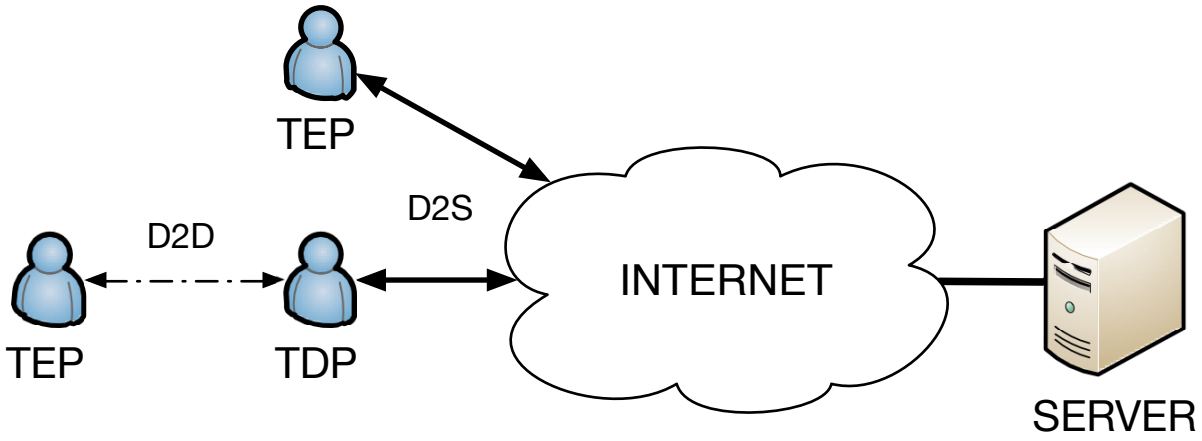


Figure 3.6: Communication links between TEP, TDP and remote server.

Batch method, devices acting as TDPs take tasks from their own queue rather than requesting them from the server. In other words, these devices can be modeled by combining the TDP and TEP roles, and we note these devices as TDEPs.

3.5 Analytical Model

Our system is composed of four elements, namely TEP, TDP, server and database. Since we assume that the TDP-TEP interactions are transparent to the remote server and database, and that the remote server and database have the same operations as in traditional distributed computing systems, in our performance evaluations we focus on the performance of the TDPs and the TEPs.

For the purpose of deriving an analytical model of the system under consideration, in what follows we refer to the scenario presented in Figure 3.6, where we label the communication link between the TDP and the TEPs as D2D, and the communication link between a device, acting either as a TDP or TEP, and the remote server as D2S. We note that the communication link determines the communication technology that can be used for the communication. In particular, this work considers the communication over the D2D links, we analyze the impact of WiFi Direct and Bluetooth, while for the D2S link that requires Internet connectivity, we consider WiFi and LTE[1].

As described in Section 3.4, both the TEP and the TDP require the transmission and reception of

[1]Our experimental results have been obtained using a device that supports WiFi, WiFi Direct and Bluetooth. The results for LTE are obtained through the analytical model presented in [52].

the tasks and results of the task execution over the D2D or D2S link, thus entailing communication energy consumption. Additionally, the TEP needs to execute the task, which will require computational energy consumption, while the TDP is required to manage the task distribution process and wait and reply to requests from the TEPs. Finally, an additional energy overhead due to standard operating system operations is consumed at both the TEP and TDP. The details of the different components are presented in the following sections and are integrated with the experimental results of Section 5.6.1 for evaluating the performance of our system.

3.5.1 Communication Energy Model

The communication model follows a standard framework, e.g., [52, 53], that relates the power consumption of different radio technologies to the mode of operation of the radio technology. While different communication states can be identified for each short range technology, such as WiFi, WiFi Direct and Bluetooth, in this chapter our interest is on the energy spent by the device in the transmission, reception and idle phases. This is particularly reasonable for the TEP participating in our mobile computing system, since we assume that each TEP participates in the distributed computation for several tasks, thus making the impact of the other states, like connection to the Access Point or Bluetooth discovery phase, a one time energy consumption with negligible impact on the steady state operation of the system. We note that the TDP, instead, needs to continuously advertise its presence to the incoming TEPs, and invite and accept connections through D2D links. However, for simplicity we incorporate the energy consumptions for these operations into the energy consumption of the task distribution process.

Given the above, we define P_{tx}^{X} and P_{rx}^{X}, to be the transmission and reception power consumption of technology X, respectively, and P_{idle}^{X} as the relative idle power consumption. Moreover, we refer to the transmission and reception throughput of each technology as T_{tx}^{X} and T_{rx}^{X}. Since the throughput on wireless networks fluctuates due to the signal strength and channel impairments, assigning values to them only serves as an example of the achievable performance of a network. However, [52, 54], as well as our measurements presented in Section 5.6.1, show a constant transmission (and reception)

energy consumption per bit, for the transmission/reception of a fixed data size. Moreover, both our results and [52] show an inverse relationship between the energy per bit and the data size (see Fig. 3.7). As a result, we define the energy required for sending s bits of information using technology $X \in \{\text{WiFi Direct, Bluetooth, WiFi, LTE}\}$ as

$$E_{\text{tx}}^{X}(s) = \alpha_{\text{tx}}^{X}(s)s, \tag{3.1}$$

where $\alpha_{\text{tx}}^{X}(s) = \text{f}\left(P_{\text{tx}}^{X}, T_{\text{tx}}^{X}, s\right)$ represents the energy consumption per transmitted bit, and $\text{f}(\cdot)$ is a function that depends on the particular radio chipset. Similarly, we define the energy required for receiving s bits as

$$E_{\text{rx}}^{X}(s) = \alpha_{\text{rx}}^{X}(s)s, \tag{3.2}$$

where $\alpha_{\text{rx}}^{X}(s) = \text{g}\left(P_{\text{rx}}^{X}, T_{\text{rx}}^{X}, s\right)$ represents the energy consumption per received bit, and $\text{g}(\cdot)$ is a function that depends on the particular radio chipset.

3.5.2 Computation Energy Model

As the name suggests, the main components of a distributed computing system are represented by the execution of the tasks assigned to the client. Thus, when modeling the energy consumption of a mobile computing architecture, the energy requirements of the computation are the most important and predominant part. To this end, we define the energy consumption per task execution as

$$E_{\text{ex}}(t_{\text{ex}}) = t_{\text{ex}}P_{\text{ex}}, \tag{3.3}$$

where P_{ex} represents the power consumption of the mobile device during task computation, while t_{ex} represents the time required to execute the assigned task. We note that the time t_{ex} depends on both the CPU characteristics of the mobile device and on the complexity of the task to be executed. While a detailed evaluation of the relationship between device process capability and task complexity is out of the scope of this chapter, we consider the ratio between the task total number of floating-point operations and the CPU FLOPS (FLoating-point Operations Per Seconds) to be a reasonable lower

bound (alternatively number of instructions / CPU Instructions per second, IPS).

3.5.3 Task Distribution Energy Model

In order to completely characterize the energy consumption of our proposed architecture, we need to define the energy overhead introduced by the task distribution process. It can be noticed that while the energy consumption of the different communication technologies are defined by the technology itself, the overhead introduced by the task distribution depends on the particular authentication and communication protocol used to route the tasks and results between the D2D and the D2S links. In Section 3.4.1 we described two different task distribution algorithms, *Proxy* and *Batch*, that rely on a different communication scheme.

We define $P_{\text{td,proxy}}^{D2D,D2S}(k)$ and $P_{\text{td,batch}}^{D2D,D2S}(k)$, to be the power requirement to handle the *Proxy* and *Batch* task distribution processes, respectively, when the TDP is not transmitting nor receiving data through the D2D and D2S links and it is serving k different TEPs. As stated earlier, we consider that $P_{\text{td,batch}}^{D2D,D2S}(k)$ and $P_{\text{td,proxy}}^{D2D,D2S}(k)$ include the power required to maintain the D2D connections with the associated TEPs, as well as the advertising of the link availability and the eventual invitation and connection of new TEPs. The energy consumption of the *Proxy* task distribution process is thus represented by

$$E_{\text{td,proxy}}^{D2D,D2S}(i,k) = iP_{\text{td,proxy}}^{D2D,D2S}(k), \tag{3.4}$$

where i represents the time in which the TDP is not serving any of its TEPs, nor communicating with the server. Similarly, for the *Batch* task distribution method we have

$$E_{\text{td,batch}}^{D2D,D2S}(i,k) = iP_{\text{td,batch}}^{D2D,D2S}(k). \tag{3.5}$$

3.5.4 TEP and TDP Energy Model

In this section, we combine the different energy models in order to derive the total energy consumption of a mobile device operating either as a TEP or TDP. Moreover, we consider the energy model of a TDEP, i.e., a device that simultaneously executes tasks and distributes tasks to others, to

be the superposition of the TEP and TDP energy models.

For the communication between the TDP and the server, and between the TEP and the TDP we assume the existence of a simple authentication protocol that requires the exchange of additional information before the actual data exchange[2]. By considering that this authentication process requires the transmission of a^X_{client} bits from the device acting as a client (this includes also the information required to request a task or to prepare the server for the reception of the result) and the transmission of a^X_{server} bits from the server (or the TDP for local D2D task distribution), we define the client authentication energy as

$$E^X_{\text{auth,C}} = E^X_{\text{tx}}(a^X_{\text{client}}) + E^X_{\text{rx}}(a^X_{\text{server}}), \tag{3.6}$$

where $X \in \{\text{WiFi Direct, Bluetooth, WiFi, LTE}\}$, and the TDP authentication energy when acting as a server for the TEPs as

$$E^{D2D}_{\text{auth,S}} = E^{D2D}_{\text{tx}}(a^{D2D}_{\text{server}}) + E^{D2D}_{\text{rx}}(a^{D2D}_{\text{client}}), \tag{3.7}$$

with $D2D \in \{\text{WiFi Direct, Bluetooth}\}$. We note that the authentication with the server can, in general, be different than the authentication method used for the D2D network, and that the authentication method can also depend on the particular radio technology used for the communication.

In order to derive the energy of the different devices participating in the mobile computing system we additionally need to define the system parameters. To this end, we consider that assigning a task to a TEP requires the transmission/reception of t bits, the computation associated with the task has a complexity of c seconds, and the result of the task execution to be reported back to the server entails the transmission of r bits. Moreover, for the TDP energy model, we define k to be the number of TEPs connected to the TDP. Given the above, the total energy consumption per task for a generic TEP is

$$E^X_{\text{TEP}}(t, c, r) = E^X_{\text{rx}}(t) + E_{\text{ex}}(c) + P^X_{\text{idle}}c + E^X_{\text{tx}}(r) + 2E^X_{\text{auth,C}}, \tag{3.8}$$

where $X \in \{\text{WiFi Direct, Bluetooth, WiFi, LTE}\}$ represents the communication technology used for receiving the task and sending back the result after the relative execution. The total TDP energy

[2]The extension to a more complicated authentication process that requires local computation can be easily be included in the model by adding an additional computational cost.

consumption for handling a single task to be executed at one of the TEPs is, instead:

$$E_{\mathrm{TDP}}^{D2D,D2S}(t,c,r,k) = E_{\mathrm{rx}}^{D2S}(t) + E_{\mathrm{tx}}^{D2S}(r) + E_{\mathrm{tx}}^{D2D}(t)$$
$$+E_{\mathrm{rx}}^{D2D}(r) + E_{\mathrm{td}}^{D2D,D2S}\left(\tfrac{c}{k},k\right) + 2(E_{\mathrm{auth,C}}^{D2S} + E_{\mathrm{auth,S}}^{D2D}), \tag{3.9}$$

where $D2D \in \{\text{WiFi Direct, Bluetooth}\}$ and $D2S \in \{\text{WiFi, LTE}\}$.

3.5.5 Total Task Time Model

The main focus of our analysis is to evaluate the impact on the energy consumption of extending mobile computing through D2D communications. However, in certain time sensitive applications, the total time to complete a set of tasks is also an important parameter to consider. In what follows, we consider the total time required to compute a task at the TEP as the time between the task request and the end of the transmission of the result of the task execution. As stated in Section 3.5.2, the time required to execute a single task depends on the CPU of the mobile device and on the complexity of the task to be executed, which, without loss of generality, can be considered a fixed parameter. Therefore, the total time required to compute a single task depends on the radio technology used for the data exchange and, for the extension with D2D communications, on the task distribution algorithm implemented at the TDP.

For a TEP directly connected to the remote server, or for a TEP connect to a TDP *Batch*, the total delay experienced for each task is given by

$$D^X(t,c,r) = \frac{t}{T_{\mathrm{rx}}^X} + \frac{r}{T_{\mathrm{tx}}^X} + 2D_{\mathrm{auth}}^X + c, \tag{3.10}$$

where $D_{\mathrm{auth}}^X = a_{\mathrm{client}}^X/T_{\mathrm{tx}}^X + a_{\mathrm{server}}^X/T_{\mathrm{rx}}^X$. For a TEP connected to a TDP operating according to the *Proxy* task distribution method, instead, the total delay between the task request and the delivery of

the task result is given by

$$D_{\text{proxy}}^{D2D,D2S}(t,c,r) = \frac{t}{T_{\text{rx}}^{D2D}} + \frac{t}{T_{\text{rx}}^{D2S}} + \frac{r}{T_{\text{tx}}^{D2D}} + \frac{r}{T_{\text{tx}}^{D2S}} + 2\left(D_{\text{auth}}^{D2S} + D_{\text{auth}}^{D2D}\right) + c, \tag{3.11}$$

where D_{auth}^{D2D} and D_{auth}^{D2S} are derived from D_{auth}^{X}. It can be noticed that the tasks assigned to a TEP connected to a TDP *Proxy* experience an increased delay due to the additional communication hop. This is because according to the *Proxy* task distribution, all the communication to and from the TEP are forwarded to the remote server. This is not the case for a TEP connected to a TDP *Batch*, since according to the *Batch* task distribution algorithm, both the tasks and the results are cached at the TDP in order to minimize the delay experienced by the TEPs.

Given the above, from the server perspective, the total time required with a single TEP to compute a set of J tasks, each one of data size t, complexity c and that generates a result data size r, is given by $J \times D^{D2S}(t,c,r)$ for a standard TEP, $J \times D_{\text{proxy}}^{D2D,D2S}(t,c,r)$ for a TEP connected to a TDP *Proxy* and $J \times D^{D2D}(t,c,r) + N\frac{t}{T_{\text{rx}}^{D2S}} + M\frac{r}{T_{\text{tx}}^{D2S}} + (N+M)D_{\text{auth}}^{D2S}$, for a TEP connected to a TDP *Batch*[3].

We note that a substantial contribution on the timing derived in this section is represented by the throughout achievable by the different radio technologies. Since the achievable throughput depends on the instantaneous channel conditions and thus varies over time, the results obtained through the model presented here represents a bound on the actual time required for completing the tasks.

3.5.6 TEP Modeling

The TEP dynamics are captured by a continuos-time Markov chain with 4 states $X_{\text{TEP}} \in \mathcal{S} = \{\textit{Idle}, \textit{Task}, \textit{Computation}, \textit{Result}\}$, where *Idle* refers to the state when the TEP is neither communicating nor computing, *Task* refers to the state where the device requests and then receives a task either from the remote server or from a local TDP, *Computation* refers to the state where the device is computing the task and *Result* refers to the state where the TEP sends the result of the

[3]The extension to multiple TEPs and non homogeneous tasks (i.e., having different values of t, c and r) is straightforward.

computation back to the task distributor. We refer to T_j, with $j \geq 0$, as the time instant at which the TEP transitions between states and $\Delta_j = T_j - T_{j-1}$ as the time elapsed between two subsequent transitions. Between T_{j-1} and T_j, the TEP is said to be in stage j, and its duration Δ_j is described by a random variable $\tau_{X_{\text{TEP}}} \in [T_{\min}(X_{\text{TEP}}), T_{\max}(X_{\text{TEP}})]$, depending on the TEP state X_{TEP} during stage j. $\tau_{X_{\text{TEP}}}$ has an associated probability distribution function (pdf) $f_\tau(T|X_{\text{TEP}})$, which determines the distribution of the duration Δ_j within the relative range. Moreover, during stage j, the TEP consumes a power P_j, which depends on the state X_{TEP}, and is assumed to remain constant until the next transition, occurring at time T_j. This power consumption is described by the random variable $\mathcal{P}_{X_{\text{TEP}}} \in [P_{\min}(X_{\text{TEP}}), P_{\max}(X_{\text{TEP}})]$, with pdf $f_{\mathcal{P}}(P|X_{\text{TEP}})$. The probabilities $p_{h,k} = \text{Prob}\{X_{\text{TEP}}(j) = k|X_{\text{TEP}}(j-1) = h\}$, with $h, k \in \mathcal{S}$ are the associated embedded Markov chain transition probabilities, are invariant with respect to j, and are defined as $p_{\textit{Idle,Task}} = p_{\textit{Task,Computation}} = p_{\textit{Computation,Result}} = p_{\textit{Result,Idle}} = 1$, and equal to 0 otherwise.

3.5.7 Task Distribution Considerations

The different components of the continuous-time Markov chain described in Section 3.5.6 can be obtained by carefully determining the different components of the analytical model described in the previous sections. In this regard, $\Delta_j P_j$ represents the TEP energy consumption at stage j, while $\sum_{j=i}^{i+4} \Delta_j$, with $i = \textit{Idle}$, represents the total delay of a particular task. This model can be used to finely profile the behavior of the different TEPs present in the network (this includes both the mobile device intrinsic characteristics and the user behavior) in order to optimize the system performance. In particular, the remote server or the local TDP can use this Markov chain model for finding the best task assignment for a particular TEP. Finally, we note that this particular formulation is useful for improving the task distribution process when either the devices or the task to be distributed are heterogeneous. In case of homogeneous tasks and TEPs, instead, performance gains can only be achieved by efficiently choosing between the available communication technologies (e.g., switching between Bluetooth and WiFi Direct) and by reducing the time required for assigning the tasks and transmitting back the results by locally batching the tasks and results, respectively.

3.6 Experimental Results

Unlike [6], we have developed our own experimental platform. We argue that our method used to measure the energy is more accurate to the scenario where D2D communication would be used, since the method used in [6] measures the energy drawn from the electrical outlet. We provide a description of our experimental setup below.

Our system was implemented using seven second generation Asus Nexus 7s (N7), which were released in 2013. The 2013 N7 has 16 GB of storage, 2 GB of memory, and a 1.5GHz quad-core Snapdragon S4 Pro 8064 CPU produced by Qualcomm [55]. Each of our devices are running Google's Android 4.4 operating system.

3.6.1 Test Environment

Similar to the work in [54], we also measured the current from the battery for each of our experiments using an Arduino Uno [56]. This was done by placing a $.005\Omega \pm 1\%$ resistor in series with the battery and measuring the voltage across this resistor to obtain the current. However, the voltage drop across this resistor was too small to be read by the Arduino Uno [56]. Thus, we configured an op-amp to act as a non-inverting amplifier with a gain of 977. This allowed the Arduino's analog input to read the voltage across the resistor throughout the tests. Additionally, due to the non-linear properties of lithium-ion batteries [57, 58], like the ones used in the Nexus 7, we restricted our experimental measurements to battery level above 70%. This is because, as shown in [58], for battery levels between 60% and 100%, the internal impedance of lithium-ion batteries scale almost linearly with regards to the state of charge, while below 40% the behavior is highly non-linear, thus providing a non negligible additional energy consumption to our experiments.

All of the measurements were taken in an indoor workspace, and the nodes were stationary during these experiments. We acknowledge that in real life, nodes generally are mobile, and may enter or leave a network, but to determine the impact that D2D communication has on traditional mobile computation offloading architectures, we kept the nodes stationary in our experiments. Moreover, all the experimental results presented in this section have been obtained over 50 runs of the same

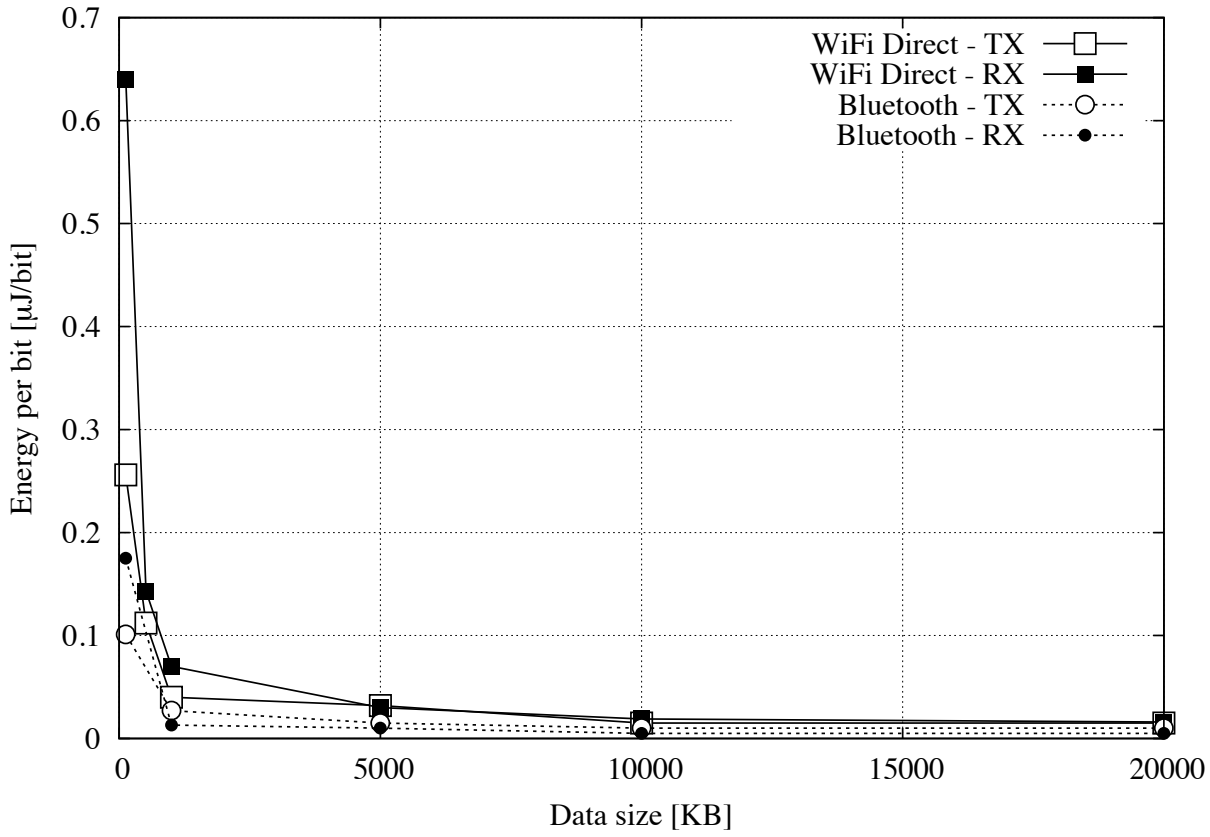


Figure 3.7: Experimental Measurements. Energy per bit for different transmitted (and received) data sizes for WiFi Direct and Bluetooth.

experiment, with the device screen turned off.

3.6.2 Results

To measure the impact that D2D communication has on our system, we performed isolated tests to gauge the impact that WiFi Direct and Bluetooth have on the mobile devices. By using Iperf [59], we were able to send various amounts of data between the WiFi Direct Group Owner (Bluetooth master) and the WiFi Direct Group Member (Bluetooth slave), and we measured the relative power consumptions as well as the throughput achieved for different tests. In order to remove any additional operating system related energy consumption, we were able to use *"adb shell"* to gain a shell session on the N7. From there, we were able to execute a one line script that would run, as a background process, several Iperf TCP tests.

Figure 3.7 shows the measured energy per bit as a function of the transmitted (and received) data size. As previously shown in [52] for WiFi, 3G and LTE, the energy per bit decreases as the data size transmitted, and received, increases, mainly because for small values of data size the achieved throughput is low with respect to the link capacity, due to the TCP slow start mechanism. It is important to note that while the N7 supports Bluetooth 4.0, we were only able to achieve the same performance as Bluetooth Enhanced Data Rate (EDR). We speculate that this is due to the fact that the Google Android APIs do not allow for high speed (HS) operation, since data rates up to 24 Mbps are achieved by utilizing the 802.11 Alternate MAC and PHYs (AMPs) [18]. Following a similar approach as the one in [52], we also evaluated the impact of a controlled transmission throughput on the energy consumption of WiFi Direct and Bluetooth by running different UDP tests with fixed data size through Iperf. Results show an (almost) linear relationship between transmission and reception power consumption and end-to-end throughput.

After evaluating the performance of the considered D2D communication technologies, we measured the energy consumed by a TEP connected to a TDP by either WiFi Direct or Bluetooth, and compared the relative energies with the energy consumed by a TEP connected to a wireless AP. Moreover, we measured the energy consumption at the TDP when serving multiple TEPs simultaneously. For all the considered scenarios, we measured the time and the energy consumption required for the system, a single TEP connected directly to the Internet or one TDP operating according to the *Batch* and *Proxy* task distribution processes and serving 1 or more TEPs, to compute a given set of tasks.

In Figure 3.8 we present the performance, in terms of both energy and time, required for a system with 1, 2 and 6 TEPs to compute a set of 50 tasks, when only one device is able to connect to the remote server. As expected, the overall time required for the computation of the tasks decreases significantly as the number of TEPs that are allowed to participate in the computation increases, since the system can take advantage of a greater degree of parallelism. It is also interesting to note that the impact in terms of overall system energy is small when allowing for a device to act as TDP and to distribute the tasks via D2D communication. Moreover, the energy overhead with respect to a single TEP connected to the Internet decreases as the number of TEPs increases. This is because, while the energy consumption of a local TEP (i.e., without Internet connectivity) is comparable to the energy

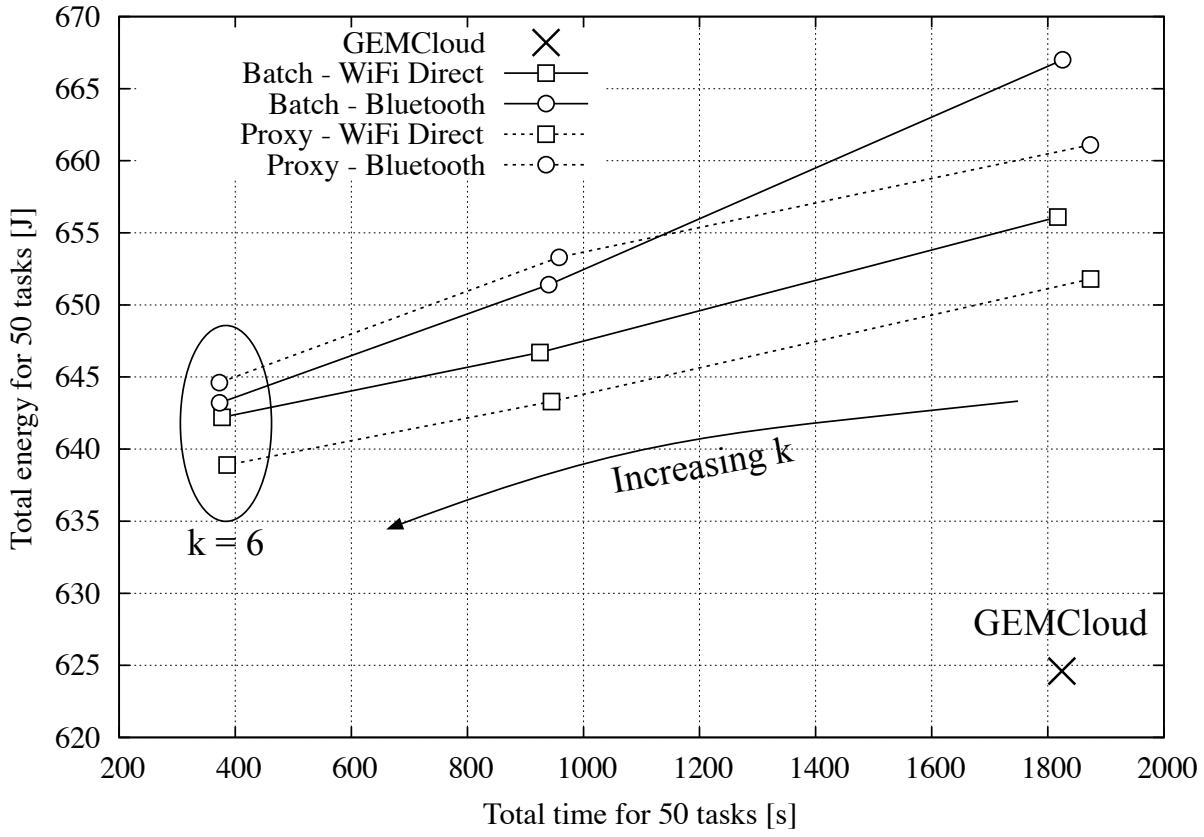


Figure 3.8: Experimental Measurements. Total energy consumption vs. total time for computing 50 tasks, when $k \in \{1, 2, 6\}$ nodes are allowed to execute the tasks. For the proxy and batch methods, the case refers to a system where the TEPs are connected to a single TDP.

	WiFi	WiFi Direct	Bluetooth
P_{idle} [mW]	20	20	18
T_{tx} [MBits/s]	20	40	3
T_{rx} [MBits/s]	20	40	3

Table 3.1: System parameters for the analytical model.

of a standard TEP, the energy spent at the TDP decreases with increasing TEPs due to a reduction in the total operation time of the device. In fact, our measurements show that the number of TEPs has a negligible impact on the average power consumption at the TDP which, combined with the decreasing operational time, lowers the total energy consumption of the device.

In addition, we ran several experiments in order to determine the impact of the different components, as presented in Section 5.3, on the distributed computing system. This allows us to combine our experimental measurements for the different radio technologies presented in Table 3.1 with the analytical model presented in Section 5.3. In addition, for the computation power consumption, instead, we measured an average of $P_{\text{ex}} = 335$ mW, $P_{\text{td,proxy}}^{Bluetooth,WiFi}(k) \approx P_{\text{td,batch}}^{Bluetooth,WiFi}(k) \approx P_{\text{idle}}^{Bluetooth} + P_{\text{idle}}^{WiFi}$ and $P_{\text{td,proxy}}^{WD,WiFi}(k) \approx P_{\text{td,batch}}^{WD,WiFi}(k) \approx P_{\text{idle}}^{WiFi}$ for $k \in \{1, 2, 6\}$. We note that this results in a very low average TDP idle power consumption that range between 20 mW and 38 mW, which is achieved by the combination of the high energy efficiency of the radio components and the duty cycle imposed by the communication protocol (i.e., alternation between periodic beaconing and sleeping). Moreover, our findings suggest that the simultaneous operation of WiFi and WiFi Direct does not entail a significant additional energy consumption, since both protocols share the same physical radio device. These results are in line with previous investigation like, e.g., [54, 60].

In what follows, using the analytical model presented earlier, we evaluate the performance of the proposed system under different settings. In particular, we also include an evaluation of the energy and time required for a system to compute 50 tasks, when an LTE connection to the Internet is available (the LTE parameters considered here are taken from [52]).

We first validate the proposed analytical model by comparing our experimental results with the relative results that can be obtained using the analytical model. To this end, in Figure 3.9 we compare the experimental results presented in Figure 3.8 with the results provided, for the same scenarios, by

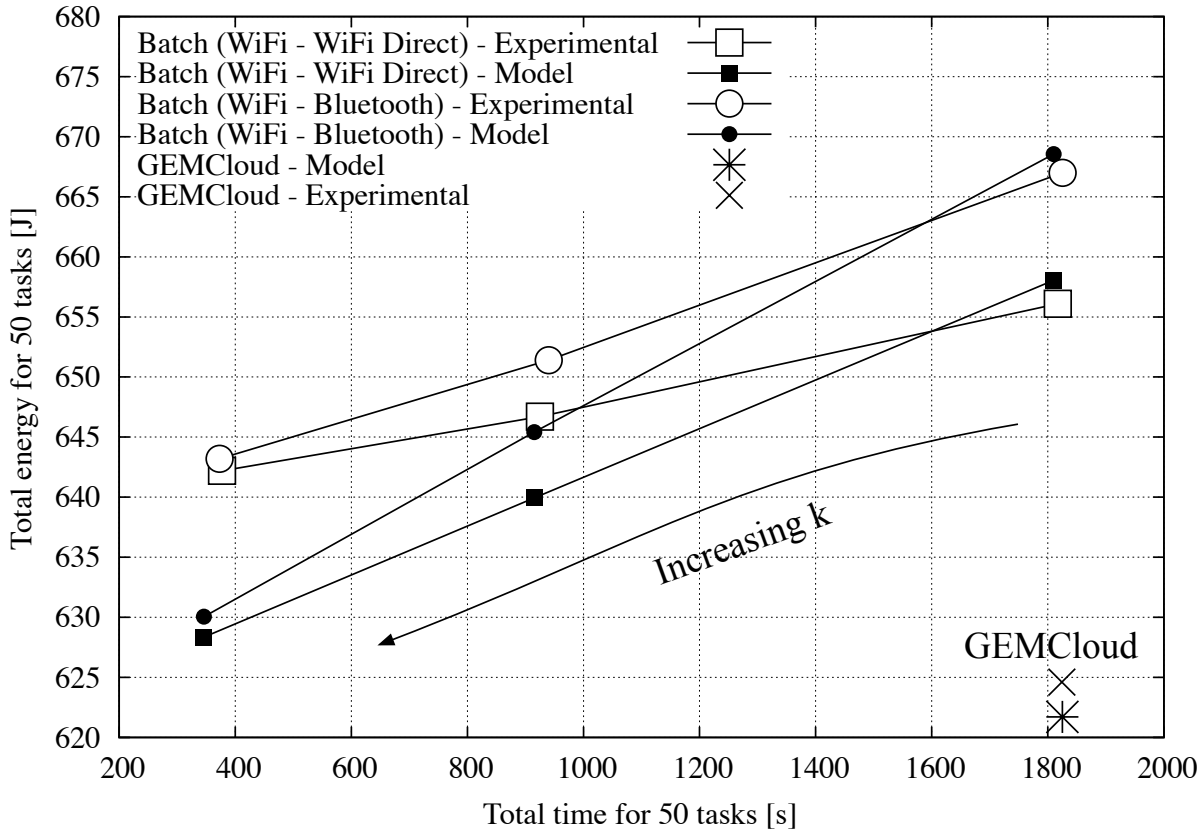


Figure 3.9: Experimental and Analytical Measurements. Total energy consumption vs. total time for computing 50 tasks, when $k \in \{1, 2, 6\}$ nodes are allowed to execute the tasks. For the batch method, the case refers to a system where the TEPs are connected to a single TDP.

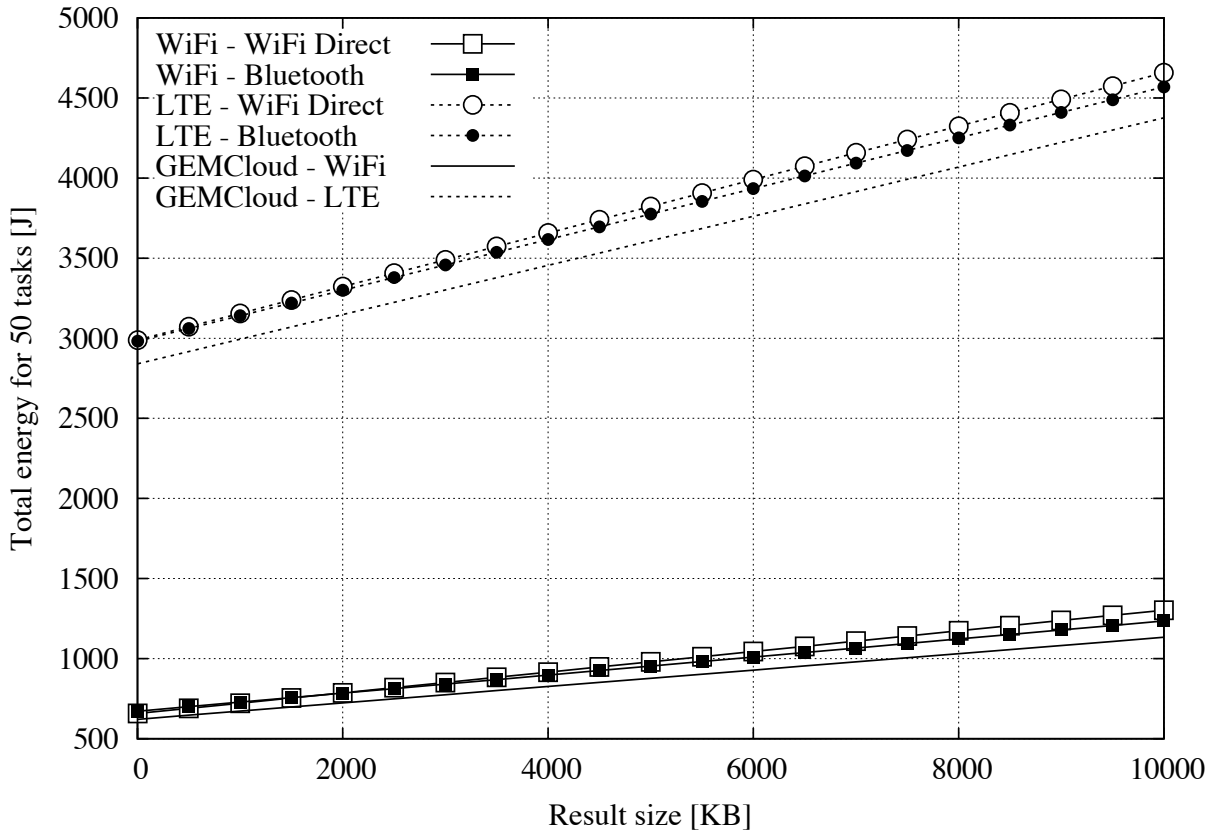


Figure 3.10: Analytical Results. Total energy consumption vs. result data size for a system where only one node is able to connect to the remote server and only one node is allowed to compute 50 tasks.

the analytical model. Figure 3.9 shows that both the experimental and analytical model results follow a similar trend when increasing the number of TEPs that participate in the distributed computation. Moreover, the analytical model error in estimating the energy consumption is, in the worst case, less than 2.5% and less than 8% for the total computation time. We note that the difference between the experimental and the analytical model results increases as the number of TEPs increases. This is due to the fact that the analytical model assumes that all the TEPs operations are completely synchronized (e.g., they start all the computations at the same time), thus reducing the total computational time and, as a consequence, the energy consumption of the TDP.

We then evaluate the impact of a larger result data size on the overall system performance. To this end, Figures 3.10 and 3.11, show the impact of the result data size on the total system energy and time, respectively, when only one device is able to connect to the Internet and is either acting as a standard GEMCloud TEP or as a TDP that distributes the tasks to a local TEP via WiFi Direct or Bluetooth. Figure 3.10 shows the substantial additional energy and time required for a system that uses the LTE technology with respect to WiFi for Internet connectivity. This is explained by the high idle power of LTE (i.e., 1288 mW [52]) and the lower throughput, when compared to WiFi. Regarding the D2D technology, instead, it is interesting to note that Bluetooth entails a lower energy consumption when compared to WiFi Direct, when the result data size is greater than 1000 KB, which is due to the lower energy consumption per bit of Bluetooth when compared to WiFi Direct (see Figure 3.7). However, the low throughput of Bluetooth severely impacts the total time required for the computation. When comparing the different task distribution policies, the analytical results show that allowing for a local caching of the tasks and results in the Batch method allows for a substantial reduction in the total computation time, allowing the Batch method combined with WiFi Direct, to outperform the total time required by a standard TEP directly connected to the remote server.

While Figures 3.10 and 3.11 show the impact of a larger result data size on the overall system performance, to better evaluate the benefits of local task distribution via D2D communications we focus our attention to the energy consumption of the TEP. To this end, in Figure 3.12 we plot the total energy consumption vs. the total time for computing 50 tasks at a single TEP, which has either direct Internet access through a WiFi AP or uses a D2D communication technology to reach a Batch

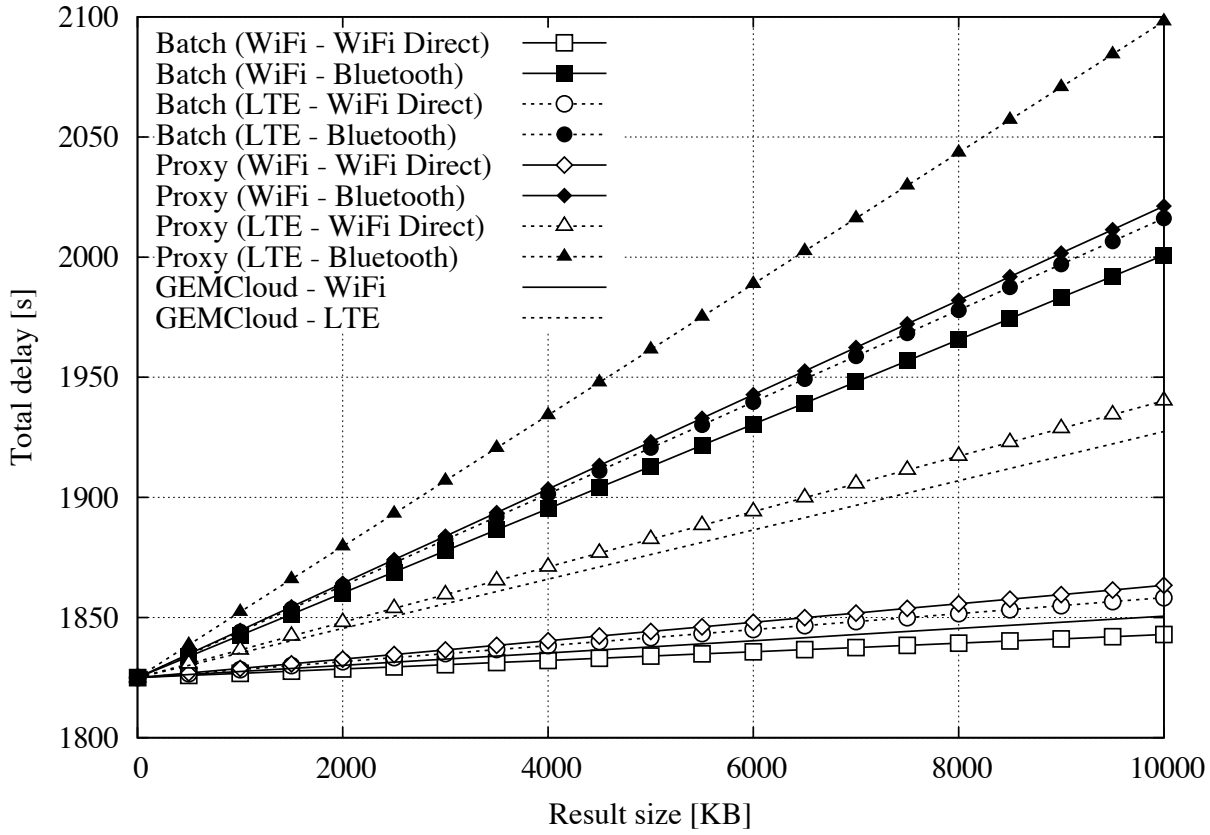


Figure 3.11: Analytical Results. Total time vs. result data size for a system where only one node is able to connect to the remote server and only one node is allowed to compute 50 tasks.

TDP, for different values of the result data size. Figure 3.12 shows that different trade-off can be achieved by the different communication technology. In particular, for small result data size (< 500 KB), the three type of TEPs attain similar performance. As the result data size increases, we note that the TEP that uses Bluetooth for communicating with the TDP entails the lower energy consumption at the expense of a larger communication delay, which is due to the reduced data rate supported by Bluetooth[4]. At the same time, both the TEP connected to the remote server via WiFi and the TEP connected to a local TDP via WiFi Direct achieve similar total time (with a difference of 7 seconds, in the worst case), while the TEP that uses WiFi Direct always consumes less energy. These results suggest that, whenever possible, using a D2D communication technology can provide a substantial reduction up to 41% in term of energy consumption at the expense of up to a 7.5% delay increase when using Bluetooth, or a slightly lower reduction up to 33.5% of the energy consumption when using WiFi Direct. We note that similar conclusions as the one presented for Figures 3.10-3.12 can be drawn for increasing task data size.

Finally, we investigate the impact of the number of TEPs on the system performance. Figure 3.13 shows the trade-off between energy and time to compute 50 tasks when multiple TEPs are invited to participate in the distributed computation via D2D communications. As expected, allowing for additional TEPs substantially decreases the total computation time. Figure 3.13 also shows the intrinsic limitations in performance achievable when using Bluetooth for the local task distribution when compared to WiFi Direct. This is due to the fact that the Bluetooth protocol only allows a TDP to serve a maximum of 7 TEPs [18], while WiFi Direct allows for the creation of a group of up to 49 devices [19].

3.7 Conclusions

In this chapter, we explored the impact of extending a traditional mobile computing system through D2D communications. In order to evaluate the impact of the different system components, we combined experimental results with an analytical model that allows us to infer the performance

[4]We consider further investigations into the possibility of achieving higher data rate, as supported by the Bluetooth specification [18], as future work.

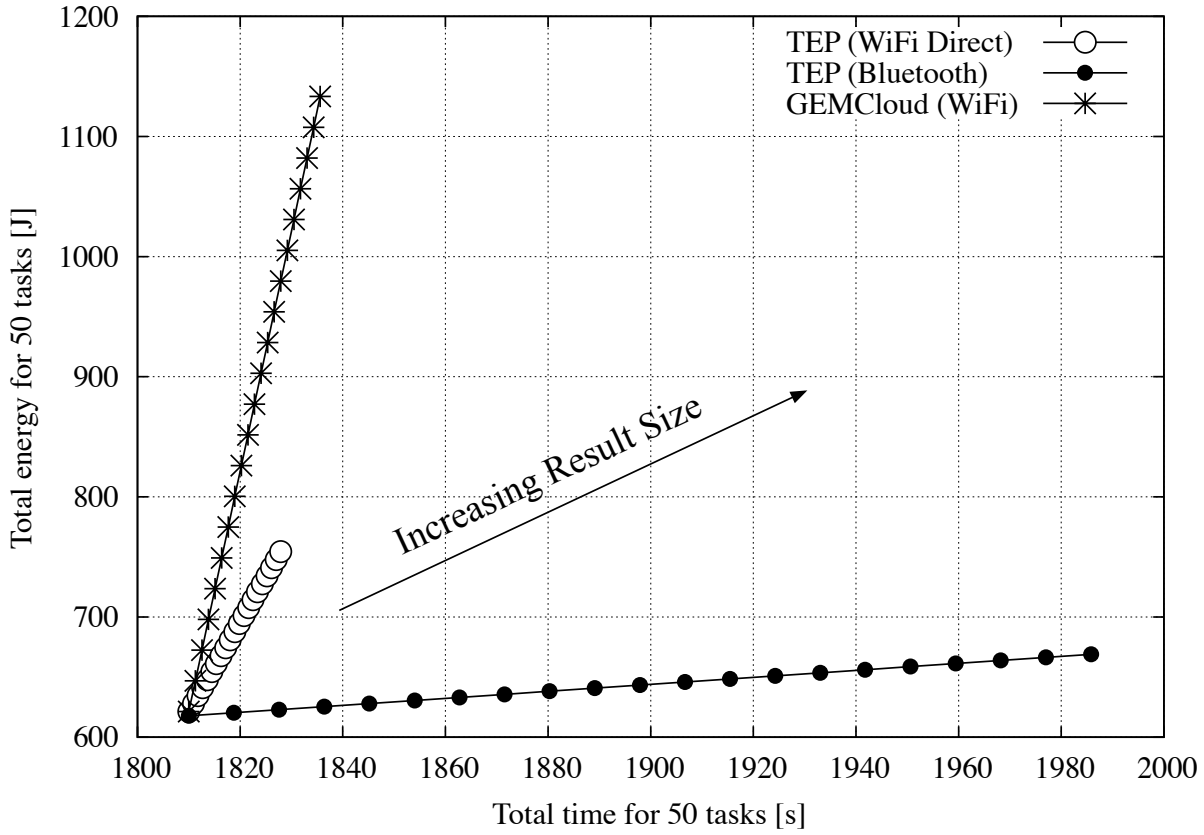


Figure 3.12: Analytical Results. Total energy consumption vs. total time for computing 50 tasks for different result data size, when a single TEP, connected to the remote server (GEMCloud) or to a local TDP Batch via WiFi Direct or Bluetooth, to compute 50 tasks. Each pair of values for energy and time to compute is obtained for a different value of r, from 1 KB to 10000 KB, with step length 500.

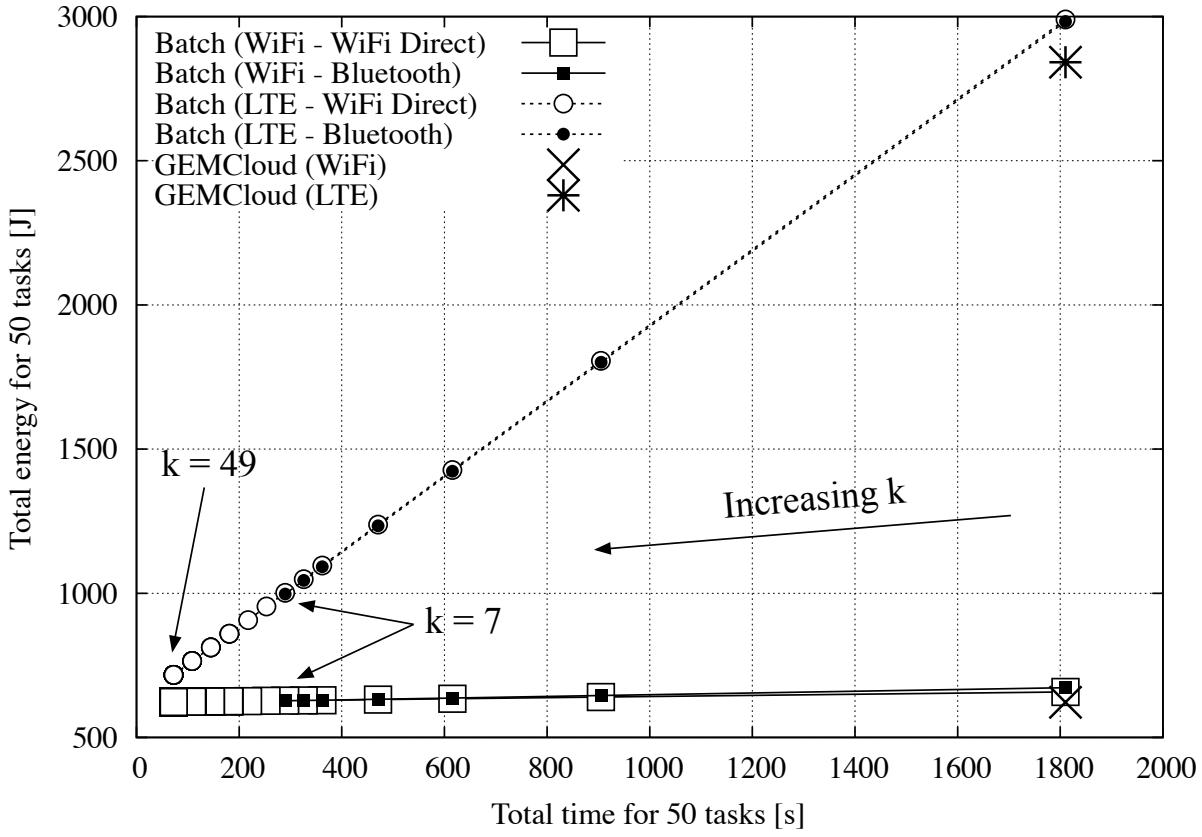


Figure 3.13: Analytical Results. Total time vs. total energy consumption for computing 50 tasks, when $k \in \{1, 2, \ldots, 49\}$ nodes are allowed to execute the tasks. For the Batch method, the case refers to a system where the TEPs are connected to a single TDP.

of the system under different settings. Our experimental results show that, when considering WiFi Direct and Bluetooth for D2D communication, the additional energy consumption required at the task distribution point is small, while substantial gain in term of energy consumption can be achieved at the device that is performing the computation. Analytical results provide a promising estimate of the performance of the system under different settings, showing that a large reduction in the total computation time with a negligible energy overhead can be achieved by parallelizing the computation to multiple TEPs connected through D2D communications. These results motivate future extensions of mobile computing to multi-hop and heterogeneous networks, as will be described in the following chapters.

Chapter-4

Enabling Multi-Group Communications in D2D Networks

4.1 Introduction

The previous chapter demonstrated the feasibility of offloading computation to nearby neighbors; however, in order to fully realize an ad hoc mobile computational offloading system, it is pertinent to understand how ad hoc networks scale and incorporate additional nodes, which may exist multiple hops away. To this end, multi-hop wireless networks have been largely developed to meet the needs of a variety of applications where infrastructure-based wireless networks are difficult to deploy and maintain. Most applications require the participating nodes to be able to route data to help extend network connectivity. These protocols have mainly been used for tactical military communications, first responder applications and sensor network operations.

In this chapter, we explore different methods for enabling multi-group WiFi Direct communication. Starting with stock Android and its implementation of WiFi Direct, we shed some light on the limitations and design considerations for realizing multi-group communication on mobile devices running Android 4.4.2. To overcome these limitations, we first propose and analyze a TCP-based time sharing mechanism where the device that connects multiple groups, referred to as the gateway node, is required to iteratively switch between different WiFi Direct groups in order to relay data from one group to the other. We then exploit a particular configuration that allows for a device to be simultaneously connected to two different groups, to implement and analyze the performance of a UDP-based broadcast technique as well as a UDP/TCP hybrid solution. In addition, we consider how

to implement intergroup communication when not limited to a stock version of Android and present a few changes that can be made to the current WiFi Direct implementation in order to facilitate inter-group communication. We discuss the tradeoffs in terms of both energy and time when implementing these different approaches. Our approaches can be used as building blocks for realizing a WiFi Direct based Mobile Ad Hoc Network by interconnecting Android devices.

4.2 WiFi Direct

4.2.1 Single-group Communications

WiFi Direct [19] is a standard released by the WiFi alliance that enables ad hoc communication between nearby devices, without requiring a wireless Access Point (AP). WiFi Direct utilizes IEEE 802.11 a/b/g/n infrastructure mode, and can transmit either at 2.4 GHz or 5 GHz.

During ad hoc communication, devices form a group were one of them is the Group Owner (GO) and all the others are considered Group Members (GM). It is important to note that these roles are not predefined but are negotiated during the construction of the group and remain fixed for the entire duration of the group. Additionally, WiFi Direct groups can also include standard IEEE 802.11 nodes that do not support WiFi Direct and are referred to as Legacy Clients (LC).

The nodes that support WiFi Direct go through a group formation process in order to determine the roles of the GO and the GMs. There are three group formation cases: standard, persistent and autonomous [19, 51]. During the standard group formation, the nodes listen on channels 1, 6, and 11 in the 2.4 GHz band and, after finding another device, they negotiate as to which will act as the GO. This is done in a handshake process, where the devices exchange an *intent value*, and the device with the highest value becomes the GO. After the roles have been established, the devices go through a WiFi Protected Setup (WPS) Provision phase and, after completion, the GO assigns an IP address using the Dynamic Host Configuration Protocol (DHCP). The persistent group formation process allows for a faster reconstruction of previous groups. During the persistent group formation, the GO negotiation phase is replaced by an invitation exchange, and the WPS Provisioning process

is significantly reduced by reusing the stored network credentials. In autonomous group formation, a node assigns itself the role of GO and creates its own group.

According to the standard [19], the GO represents an AP-like entity that provides basic service set (BSS) functionality and services for the associated clients. Acting as a soft AP, the GO advertises and allows nodes to join the group. The advertisement and group maintenance are performed through beacon packets, just like a typical IEEE 802.11 AP, and the GO is responsible for giving control of the channel to nodes in its network as well as routing data through clients in its group[1]. As a result, the group topology is a $1 : N$ hierarchical structure, where multiple clients (i.e., GMs and LCs) are connected to one GO.

WiFi Direct devices can operate concurrently with an infrastructure wireless network, through multiple physical or virtual MAC entities. Moreover, the specification [19] does not preclude a WiFi Direct device from simultaneously operating as a member of more than one group. However, both the multiple MAC functionalities and the simultaneous operations in multiple groups are out of scope of the standard.

4.2.2 Multi-group Communications

Our focus is to investigate the feasibility and relative performance of different techniques for allowing communication between different WiFi Direct groups. In this regard, in order to act as a gateway between two (or more) WiFi Direct groups, a device can use the MAC virtualization functionality described earlier. Thus, the physical radio interface can be shared by multiple separate MAC entities that independently use the hardware. Following the same principle, a device can act as a gateway between a local ad hoc network and the Internet, through the simultaneous connection to an infrastructure AP. We note that, when connected to a standard WiFi AP, the device is in fact acting as a LC since it is not leveraging the WiFi Direct protocol.

Given the above, we envision two possible scenarios in which a device can act as a gateway between two separate groups: the first where the gateway node acts as a client in both groups (see Figure 4.1), and the second scenario in which the gateway is the GO of one group and a client in the

[1]Routing data between clients in a group is allowed but not defined by the standard.

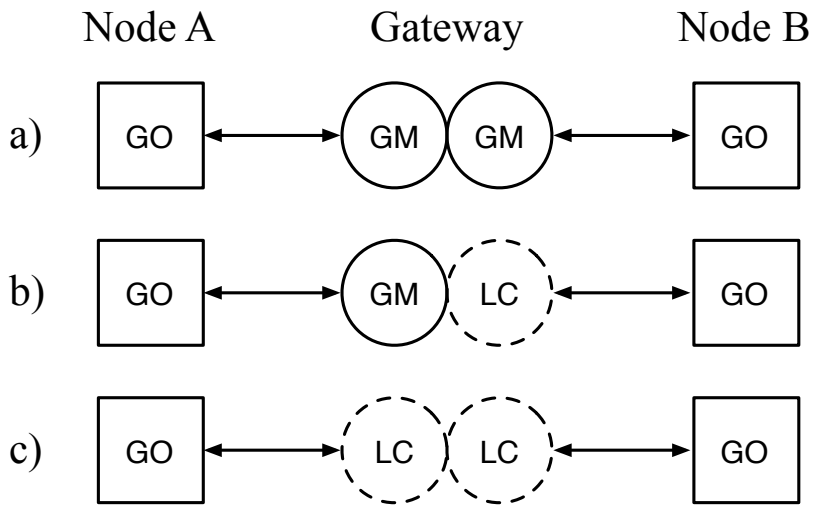


Figure 4.1: Multi-group communication scenarios where the gateway node acts as a client in two groups.

other (see Figure 4.2). Extensions of these scenarios to more than two groups or the case in which the GO hosts more than one group are also possible. However, the case in which the GO hosts more than one group only allows for an increase in the number of clients that the GO can simultaneously serve.

4.3 Multi-group Networking on Android Devices

As described in the previous section, a traditional WiFi Direct network topology is represented by a hierarchical structure, where the GO is at the center of all the communications, but multi-group communications are allowed by the standard specification. In principle, it is therefore possible to realize a multi-group wireless network where some of the devices are clients, or simultaneously a GO and a client, of more than one group. However, the MAC virtualization and the simultaneous operations in multiple groups are not required by the standard, and thus their availability depends on the actual implementation.

In what follows, we first provide a high level description of the Android implementation of WiFi Direct, we then present the limitations of using stock Android, and we finally describe our proposed methods for realizing intergroup communications with stock and non-stock Android devices.

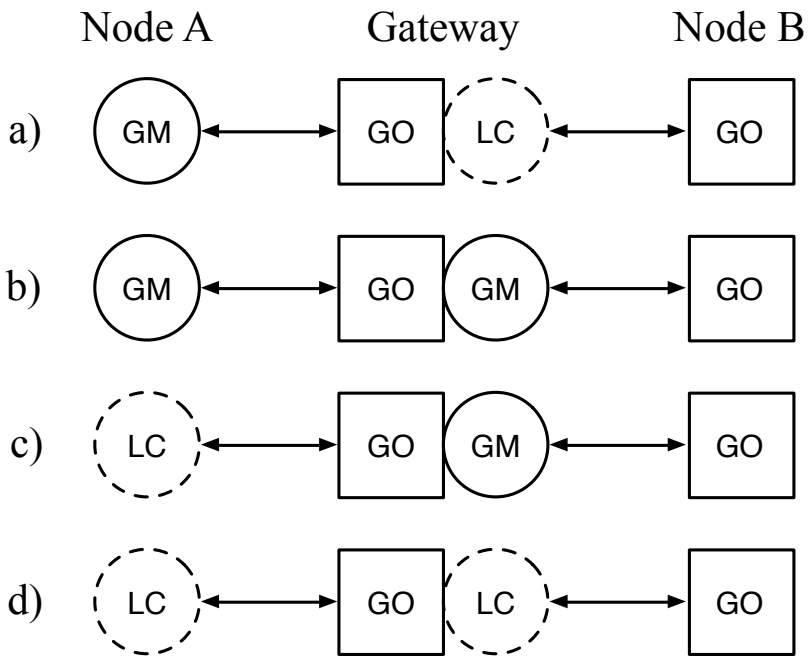


Figure 4.2: Multi-group communication scenarios where the gateway node acts as the GO in one group and as a client in the other.

4.3.1 WiFi Direct on Android

Android has included support for WiFi Direct since version 4.0 (API level 14), within the Android's Wi-Fi P2P framework [61]. This framework complies with the WiFi Alliance's WiFi Direct certification program. Using the Android APIs, a developer can discover and connect to other devices that support WiFi Direct and then communicate over an ad hoc connection. The *WifiP2pManager* class provides all the methods that allow the interaction with the WiFi hardware, like discover, connect and disconnect to peers. Due to the interaction with the hardware, all these methods are asynchronous, and the framework uses listeners to notify the application of the status of a call.

In order to use the *WifiP2pManager* functionalities, an Android application needs to have access to the hardware and run on a device that supports the WiFi Direct protocol. If both these conditions are satisfied, the *WifiP2pManager* undergoes an initialization process, where all the WiFi Direct related services are started. This allows the device to react to WiFi Direct events, like client discovery and connection.

For establishing an ad hoc connection, a device needs to discover nearby peers that support WiFi

Direct and are available for connection. After a device is discovered, the actual connection can be made. If one of the two devices is already a GO of an existing group, the other joins the group as a GM, while if none of them is a GO or a GM, according to the WiFi Direct protocol described earlier, the devices negotiate their role within the group. Moreover, if a GO sends a connection request to the GM of another group, the connection is refused, and if a device sends a request to a GM, the GM forwards the request to its GO. We note that a device can be a GO of a group without any connected clients.

After the connection, the GO acts as a DHCP server and assigns an IP address to the clients. The DHCP scope is fixed by the framework and cannot be modified by the developers. As a result, GOs of different groups always have the same IP address (192.168.49.1), while the connected clients receive an IP address at random from the same range (192.168.49.2-254).

The GO advertises its group through a unique Service Set ID (SSID), that can be used by devices that do not support the framework (i.e., LC) to join the group. In Android, standard WiFi operations, like scanning for available networks and the connection to infrastructure APs, are handled by the WiFi framework. Developers can use the *WifiManager* class to perform WiFi specific functionalities.

For a detailed description of the WiFi and WiFi Direct frameworks and the relative APIs, we refer interested readers to the Android API guides [62].

4.3.2 Limitations of Stock Android

Even though Android is an open-source operating system, it provides a certain set of limitations in the way in which the developer can interact with the different services and hardware. We acknowledge that these limitations can be removed by reprogramming the operating system (e.g., rooting the device). However, this operation is non trivial for an average user and its legality is still controversial in several countries. Thus, we first focus on devices that run stock Android so that we can provide a general methodology for allowing WiFi Direct multi-group networking on a broader set of devices.

Using stock Android, intergroup communications need to be handled at the application layer, and all the transport and network layer functionalities, like setting the IP address and managing the routing table, cannot be performed natively. Moreover, the developer is not allowed to create either custom

virtual network interfaces nor multiple virtual MAC entities. As a result, the methods described in Section 4.2.2, where a device simultaneously operates in multiple WiFi Direct groups either as a GO or a GM, cannot be implemented directly.

Nevertheless, our experiments show that the WiFi Direct functionalities are able to concurrently operate with an infrastructure wireless network[2], through the simultaneous utilization of the *WifiP2pManager* and *WifiManager*. In this case, we infer that the OS is in fact virtualizing the network interface (or the MAC). Following the same rationale, we exploited the GO group advertisement process and connected a device participating in a WiFi Direct group to a second group as a LC, using the *WiFiManager*. While this operation was allowed and the connection between the devices was correctly established, we were not able to create a unicast communication to and from the gateway node. In particular, for the scenario b) of Figure 4.1, the gateway was able to receive data from both Node A and Node B, but was not able to communicate with either one of them; for the scenarios a) and d) of Figure 4.2, instead, the gateway was able to communicate with Node A, while the communication with Node B was not allowed. According to our experiments, this is due to the fact that the DHCP protocol assigns the same IP address to multiple GOs, thus creating routing problems. A flooding like UDP-based communication protocol can overcome this limitation, as described in Section 4.3.3.

4.3.3 Proposed Solutions

Time Sharing. Given the limitations of the current Android implementation of WiFi Direct and the restrictions on implementing routing functionalities at the application layer, we propose a time sharing mechanism in which the gateway node switches between two (or more) groups. In this way, all the scenarios presented in Figures 4.1 and 4.2 can be successfully implemented. We note that there is no built in switching functionality, rather switching is comprised of a disconnection from the current group, a request to scan for active nodes, and a request to connect to a new group. In what follows, we describe the main differences between the different scenarios.

We start by analyzing the scenarios in Figure 4.1. In these cases, the gateway node acts as a client in both groups and iteratively connects to and disconnects from the two groups. Scenarios a) and b)

[2]We note that this functionality is not described in the Android APIs [62].

are limited to devices that support WiFi Direct: while in case a) the gateway can fully exploit the WiFi Direct protocol, case b) can potentially provide some gains in terms of switching time since it is using both the *WiFiManager* and the *WiFiP2pManager*. A special case is represented by case c), where the gateway is a LC in both groups. This method relies solely on the *WiFiManager*, thus allowing devices that do not support WiFi Direct to take on the role of routing information between different groups. By acting as a LC, the gateway cannot capitalize on the WiFi Direct power saving mechanisms, but it follows the same process required for switching connections between traditional WiFi APs. While methods described in a) and b) have the potential for a faster and seamless bridging of groups, the method in c) allows for any device to act as a gateway node, allowing for a more inclusive network.

The scenarios in Figure 4.2, instead, consider the situation in which the gateway node is a client in one group and acts as a GO of the other. In these cases, the switching process potentially requires more time then the methods described in Figure 4.1 because all of the operations required to create one of the groups need to be performed every time the gateway node ceases to be the GO. The role and responsibilities of being a GO are negotiated during the group formation and cannot be transferred [19]. Scenarios b) and c) in Figure 4.2 fully exploit the WiFi Direct protocol, while scenarios a) and d) use both WiFi Direct and the standard WiFi functionalities.

Simultaneous Connections. As described in Section 4.3.2, by combining standard WiFi and WiFi Direct functionalities into the gateway node, it is possible for the gateway node to simultaneously maintain a physical/MAC layer connection to two groups. However, this imposes some restrictions on the actual application data exchange. We thus explored the different scenarios presented in Figures 4.1 and 4.2 that utilize both WiFi Direct and WiFi, with the objective of finding a suitable configuration and communication protocol that allow for intergroup data exchange.

To this end, we ran several experiments by implementing the different scenarios from Figures 4.1 and 4.2 with the different network sockets (e.g, stream, datagram and multicast sockets) provided by Android. We found that when combining a *LC/GM* (or, equivalently, *GM/LC*) gateway node with a multicast socket, a specific implementation of the UDP datagram socket, the gateway node is able to forward data between the two groups. This is because the multicast socket allows the node to specify the particular interface to be used by the socket for receiving and transmitting data packets. We note

that this functionality is not available for a traditional datagram or stream socket, and Android follows a weak end system model (i.e., routing decision are based only on the destination IP address and type of service) [63]. Moreover, it is important to note that the multicast socket encapsulates a one-to-many unicast communication and, as a result of this, cannot fully utilize the total available WiFi and WiFi Direct bandwidth.

From our experiments, the same gateway configuration allows the gateway node to receive and send data over the *LC* link (i.e., the standard WiFi) while simultaneously connected to both groups also with a unicast socket, while no data can be routed with a unicast socket over the WiFi Direct link. This is due to the fact that Android prioritizes the WiFi link over the WiFi Direct link. We thus propose a simple protocol, referred to as *Hybrid*, that exploits this functionality and uses the multicast socket as a control channel that, if necessary, triggers a gateway node configuration change. According to this protocol, the group that has data to send to the other group, uses the control channel to notify the gateway node. After the reception of the control message, the gateway node checks if it is connected to this group as a *LC* or a *GM*. In the first case, the gateway is allowed to receive data, thus it notifies the source and it starts receiving data. After receiving the data, the gateway disconnects from the first group and forwards the data to the second group using a TCP connection. In the second case, instead, the gateway node is not allowed to receive data from the WiFi Direct link. Thus, it notifies the source, disconnects from both groups and connects back with the right configuration (i.e., it switches from *GM/LC* to *LC/GM*). At this point the communication continues as in the previous case. We note that this change in configuration can be avoided if a second gateway node is present (i.e., one gateway node is a *GM/LC* while the other gateway node is an *LC/GM*).

A data dissemination protocol that follows a similar approach has recently been proposed in [64]. The protocol presented in [64] operates under the assumption that an additional relay node is present in each group. This protocol uses the configurations a) and d) of Figure 4.2, and exploits the additional relay node to forward information from the GO to the gateway node. By doing so, the method presented in [64] mitigates the fact that the gateway node has the same IP address as its GO. However, this does not provide seamless communication between two groups, since a TCP connection defaults to the WiFi interface (i.e., only the LC link can use TCP sockets). As a result, the gateway node trans-

mits data to the other nodes in its group by broadcasting UDP packets. While this protocol does not require any configuration change, it requires the presence of an additional relay node in every group (which cannot always be guaranteed), and adds an additional communication hop.

In order to fully explore the WiFi Direct protocol for multi-group networking, we downloaded the source code of Android $4.4.2$ and modified the existing implementation of WiFi Direct to assign a unique IP address to the GOs and change the DHCP range accordingly (the IP address and DCHP range are statically defined inside the *WifiP2pService*). This simple modification allows the simultaneous operation of a gateway node that uses both the WiFi and WiFi Direct interfaces. Further modifications are required for creating multiple WiFi Direct interfaces. This requires changing the *WifiP2pService* to instantiate different *NetworkInterfaces* as well as to change the *SystemServer* and the *Context* to instantiate a new WiFi Direct Service and a new identifier for this service, respectively.

4.4 Performance Evaluation

In this section we evaluate the impact at the gateway node of allowing multi-group WiFi Direct communications. We measured both the time and the energy required to forward data between two groups. Our proposed schemes were implemented using three second generation Asus Nexus 7s, which were released in 2013. The 2013 Nexus 7 has 16 GB of storage, 2 GB of memory, and a 1.5GHz quad-core Snapdragon S4 Pro 8064 CPU [55]. Each of our devices are running Google's Android version $4.4.2$.

4.4.1 Test Environment

Similar to the work in [54], we measured the current from the battery for each of our experiments using an Arduino Uno [56]. To this end, we added a $0.005\Omega \pm 1\%$ shunt resistor in series with the battery and measured the voltage across the resistor to obtain the current. However, the voltage drop across this resistor was too small to be read by the Arduino Uno [56]. We therefore configured an op-amp to act as a non-inverting amplifier with a gain of 977. This allowed the Arduino's analog input to read the voltage across the resistor throughout the tests. Additionally, due to the non-linear properties

of lithium-ion batteries [57, 58], like the one used in the Nexus 7, we restricted our experimental measurements to battery level above 70%. [58] has shown that for battery levels between 60% and 100%, the internal impedance of lithium-ion batteries scales almost linearly with regard to the state of charge.

All of the measurements were taken in an indoor workspace, and the nodes were stationary during these experiments. We acknowledge that in real life, nodes are generally mobile, and may enter or leave a network. However, to accurately determine the impact of the switching process on the gateway node and to limit the variability across different experiments, we kept the nodes stationary. Moreover, all the results of this section have been obtained with the screen turned off.

For all the scenarios presented in Figures 4.1 and 4.2, we consider a situation in which the gateway must relay 10 MB of data between Node A and Node B. Thus, the gateway first receives all the data from one node, and then sends the data to the other node following one of the techniques described in Section 4.3.3. For the *Time Sharing* experiments, we measured the energy and time required for the switching process, from the disconnection from the first group to the actual availability of the second link. We assumed that the gateway is able to communicate with the second node as soon as the operating system updates the ARP table with its address. For the *Simultaneous Connections* solutions, instead, we measured the total time and energy consumption required to relay the data between Node A and Node B, since there is no actual switching between groups. Moreover, all the groups are considered to be persistent to allow for a faster switching between the two groups, that we assume to be known by the gateway. This accounts for the situation in which the gateway node switches between the groups multiple times[3]. All the results presented in this section are obtained over 50 runs of the same experiment, where the gateway node forwards the data between Node A and Node B. However, we also performed some continuous tests where the gateway node exchanges data back and forth between the two groups and found consistent results (i.e., the total time and energy consumption of the continuous tests were simply the sum of the time and energy for the isolated tests).

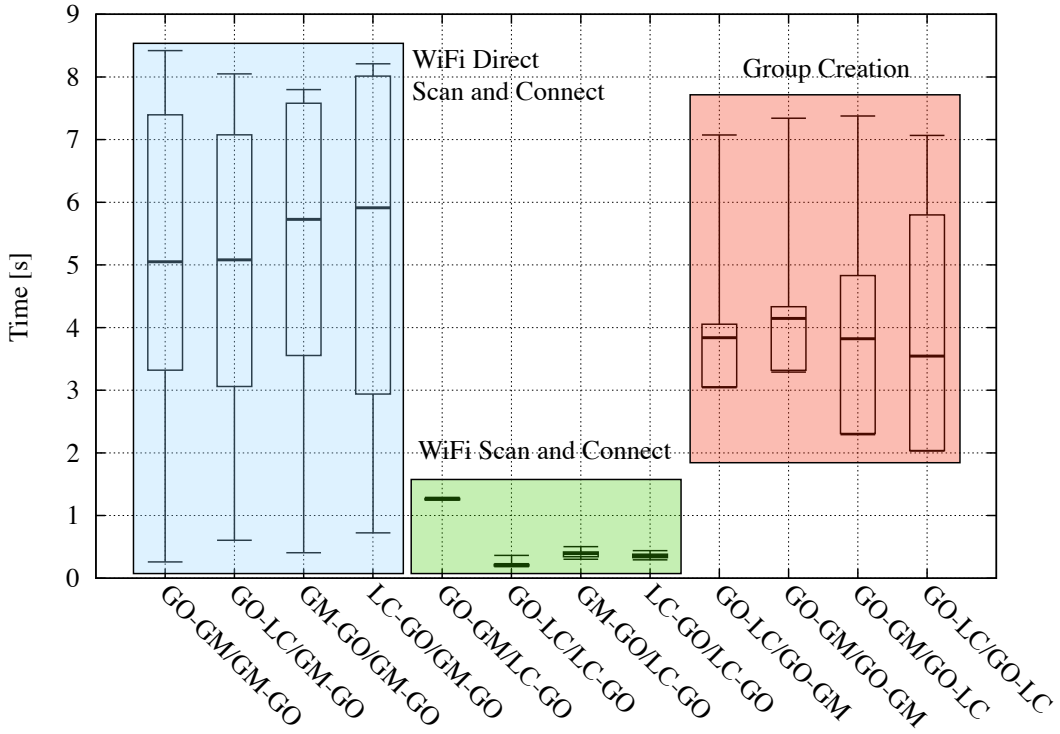


Figure 4.3: Experimental Measurements. Time required to switch between two groups for the different scenarios described in Section 4.2.2.

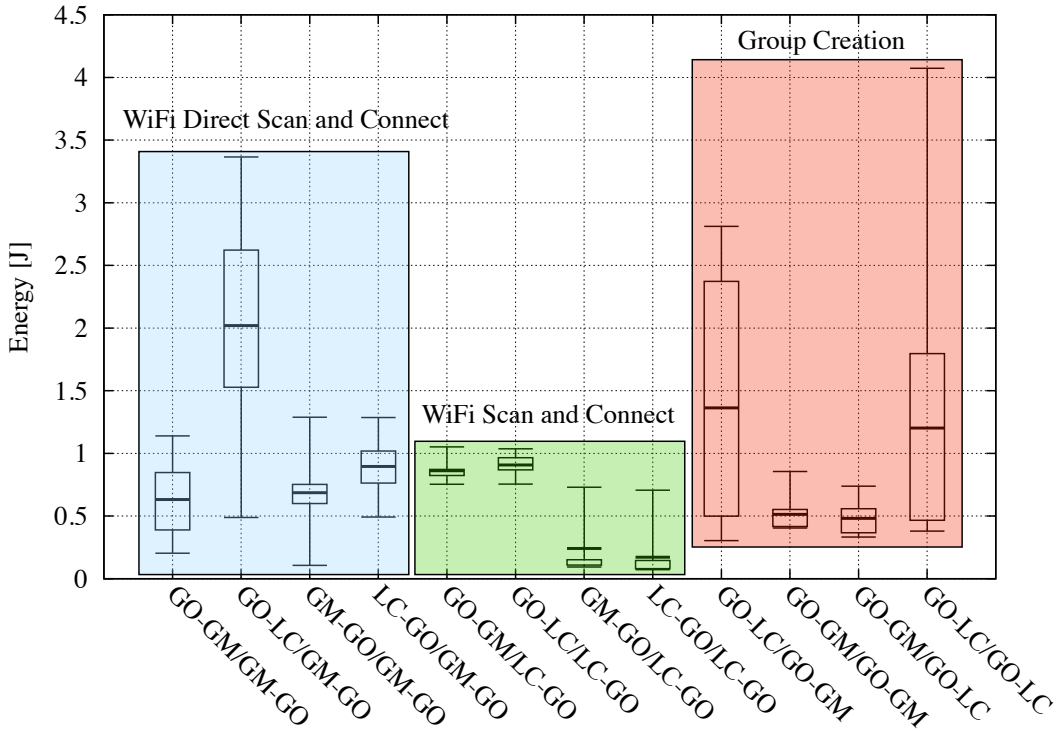


Figure 4.4: Experimental Measurements. Energy required to switch between two groups for the different scenarios described in Section 4.2.2.

4.4.2 Numerical Results - Time Sharing

To measure the amount of energy and time required to switch between the two groups, we performed isolated tests to determine the impact that the switching process has on a particular node. We first established some baseline measurements for the transmission and reception energy consumptions. Using the experimental setup described above and Iperf [59], we measured the average energy required to send (and receive) 10 MB of data between devices, when acting as a GO, a GM or a LC. We then logged the total energy required at the gateway node to receive the 10 MB data, switch and transmit the data to the other group. Finally, we subtracted the reception and transmission energy from the total energy to determine the energy required for switching between the groups.

In Figures 4.3 and 4.4, we plot the time and energy, respectively, required to switch between the two groups, for all the scenarios described in Section 4.2.2. Figure 4.3 shows that the switching time depends only on the method used by the gateway node for connecting to the second group. In particular, the lower switching times are achieved by the gateway that connects to an existing group as a LC. This is because the Android API that manages the standard WiFi connection allows to scan and connect to an existing AP faster than the WiFi Direct API, that instead relies on a combination of asynchronous call and event notifications. As a result, when connecting as a GM, the gateway is either able to join the second group in less than 500ms (see the minimum values of cases */GM-GO in Figure 4.3) or, in other cases, the switching process can require even more than 8s. We note that these variable connection times are due to the WiFi Direct protocol that requires, during the group formation process, an initial discovery phase where each device iteratively searches and listens over channels 1, 6 and 11 for nearby clients [19, 51].

When the gateway creates the second group and acts as a GO, instead, the average switching time is around 4s, with lower connection times for the case in which it is connecting to a LC. It is important to notice that when the gateway node acts as a GO for the second group, it cannot send connection requests to the LC. In this case, instead, the LC is constantly scanning for the second group and, as soon as the gateway creates the group, it will connect to the group.

Our results are in line with the experimental evaluations of the WiFi Direct protocol presented

[3] We acknowledge that, when creating the groups for the first time, the energy and time required are on average higher.

in [51] and [65]. For example, [65] showed that the average time required for a device to autonomously create a group and immediately become a GO (autonomous group formation) is 3s, while the average time required for a node to join an existing group is 6s.

Regarding the energy, Figure 4.4 shows that the energy required to switch between two groups is similar across the different scenarios, with the exception of the cases in which the gateway is required to switch from a LC to a WiFi Direct node, e.g., a GO or a GM. This is due to the fact that the Android WiFi Direct implementation is built on top of the standard WiFi APIs but it requires some additional initialization (i.e, starting the WiFi Direct services) before the device can actually use the WiFi Direct protocol. For the same reason, switching between WiFi Direct roles or from a WiFi Direct role to a LC, instead, does not provide any significant impact on the energy consumption of the node. In addition, we note that even if the energy consumed by the gateway node when switching as a LC is comparable and, in some cases, lower than the energy required by the other scenarios, this energy is consumed in a short amount of time (i.e., around 500ms in Figure 4.3) and thus the power consumption of this operation is much higher. This is because the LC implements a more aggressive approach for scanning and connecting to an existing group, that causes the higher power consumption when compared to the other methods.

4.4.3 Numerical Results - Simultaneous Connections

We now focus on the gateway node configurations that allow for simultaneous connections in two WiFi Direct groups. In this regard, in Figures 4.5 and 4.6 we plot the time and energy, respectively, required to transfer 10 MB of data between the two groups, for the scenarios in which the gateway node acts as a LC in one group and a GM in the other (i.e., scenarios GO-LC/GM-GO and GO-GM/LC-GO). We remind the reader that in these configurations (except for the *Time Sharing* scenarios that are included for comparison), the gateway node is allowed to maintain a simultaneous physical connection to both groups (see Section 4.3.3). Figures 4.5 and 4.6 show that substantial gains in both time and energy can be achieved by the techniques that allow for simultaneous connection in both groups. As expected, the best performances are attained by the non-stock Android implementations, which represent the lower bound on the performance achievable by any scheme. Nevertheless, both the

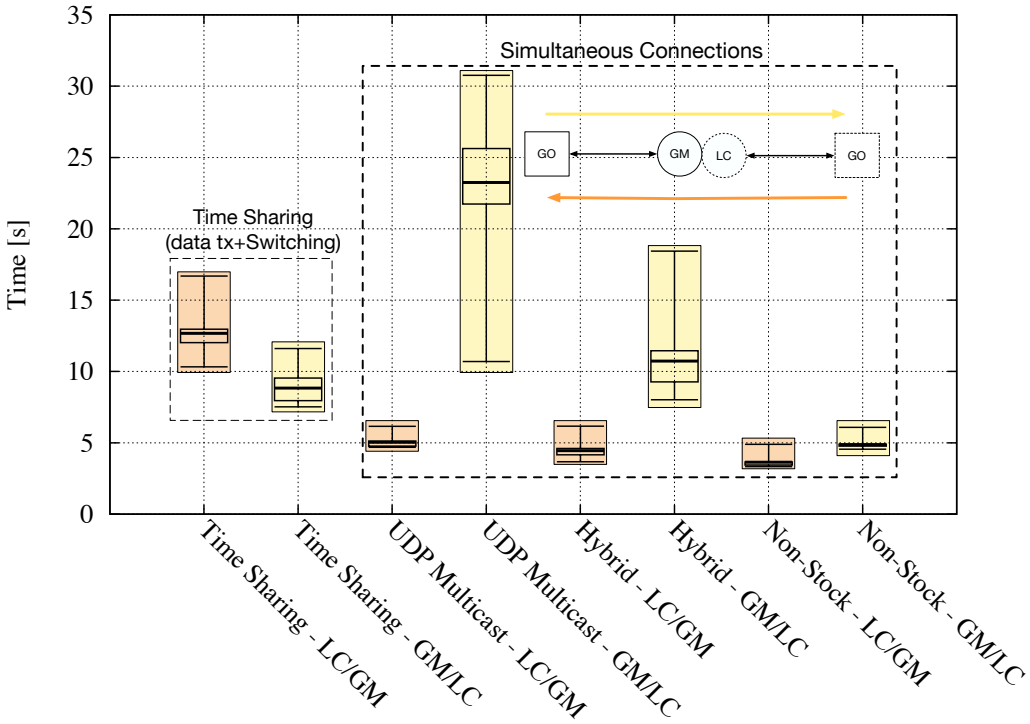


Figure 4.5: Experimental Measurements. Time required to transfer 10 MB of data between two groups for a gateway node acting as LC and GM.

UDP Multicast and the *Hybrid* approaches perform very close to the lower bound, with the exception of the *GM/LC* configuration that requires a higher time and energy consumption. This is due to the encapsulation of a one-to-many unicast communication protocol that impacts the data reception over the WiFi Direct link of the *UDP Multicast* approach, and to the configuration switch of the *Hybrid* scheme. Moreover, we note that while the *Time Sharing*, *Hybrid* and *Non-Stock* implementations use TCP sockets for reliable data transmission, the *UDP Multicast* communication does not implement any retransmission mechanism and, as such, is subject to a variable data loss (in our experiments, an average of 93% of the total data was successfully delivered).

Finally, while all the results presented in this section were obtained using a 2013 Nexus 7 [55], similar conclusions can be drawn when changing to a different device. In particular, we ran the tests presented in this section on a 2012 Nexus 7, which has a completely different hardware configuration (e.g, different memory, processor and wireless chipset), with Android 4.4.2 and found similar relative results in term of transmission/reception and switching time, but an overall higher (almost doubled)

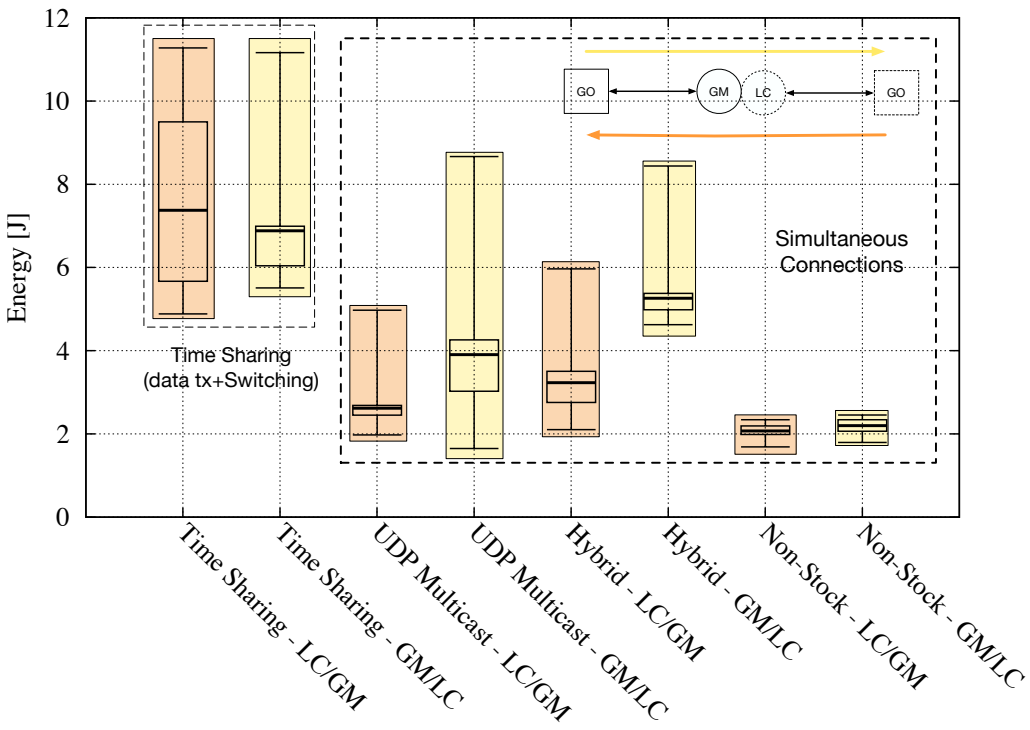


Figure 4.6: Experimental Measurements. Energy required to transfer 10 MB of data between two groups for a gateway node acting as LC and GM.

energy consumption for all schemes. These results suggest that similar conclusions can be drawn for different Android devices.

4.5 Conclusions

In this chapter, we propose and explore different practical methods for enabling multi-hop ad hoc networks using the WiFi Direct standard. We present the limitations of enabling WiFi Direct multi-group communication, and we propose and analyze different simple, yet effective mechanisms to allow the communications between devices belonging to different groups. For all the proposed methods, we discuss the achievable tradeoffs in terms of both time and energy.

Our experimental results show that a faster switching time can be achieved by connecting to an existing group as a legacy client, at the expense of a higher power consumption. In addition, switching from standard WiFi to WiFi Direct entails a higher energy consumption, when compared to all the other scenarios. Better performance can be achieved by techniques that exploit the device ability

to maintain simultaneous physical connections to two groups. In particular, the *Hybrid* approach, where a UDP multicast communication is used as a control channel for triggering a configuration switch, allows for a reliable data transfer between groups and attains performance close to a non-stock Android implementation.

Future work includes the inclusion of a routing protocol, and further modifications to the Android OS to allow simultaneous communication over multiple WiFi Direct interfaces.

Chapter-5

Mobile Computational Offloading in Multi-hop Ad Hoc Networks

5.1 Introduction

By combining the work from the prior chapters, we strive to enable ad hoc networks with the ability to utilize all available computational resources in a network, especially due to the fact that single hop ad hoc networks might not fully satiate the application space where infrastructure-based wireless networks are difficult to deploy and maintain. In doing so, we aim to enable the next step in the evolution of mobile computation offloading by considering all the computational resources available in a multi-hop ad hoc network.

This chapter aims to address the opportunities and limitations of a multi-hop computational offloading system, using a time and energy model for offloading, a routing heuristic, and an iterative task distribution algorithm. To verify the accuracy of our model, we implement and compare our algorithm to a variety of other task distribution schemes, on an Android-based multi-hop WiFi Direct network. We demonstrate that the benefit of expanding the number of computational resources that can be utilized for task completion outweighs the overhead to transmit tasks to non-nearest neighbor nodes for all cases where the computation has a higher cost than the communication. Finally, we show the ability of our routing heuristic to select nodes for offloading such that all of the available resources in the network are fully utilized.

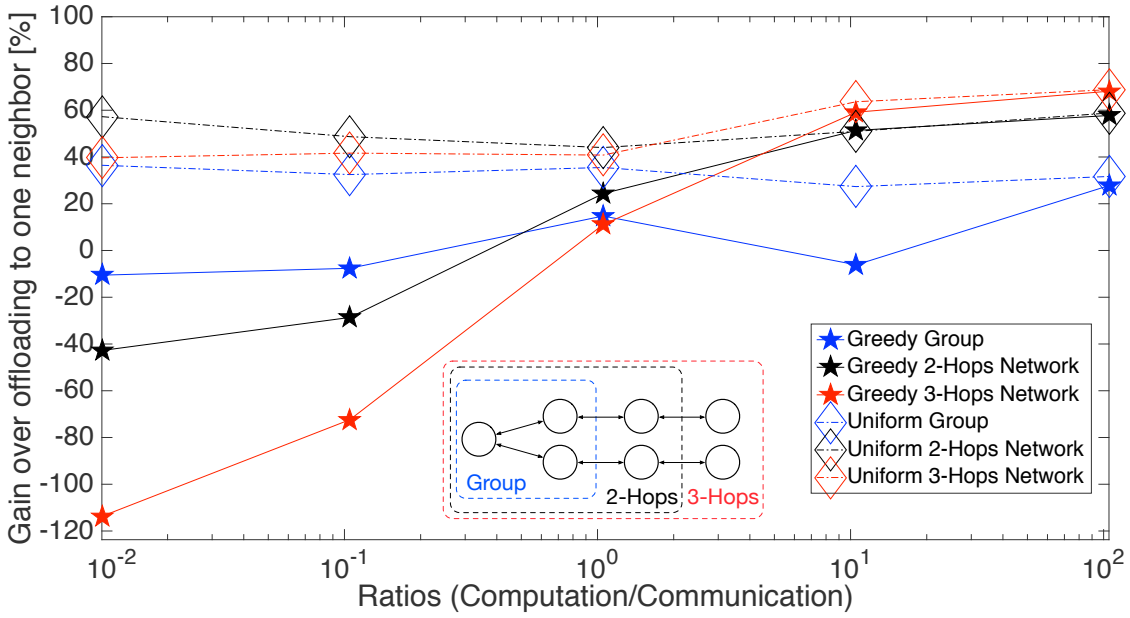


Figure 5.1: Experimental measurement of the percent gain over offloading to only a single neighbor of a simple greedy and uniform task distribution scheme.

5.2 Motivation

Similar to [66], our mobile to mobile computational offloading system is composed of a set of N mobile devices, organized into a network through ad hoc communications. The devices are co-operative, meaning that we do not consider the presence of any selfish user. Moreover, we assume the existence of a global routing protocol, and each node has knowledge of all other available nodes within the network.

At a certain point in time, one of the devices needs to run a complex application, which can be divided into clearly defined tasks, as is the case for a military SIGINT application [67], or non-invasively determining the structural integrity of a building during disaster relief efforts [68]. These tasks are independent from each other, and their computation can be parallelized by offloading the tasks to different devices. The results of all the tasks' execution is then merged in order to complete the initial complex application. As a result, assigning a task to a device requires the transmission and reception of the task and the results of the task execution over the ad hoc route. This requires communication energy consumption on all the nodes along the routing path, and incurs a time delay due to the propagation of the data throughout the network.

In order to determine the benefit of utilizing multi-hop mobile to mobile computation offloading, we implemented a computational offloading system on Android devices, utilizing the WiFi Direct protocol [19]. We note that the functionalities for creating multi-group networks using WiFi Direct are not natively implemented in Android [69]. Nevertheless, the work presented in [64, 69] provide different solutions (e.g., time sharing between groups, broadcasting and multicasting data between groups and modifications of the Android OS) to seamlessly enable multi-group communication using stock and non-stock Android devices.

Given the above, we developed a testbed by interconnecting several 2013 Nexus 7s $(\mathrm{N7})$, using the modified version of Android presented in [69]. The 2013 N7 has 16 GB of storage, 2 GB of memory, and a 1.5GHz quad-core Snapdragon CPU [55]. For all the results presented in this chapter, we limited each device to compute only one task at a time, and the device cannot request the next task until it has returned the result. Each of the presented data points are an average of thirty trials. All of the measurements were taken in an indoor workspace, and the nodes were stationary during these experiments. We acknowledge that in real scenarios, nodes are generally mobile, and may enter or leave a network. However, to accurately determine the impact of the task assignment and to limit the variability across different experiments, we kept the nodes stationary. Moreover, all the results of this chapter have been obtained with the screen turned off.

To evaluate the performance of the envisioned multi-hop mobile to mobile computation offloading system, in Figure 5.1 we compare the experimental performance gain to complete a set of 50 homogeneous tasks relative to a task distribution that only utilizes one additional device with extending the computation to a larger network (see Figure 5.1). For assigning the tasks, we implemented both a simple Greedy task distribution, where each device in the network requests a new task as soon as it has completed the previous task, and a Uniform task distribution, where the tasks are uniformly distributed throughout the network. While both task assignments provide substantial performance gain compared to offloading to a single neighbor when the computation time is much larger then the time required to assign the tasks and receive the results, extending the computation to the multi-hop network can be detrimental when the communication time is greater than the computation.

Furthermore, this provides no analysis of the affect this offloading has to the entire network. For

instance, does an assignment bring the network closer to partitioning? Additionally, in the event that partitioning is inevitable, these approaches do not have the ability to partition the network in the least adverse manner. As a result, it is not straightforward to determine how to best assign the tasks due to the communication costs of distributing the tasks. Moreover, while both the Uniform and the simple Greedy task distribution already provide gains in some cases, it is not possible to directly determine the performance of a Greedy distribution before assigning the tasks and, at the same time, it is not clear if further performance improvements are even possible. Thus, in the next sections we first model the different elements of the system under consideration and then present an iterative algorithm that is guaranteed to return the optimal task assignment.

5.3 System Model

In this section, we present new task time and energy models for a multi-hop network, as well as a heuristic that models how important a device is to the task of routing data within the network. Unlike the model presented in Section 3.5.6, the aim of these new models is simply to capture the relative cost, in terms of time and energy, that is associated with a given device computing a particular task. The model presented in Section 3.5.6, aims to more accurately capture the actions performed by a partcipating device, rather than represent the effects that sharing computation has on the network. As a result, we use these new models to derive different task distribution policies, which will be presented in Section 5.4.

5.3.1 Task Time Model

We start by considering the total time D_i required to compute a given task at a particular device i. We define this total time as the time between the task assignment and the end of the transmission of the result of the task execution. Thus, in order to determine the time required to assign, compute and then receive the results of the task execution we need to consider two main components, namely the communication time required to distribute the task and receive the results, and the time actually spent for the task execution.

In order to derive the communication time, we consider that each task is characterized by three parameters, namely the task data size t, the result data size r and the task complexity c. Thus, according to this definition, assigning a task (t, c, r) to a device requires the transmission/reception of t bits, while the result of the task execution to be reported back to the task generator entails the transmission/reception of r bits. The time required to execute a single task depends on the CPU of the mobile device and on the complexity c of the task to be executed, which, without loss of generality, can be considered as a function $e_i(c)$. Therefore, the total time required to compute a single task at device i depends on the device's computational capabilities, and on the characteristics of the radio technology used for the data exchange, as well as on the number of hops H_i between the device that generated the tasks and device i.

Given the above, the total delay $D_i(t, c, t)$ experienced by a task (t, c, r) assigned to device i is given by

$$D_i(t, c, r) = \sum_{l=1}^{H_i} \left(\frac{t}{T_l^{\text{tx}}} + \frac{r}{T_{\bar{l}}^{\text{tx}}} \right) + \sum_{l=1}^{H_i - 1} 2T_l^{\text{sw}} + e_i(c), \tag{5.1}$$

where T_l^{tx} represents the estimated transmission throughput[1] of the l communication hop, $\bar{l}$ represents the communication hop l in the reverse path, and T_l^{sw} accounts for additional switching time when routing the data between two subsequent communication links as can be the case, for example, with the WiFi Direct protocol (see, e.g., [69]). We note that the transmission throughput also accounts for parallel use of the same communication link (by, e.g., the simultaneous assignment of different tasks) and, since the throughput on wireless networks fluctuates due to the signal strength and channel impairments, assigning values to the transmission throughput only serves as an example of the achievable performance of a network.

5.3.2 Task Energy Model

In most cases, the networks described in this chapter will consist of battery operated mobile devices. As a result, it is important not only to characterize the delay experienced by the computation of each task, but also to determine the impact in terms of energy consumption of the task distribu-

[1] This metric can be expanded to use a moving average of the channel throughput or other forms of uncertainty analysis.

tion. Therefore, in what follows we define the total energy consumption required to assign a task to a particular device.

Similar to the task time model defined in the previous section, distributing a task entails a communication energy consumption and the task execution energy consumption. In order to define the communication energy model, we first define P_l^{tx} and P_l^{rx} to be the transmission and reception power consumption of a particular ad hoc communication link l, respectively. We note that a particular link l actually represents a pair of nodes (h, k), where h is the device transmitting the data and k is the device receiving the data. Thus, with a little abuse of notation we can write $P_l^{\text{tx}} = P_h^{\text{tx}}$ and $P_l^{\text{rx}} = P_k^{\text{rx}}$, where P_h^{tx} represents the power consumption of mobile device h during transmission, and P_k^{rx} represents the power consumption of mobile device k during reception. In the same way, we define P_i^{ex} to represent the power consumption of mobile device i during task computation.

As a result, we define the total energy expenditure when assigning a task (t, c, r) to device i as

$$\begin{aligned} E_i(t,c,r) = &\sum_{l=1}^{H_i} \left[\frac{t}{T_l^{\text{tx}}}(P_l^{\text{tx}} + P_l^{\text{rx}}) + \frac{r}{T_{\bar{l}}^{\text{tx}}}(P_{\bar{l}}^{\text{tx}} + P_{\bar{l}}^{\text{rx}}) \right] \\ &+ \sum_{l=1}^{H_i-1} 2E_l^{\text{sw}} + P_i^{\text{ex}} e_i(c), \end{aligned} \tag{5.2}$$

where, similar to Eq. (5.1), $\bar{l}$ represents the communication hop l in the reverse path (e.g., if l represents the pair of nodes (h, k), $\bar{l}$ refers to the pair (k, h)), and E_l^{sw} represents the amount of energy required for switching when routing the data between two subsequent communication links.

5.3.3 Routing Metric

Although the aforementioned models assess the cost associated when assigning a task to a particular device, neither represent the effect this assignment has on the network as a whole. To this end, we define a heuristic routing metric R_i, to represent the routing importance a particular device i has to the network, and hence we define R_i in terms of energy.

Given that the aim of R_i is to represent a device's importance to maintaining a connected network, we start by examining the trade-off between routing and computing. More specifically, it is important

to understand how computing a task can affect the descendants for a given device i. We define R'_i to be the amount of residual energy remaining in the batteries of device i's descendants, divided by the residual energy in device i's battery. It is sufficient to only consider the R'_is reported by a device's immediate descendants, $\boldsymbol{K}$, due to the fact that all subsequent descendants will have to either route through a device in $\boldsymbol{K}$, or be a member of $\boldsymbol{K}$ itself. This allows us to formulate R'_i as:

$$R'_i = \frac{\sum^{n \in K} R'_n + Q_n}{Q_i} \tag{5.3}$$

where Q_i is the amount of residual energy in the battery of device i. In addition to representing how assigning a task can affect a device's descendants, it is also important to consider any tangential effects that an assignment can have on other devices in the network. Specifically, how does assigning a task to device i affect either device i's ancestors, or devices that are siblings to device i? By considering the R'_is of a device's ancestors, $\boldsymbol{A}$, we are able to represent any tangential effects an assignment has on the network; however, we argue that these effects are dependent on the ratio between the energy required to communicate a task and result and the energy required to compute. This allows us to define the routing metric for a given device i to be

$$R_i(t, c, r) = R'_i + (\sum^{a \in A} R'_a) \frac{\frac{t}{T_l^{\text{tx}}} P_l^{\text{tx}} + \frac{r}{T_{\bar{l}}^{\text{tx}}} P_{\bar{l}}^{\text{tx}}}{P_i^{\text{ex}} e_i(c)} \tag{5.4}$$

Similar to the Task Energy Model, P_l^{tx} and P_l^{rx} are the transmission and reception power consumption of a particular ad hoc communication link l, respectively. Additionally, we note that this metric operates under the assumption that the network can be formed into a tree structure, when distributing tasks. We argue that this is feasible due to the fact that in most cases, tasks would be originating from a single device, allowing for the use of spanning tree algorithms to fit this constraint.

5.4 Task Distribution

The goal of our analysis is to find the distribution of a set of heterogeneous tasks that minimizes the overall cost metric, by all devices in a multi-hop network. This can be seen as a combinatorial optimization problem, known as the general assignment problem (GAP). Although the GAP has been shown to be NP-hard, under certain conditions, the GAP can be translated to the Linear Bottleneck Assignment Problem (LBAP), which can be solved in polynomial time. Furthermore, we present an iterative solution to solve an augmented form of the LBAP. In what follows, we present the formulation of the GAP, LBAP, and an augmented form of the LBAP problems.

5.4.1 General Assignment Problem (GAP)

Let N be the number of mobile devices participating in the computations and M be the number of heterogeneous tasks to be distributed. Furthermore, let i be a given mobile device and j be a particular task, such that $i \in N$ and $j \in M$. We define $\boldsymbol{\phi}$ to be a binary matrix representing a potential distribution of tasks, where each entry $\phi_{ij} = 1$ if task j is assigned to device i, and $\phi_{ij} = 0$ otherwise. Moreover, let $\boldsymbol{C}$ be a matrix, where each entry C_{ij} represents the cost to assign and execute task j at device i. Given the above, we define our objective function to be:

$$\min_{\boldsymbol{\phi} \in \Phi} \mathrm{F}(\boldsymbol{\phi}), \tag{5.5}$$

where Φ is the collection of possible task distributions, ϕ, and $\mathrm{F}(\boldsymbol{\phi}) = \sum_{i=1}^{N}\sum_{j=1}^{M}\phi_{ij}C_{ij}$ represents the cost of a given distribution. We note that $\mathrm{F}(\boldsymbol{\phi})$ accounts for the assignment of tasks to devices and is suitable for finding the minimum cost required to perform the distributed computation of the M tasks.

This problem has been shown to be NP-Hard, and the only method to ensure an optimal task distribution is to exhaustively search over all possible permutations of tasks Φ. As a result, several algorithms and heuristics have been proposed that generate a distribution that is capable of approximating the optimal distribution. In particular, Dantzig et al. proposed a greedy algorithm where the tasks are ordered from longest to shortest, before being assigned uniformly to devices. Additional so-

lutions include a fully polynomial time approximation scheme proposed by Kellerer et al. [70]. This scheme partitions the tasks into long tasks and short tasks. A variable number of tasks are distributed from each set. This number of tasks to be distributed is generated every cycle until there are no more tasks left. More recently, Luo et al. demonstrated that their multi-robot auction based algorithm converges to a solution [71]. Lou et al. go on to prove that their algorithm converges to a solution to GAP with an approximation ratio of $1 + \alpha$, meaning that their solution is within $1 + \alpha$ times the optimal solution, where α is the "aggressiveness[2]" of each of the participating agents. Furthermore, Racer et al. developed a heuristic to approximate an optimal solution to the GAP [72].

5.4.2 Linear Bottleneck Assignment Problem (LBAP)

As stated above, the LBAP is a special case of the GAP, where the number of heterogeneous tasks and the number of mobile devices are equal. The solution to LBAP takes the form of a bipartite graph, meaning that each device can only by assigned one task. Similar to GAP, we define ϕ to be a permutation, corresponding to a solution to distribute the tasks to the devices, and Φ to be the set of all solutions, such that $\phi \in \Phi$. Likewise, let $\boldsymbol{C}$ be a cost matrix, where C_{ij} is the cost associated with device i executing task j. Furthermore, we redefine $\mathrm{F}(\boldsymbol{\phi})$ from Eq. (5.5) to be:

$$\mathrm{F}(\boldsymbol{\phi}) = \max_{i=1,\ldots,N} \left\{ \sum_{j=1}^{M} \phi_{ij} C_{ij} \right\}. \tag{5.6}$$

To solve the LBAP, Zimmerman et al. proposed a solution using Dijkstra's algorithm, where the authors constructed a graph, similar to the one seen in Figure 5.2 [73]. Figure 5.2 explicitly shows the "first stage," where all of the devices are connected to all of the tasks, and the edges connecting these nodes are representative of the costs associated with that device executing the connected task. Each task is then connected to a set of potential stages, k_i for $i \in 1, z$, where an example of a subsequent stage can be seen in Figure 5.3. Finally, by traversing from node s to node e, an optimal solution can be found by looking at the device-task pairs that comprise the route.

[2]How aggressive each agent is towards bidding.

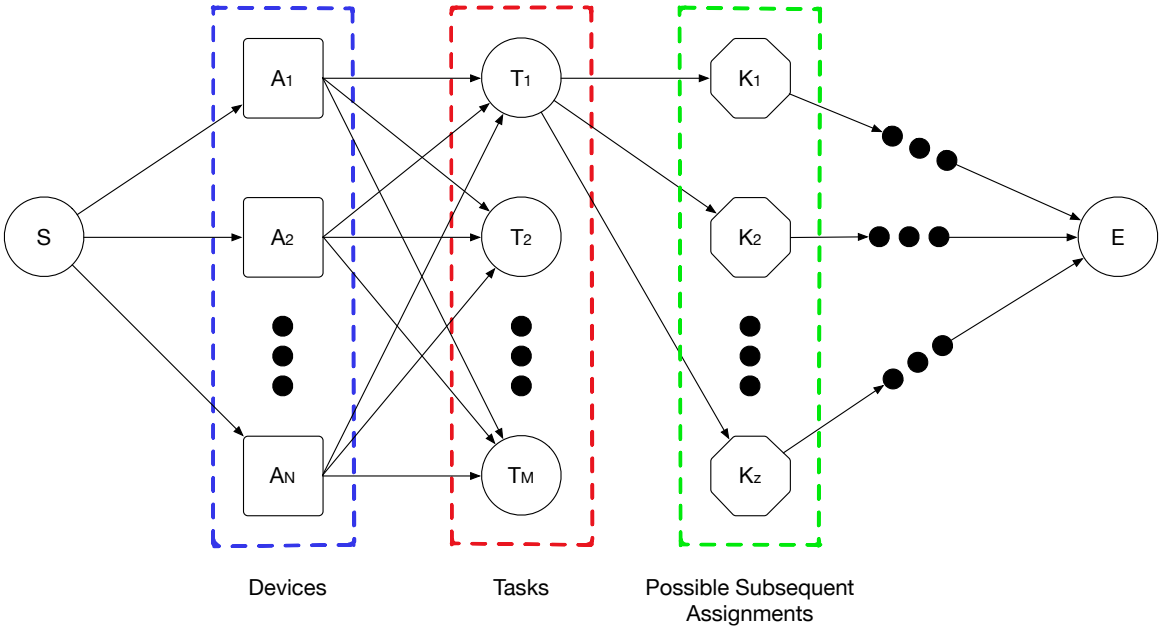


Figure 5.2: A representation of how Dijkstra's algorithm is used to solve the LBAP.

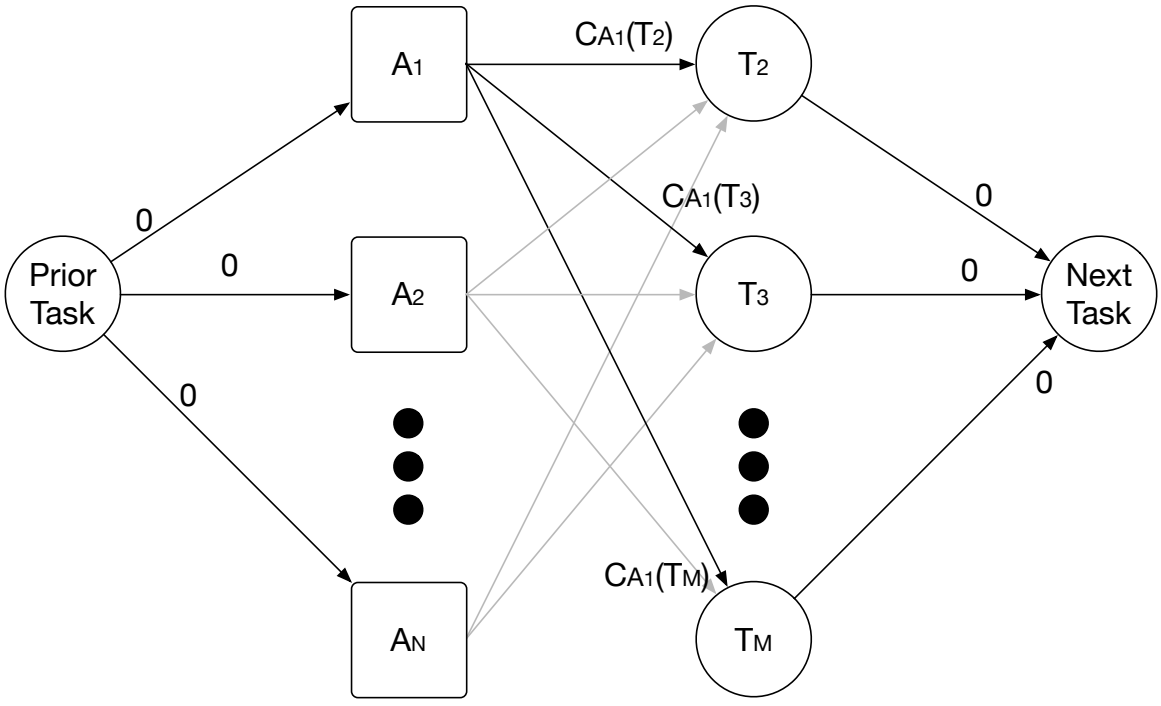


Figure 5.3: A representation of an individual stage, K, in Figure 5.2

5.4.3 Augmented Form of LBAP

Additionally, there exists an augmentation of the LBAP, where the system wishes to optimally distribute M homogeneous tasks to N devices. Although the LBAP requires the number of tasks and devices to be the same, when the tasks are homogeneous, we can linearly combine multiple tasks to make N heterogeneous tasks. As a result, this special case cannot be solved using the same algorithms presented in Section 5.4.2, due to their in ability to combine homogeneous tasks ; however, we have developed an iterative algorithm to solve this special case. In what follows, we present and prove the optimality of our iterative task distribution algorithm.

Proposed Policy

We aim to find the optimal policy $\boldsymbol{\phi}$ for the following objective function:

$$\min_{\boldsymbol{\phi}\in\Phi} \mathrm{F}(\boldsymbol{\phi}) \tag{5.7}$$

where

$$\mathrm{F}(\boldsymbol{\phi}) = \max_{i=1,\ldots,N} \left\{ \sum_{j=1}^{M} \phi_{ij} C_{ij} \right\} \tag{5.8}$$

To solve this problem, we have developed an iterative algorithm, which assigns tasks one at a time to the device who's cost will change the least with the addition of this new task. More formally, let us redefine $\boldsymbol{\phi}$ to be a vector such that $\boldsymbol{\phi} = [\phi_0, \phi_1, \ldots, \phi_N]$, where ϕ_i represents the number of homogeneous tasks assigned to a particular device i, such that $\phi_i \in \mathbb{Z}^+$, $0 \leq \phi_i \leq M$ and $\sum_{i=1}^{N} \phi_i = M$, and Φ is the set of all the possible task assignments that satisfy these conditions. Moreover, let $\boldsymbol{C} = [C_1, C_2, \ldots, C_N]$ be a cost vector, where C_i with $i = 1, \ldots, N$ represents the cost of assigning a task to device i. To this end, let $\boldsymbol{X}^{\alpha}$ be a vector of length N, so that X_i^{α} represents the number of tasks that have been assigned to device i up to the assignment of task α, with $\alpha \leq M$. In addition, let $\boldsymbol{Y}$ be a decision vector of length N, so that all but one of the elements of $\boldsymbol{Y}$ are zero. Given the above, the algorithm for finding the optimal task assignment is defined in Algorithm 1.

To prove that Algorithm 1 is able to determine an optimal solution to the optimization problem

Algorithm 1 Cost–Optimal Task Distribution

1: X^0=$\{0, 0, 0, ..., 0\}$
2: **for Each Task** $\alpha \in [1, ..., M]$ **do**
3: $\mathbf{Y}$=$\{0, 0, 0, ..., 0\}$
4: $i = \text{argmin}((\mathbf{X^{\alpha-1}} + \overrightarrow{1}) \circ \mathbf{C})$
5: $Y_i = 1$
6: $\mathbf{X}^\alpha = \mathbf{X^{\alpha-1}} + \mathbf{Y}$
7: **end for**

defined in Eq. (5.5), we first show that starting from an optimal task assignment for K tasks, it is possible to construct an optimal task assignment for $K + 1$ tasks.

Theorem 5.4.1. *Given an optimal K tasks assignment Γ^K, an optimal $K + 1$ tasks assignment Γ^{K+1} can be obtained as*

$$\Gamma^{K+1} = \underset{\psi \in \Psi^{K+1}}{argmin}[\, max_i\{\psi_i C_i | i = 1, ..., N\}],$$

where $\Psi^{K+1} = \{\psi^i\}$, $\psi^i = \Gamma^K + \epsilon^i$ and ϵ^i is an all-zeros vector with a one in position i, for each $i = 1, \ldots, N$.

Theorem 1. *See Appendix 7.3.*

We will now use the result of Theorem 5.4.1 to show that the algorithm presented in Section 5.4 returns an optimal solution to the optimization problem defined in Eq. (5.6), when considering the objective function defined in Eq. (5.6).

Theorem 5.4.2. *Algorithm 1 yields an optimal solution to the optimization problem defined in Eq.* (5.5).

Theorem 1. *See Appendix 7.4.*

5.5 Maximizing the Number of Tasks

While Section 5.4 formulated the objective function for finding the optimal task distribution for a set of tasks, it is also important to understand the full capabilities of a network, such as the maximum number of tasks the network can compute before partitioning or in a given amount of time. These

types of problems can be solved using Mixed Integer Linear Programming (MILP). In what follows, we present the objective function for solving this problem. We note that we cannot find the optimal number of tasks the network can compute for heterogeneous tasks using this method, due to the fact that MILP will return a set containing only the "cheapest" tasks. As a result, we present the objective function for homogeneous tasks.

Similar to Section 5.4, we start by defining N to be the number of mobile devices participating in the computation and M to be the number of homogeneous tasks that can be computed. Furthermore, let B be a vector representing all of the devices' batteries capacities such that $\boldsymbol{B} = [B_0, B_1, \ldots, B_N]$, where B_i represents the total energy capacity of device i's battery. Likewise, we define $\mathbf{X}$ to be a vector such that $\boldsymbol{X} = [X_0, X_1, \ldots, X_N]$, where X_i represents the number of homogeneous tasks assigned to a particular device i, such that $X_i \in \mathbb{Z}^+$, $0 \leq X_i \leq M$ and $\sum_{i=1}^{N} X_i = M$. Moreover, let $\mathbf{C^{energy}}$ and $\mathbf{C^{time}}$ be $N \times N$ matrices, representing the cost, either as a function of energy or time, associated with a given action. For instance C_{ij}^{energy} is the cost to device i for forwarding a task to device j and C_{ii}^{energy} is the cost to device i to compute, and if applicable receive, a task. Finally, we define the time required to compute the set of tasks to be $\mathbf{C^{time}} \cdot \mathbf{X}$ and L a vector representing a time constraint[3]. Given the above, we define our objective function as

$$\begin{aligned} \underset{M}{\text{maximize}} \quad & \mathrm{M} = \sum_{\mathrm{i}=0}^{\mathrm{N}} (X_i) \\ \text{subject to} \quad & \mathbf{D}(\mathbf{t}, \mathbf{r}, \mathbf{c}) \cdot \mathbf{X} \leq \mathbf{B} \\ & \mathbf{E}(\mathbf{t}, \mathbf{r}, \mathbf{c}) \cdot \mathbf{X} \leq \mathbf{L} \\ & 0 \leq X_i : \forall i \in [1, ..., N] \end{aligned} \tag{5.9}$$

where $0 \leq X_i : \forall i \in [1, ..., N]$ is constraining each device to a minimum of zero tasks. As stated above, this optimization problem can be solved using MILP.

[3] L does not have to be defined, for instance if L is not defined the function will search for the maximum number of tasks that the network can compute before partitioning.

5.6 Results

In order to evaluate the performance of the algorithms presented in Section 5.4, we define the cost for a given device i to be:

$$C_i(t, c, r) = \alpha D_i(t, c, r) + (1 - \alpha) R_i(t, c, r) \tag{5.10}$$

where α is an adjustable parameter emphasizing whether the algorithm is optimizing for delay ($D_i(t, c, r), \alpha = 1$) or network lifetime ($R_i(t, c, r), \alpha = 0$). The performance of these algorithms are compared to an algorithm that uniformly distributes the tasks to the nodes as well as a greedy task distribution algorithm. We start by presenting the performance of these algorithms when distributing a given set of homogeneous tasks. Additionally, we couple the routing heuristic with these algorithms to gauge its ability to maximize the number of homogeneous tasks a network can compute. These values are then compared to the solutions generated by the optimal algorithm defined in Section 5.5. Finally, we present the performance of these algorithms when distributing heterogeneous tasks.

5.6.1 Performance Evaluation: Homogenous Tasks

For all the results presented in this section, we consider a network comprised of homogeneous devices, that is distributing homogeneous tasks. To this end, we define the cost vector $\boldsymbol{C} = [C_1, C_2, \ldots, C_N]$ of the optimization problem in Eq. (5.6) such that $C_i = D_i(t, c, r)$, $\alpha = 1$, where $t = 13$B, $r \in \{10\text{KB}, 100\text{KB}, 1\text{MB}\}$ and c is determined so that $e_i(c) \in \{10\text{ms}, 100\text{ms}, 1\text{s}, 10\text{s}\}$.

Offloading Results

After implementing the testbed described in Section 5.2, we compared the task distribution found using the iterative algorithm presented in Section 5.4 as well as the uniform and greedy task distribution algorithms described in Section 5.2. Each trial was started after the correct task distribution, if applicable, was found and was stopped when the last result of the computation was received. While we do not include the energy consumption at each device (i.e., E_i in Eq. (5.2)) in the definition of the

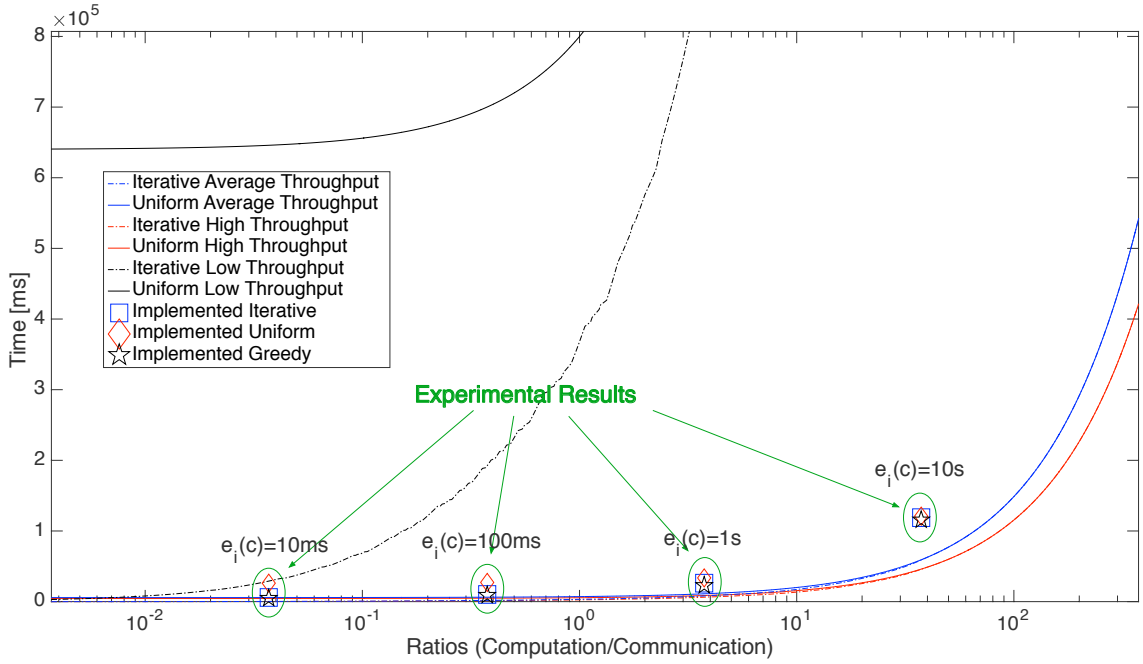


Figure 5.4: Implementation and simulation results for different computation/communication ratios. Result size is 1 MB.

cost vector $\boldsymbol{C}$, when our algorithm finds multiple task assignments that result in the same delay cost, we select the task assignment that minimizes the overall network energy consumption.

In Figures 5.4-5.6 we plot the time required to compute a set of 50 homogeneous tasks as a function of different computation/communication time ratios, for both an Android implementation of the task distribution as well as a simulation of the task distribution using the model described in Section 5.3. In these figures, the dashed lines represent the simulated performance of the task distribution obtained through the iterative algorithm described in Section 5.4, while the continuous lines represent the simulated performance of a uniform task distribution for different values of transmission throughput. In particular, we set $T_l^{\text{tx}} = T_{\bar{l}}^{\text{tx}} \in \{72\text{Mbps}, T_r^{avg}, 0.5\text{Mbps}\}$ in Eq. (5.1), where T_r^{avg} is the average experimental throughput measured for the different data size r (i.e., $T_{10\text{KB}}^{avg} = 0.87$Mbps, $T_{100\text{KB}}^{avg} = 8.4$Mbps and $T_{1\text{MB}}^{avg} = 30$Mbps). The squares, diamonds and stars, instead, represent the results obtained by implementing the Iterative, Uniform and the Greedy task distribution policies in our real network of Android devices, respectively.

The results in Figures 5.4-5.6 show that when the computation takes about 40 times longer than

the communication, a uniform task distribution provides the same performance as both the greedy algorithm as well as our iterative algorithm. Below this point, there are clear benefits in using the task distribution found with our iterative algorithm, with gains that become more evident when the time spent computing is comparable to the communication time. When the communication takes significantly longer than the computation, the iterative algorithm can complete the set of tasks about twice as quickly as the uniform algorithm (for a result size of 1MB and computation time of 10ms). Moreover, these results show that the simulated results can provide a good approximation of the implementation results so long as the appropriate average link throughput are used. Hence, the model developed in Section 5.3 can be used to explore the performance of computation offloading in multi-hop ad hoc networks.

The simple Greedy task distribution is able to adapt to the instantaneous variations in computational time (due to, e.g., operating system operations unrelated to the actual task execution), as well as to the channel impairments that can severely affect the actual throughput of the communication links. As a result, the Greedy task distribution is able to attain performance very close to the simulated performance of our iterative algorithm with the high WiFi Direct throughput. When minimizing the total completion time, thanks to its adaptability to the instantaneous system variations, the Greedy task distribution outperforms the implementation of the iterative task assignment, thus making it the de-facto choice for real device implementations.

To gain further insight into the differences between the Greedy task distribution and our iterative algorithm, in Figure 5.7 we compare the average task assignments that were used in each case. As can be seen in Figure 5.7(a), both the iterative and Greedy task distribution schemes start assigning tasks in a uniform way when the computation time (10s) is much longer than the time spent communicating. When the situation is reversed (i.e., communication is much longer than the 10ms computation), instead, in some cases no tasks are actually assigned to the furthest node (see Figure 5.7(d)). Overall, these results show that the throughput used to compute the task assignment is over estimated, which is particularly evident by the fact that the device that is generating the tasks (i.e, the device at 0 hops), is most of the time computing fewer tasks than when using the Greedy approach.

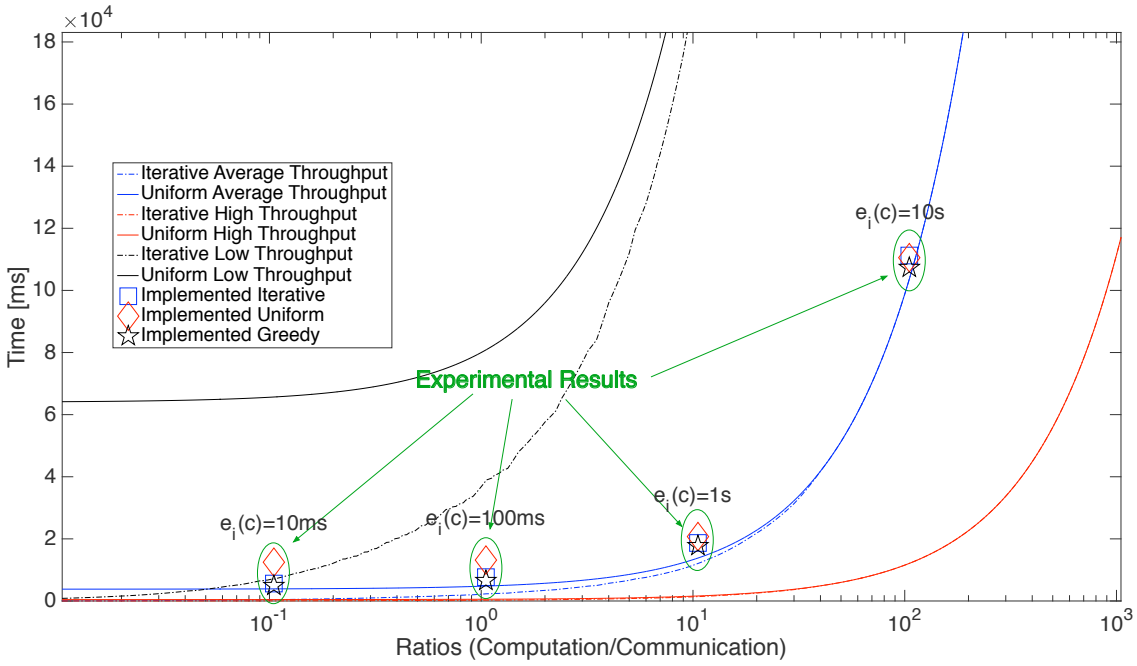


Figure 5.5: Implementation and simulation results for different computation/communication ratios. Result size is 100 KB.

Benefits of Offloading to Multi-hop Neighbors

The results presented in the previous section show that a Greedy task assignment can adapt to changing computation and communication environments and hence can return all the tasks in the least amount of time. However, our iterative algorithm provides a task distribution that is close to that provided by the Greedy algorithm. Additionally, the results in the previous section show that the simulated results match the implementation results and, as a consequence, simulations based on the model from Section 5.3 can be used to provide insight into the performance of the system. Thus, in what follows we use the analytical model and the iterative algorithm described in Sections 5.3 and 5.4 to further explore the gains that can be achieved by extending the distributed computation to all the available network resources.

In particular, in order to highlight the benefits of offloading the computation to all the mobile devices in a network, in Figure 5.8 we compare the gain in time to complete all 50 tasks relative to a task distribution that only utilizes one additional device (i.e., single-hop task distribution as considered in the literature [74, 75, 6]) with: a) offloading the computation to all of the nearest neighbors of the

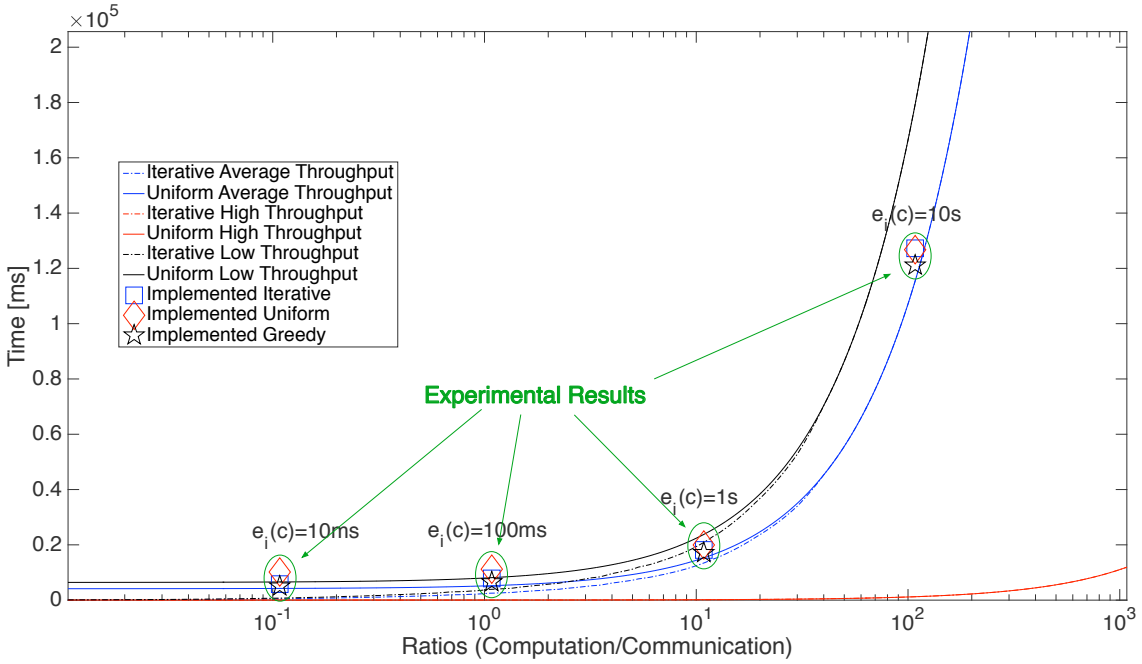


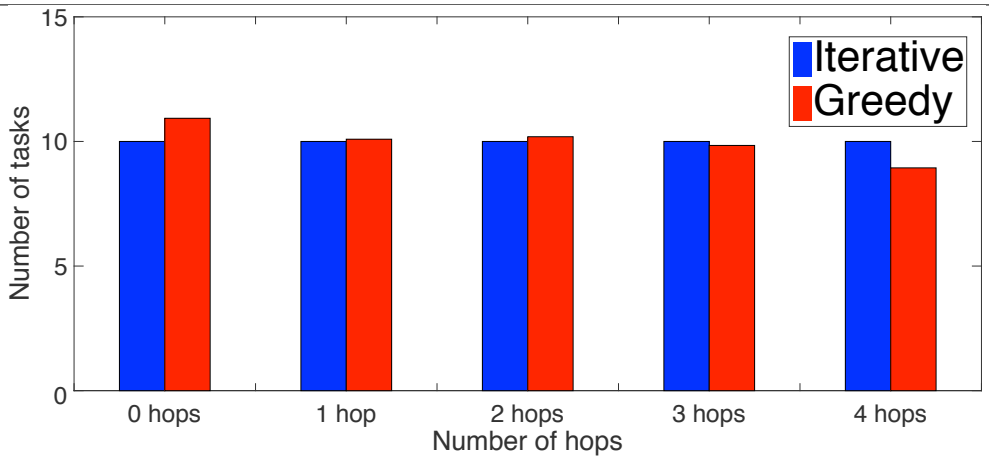
Figure 5.6: Implementation and simulation results for different computation/communication ratios. Result size is 10 KB.

device that is generating the tasks (i.e., Single Group), and b) offloading the computation to a multi-hop network (i.e., 2-hops, 3-hops and 4-hops network). As shown in Figure 5.8, in this simulation we consider a tree-like network with the root node generating the tasks having four children nodes, and each subsequent child servicing one additional node. This network extends for 2, 3 or 4 hops, resulting in a total of 4 devices in the source node's group that can be used for group task distribution, and 8, 12 and 16 devices that can be used for network task distribution.

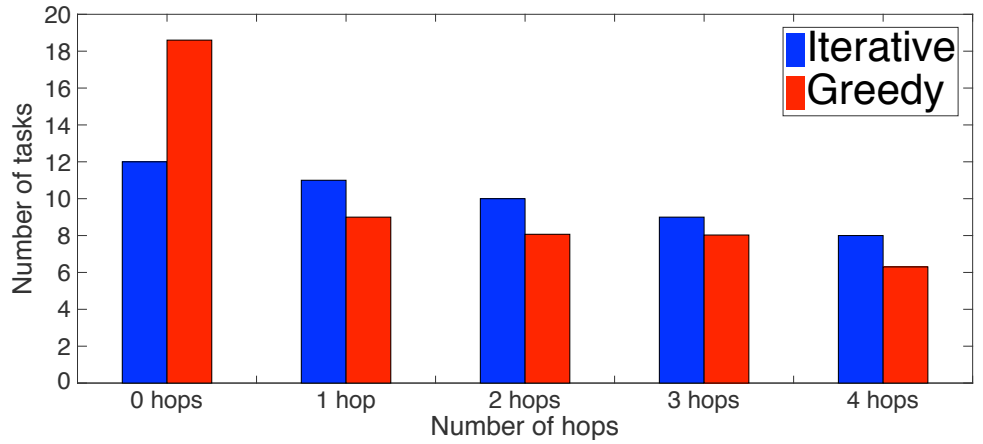
Figure 5.8 clearly shows the benefit of offloading to a multi-hop network. In particular, offloading to the larger network provides up to 30% faster computation time than offloading the tasks only to the first group, and a gain of up to 88% against offloading to only a single device (as is currently supported in the literature).

Network Utilization

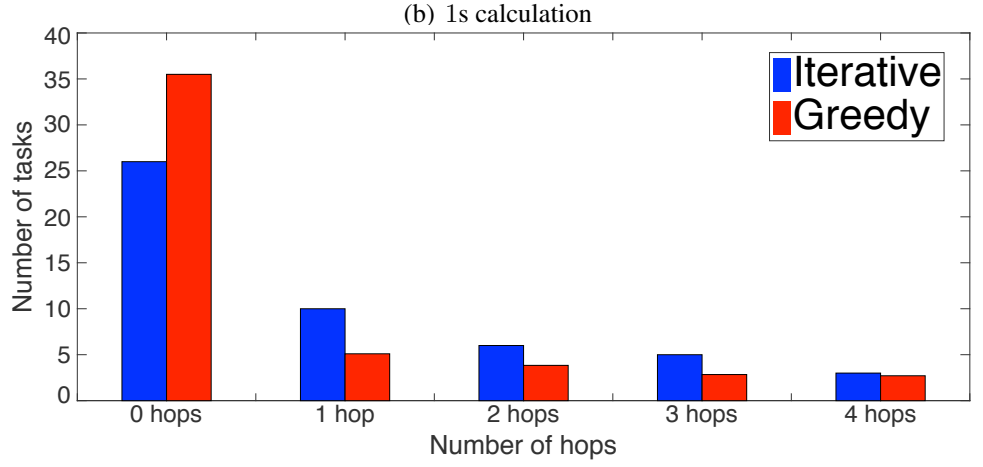
Due to the fact that mobile devices are often energy constrained, it is imperative to consider how offloading computation impacts the network. In this section, we examine the impact of the network in terms of network utilization. More specially, in order to best utilize the network, computation should



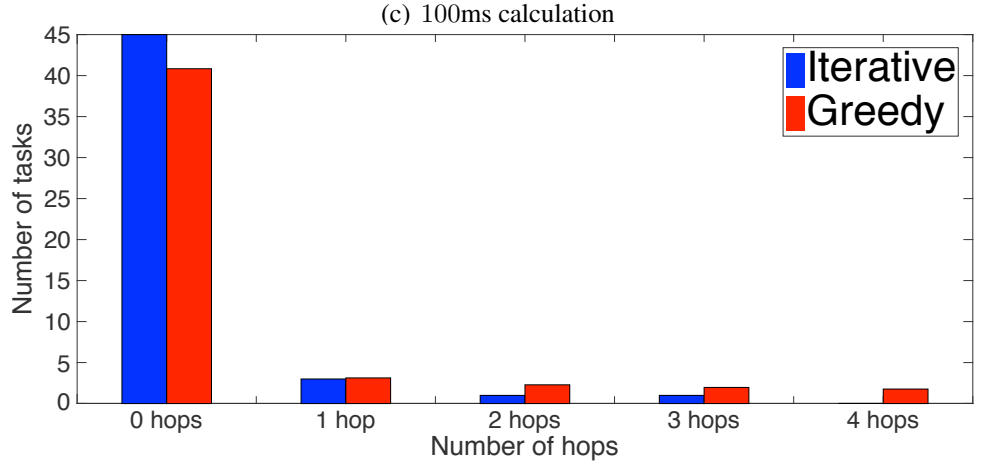
(a) 10s calculation



(b) 1s calculation



(c) 100ms calculation



(d) 10ms calculation

Figure 5.7: Task distribution for the Greedy and Iterative approaches with result size fixed to 1 MB.

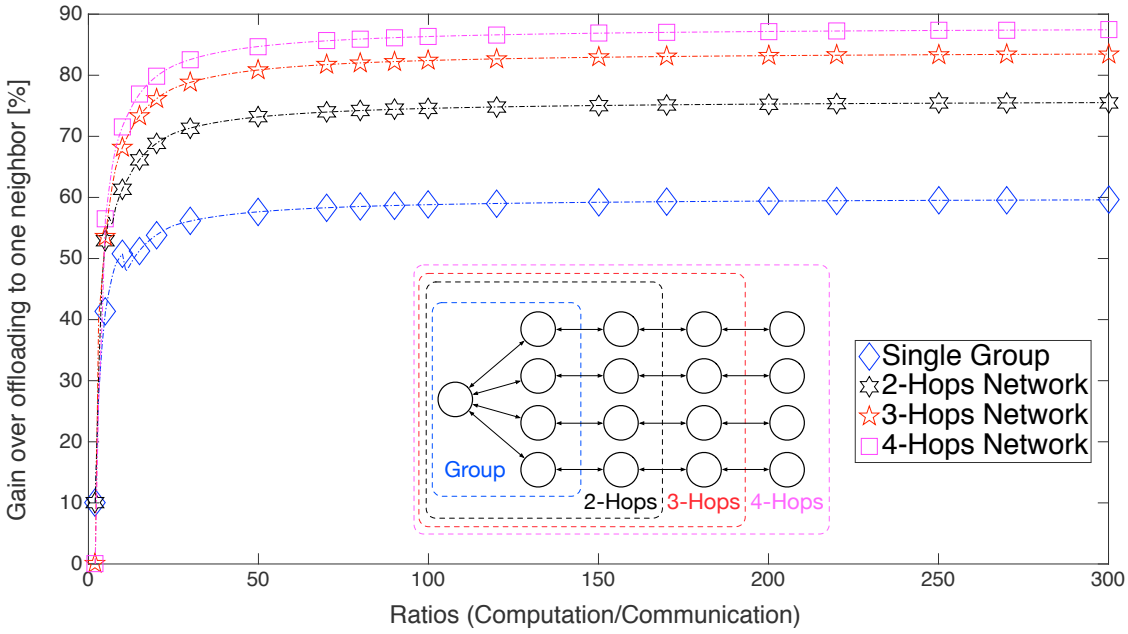


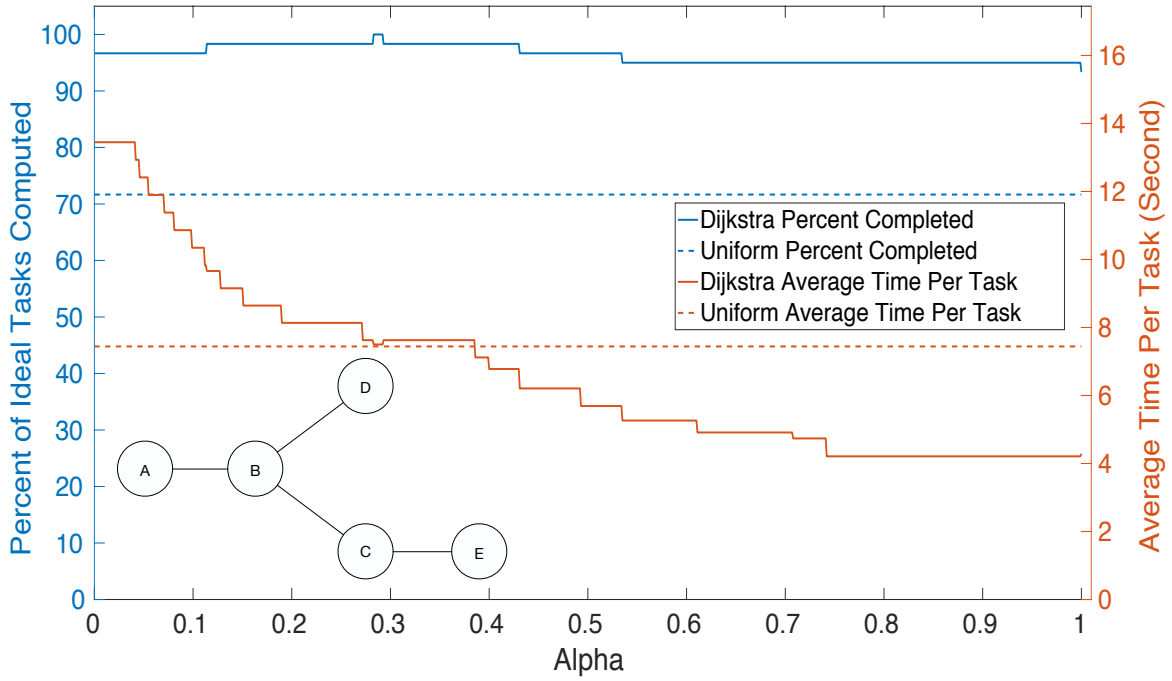
Figure 5.8: Simulated measurement indicating the percent speed up over offloading to only a single neighbor.

be offloaded not only to the devices that have the most residual energy, but also those devices that are not critical to the connectivity of the network. The prior sections indicate that the ratio of the cost to communicate to the cost to compute can impact the assignments made by our iterative algorithm. We use a ratio of $1:1$ when simulating the networks detailed in Figures 5.9(a)-5.11(a). For the purposes of this section, we define performance as a percentage of the maximum number of homogeneous tasks that a network is able to compute [4], as determined by the solution to Eq. (5.9).

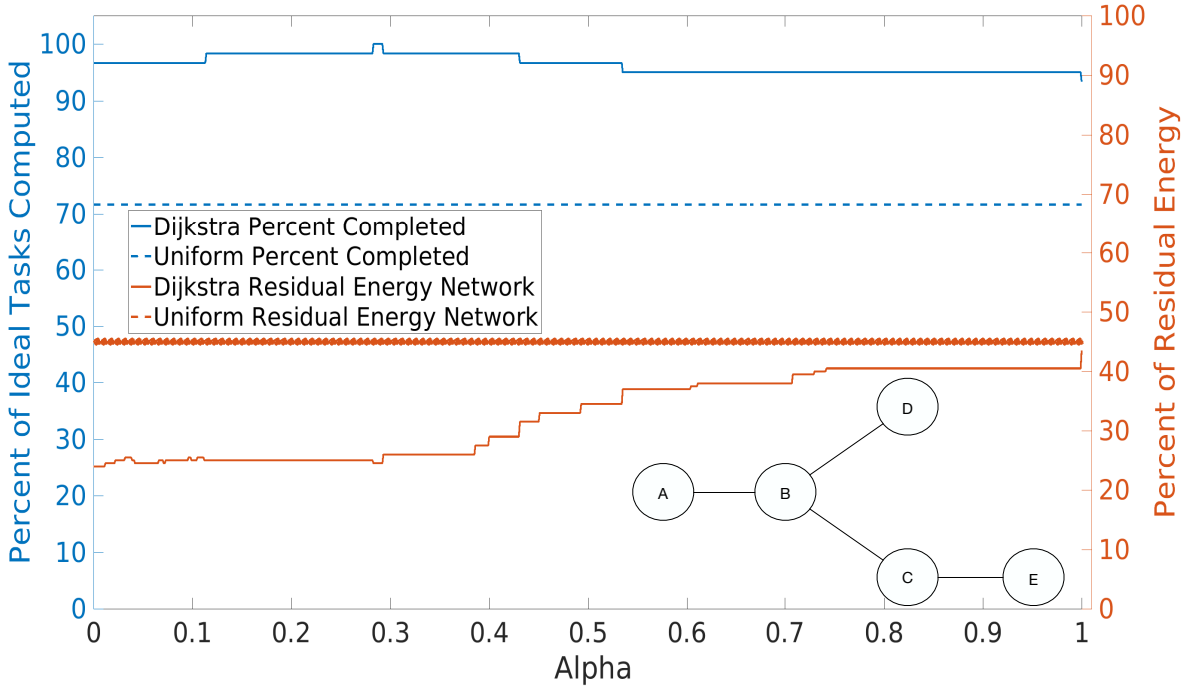
Figures 5.9(a)-5.11(a) plot, in blue, the performance of our iterative algorithm with respect to α, as well as the performance of the uniform algorithm, where the dashed lines represent the performance of the uniform algorithm and the solid lines represent the performance of our iterative algorithm. The results in these figures show that the uniform algorithm is able to compute 72%, 83%, and 73% of the maximum number of tasks. On the other hand, our iterative algorithm is able to compute all of the tasks for $\alpha \in [.284, .293]$, $\alpha \in [0, .316]$, and $\alpha \in [0, .434]$. At its worst, the iterative algorithm is only able to compute 93%, 93%, and 75% of the maximum tasks when $\alpha = 1$, $\alpha = 1$, and $\alpha \in [.892, 1]$.

To gain further insight into the effect α has on the network, Figures 5.9(a)-5.11(b) plot, in red, the

[4]The maximum number of tasks were generated using the technique detailed in Section 5.5.

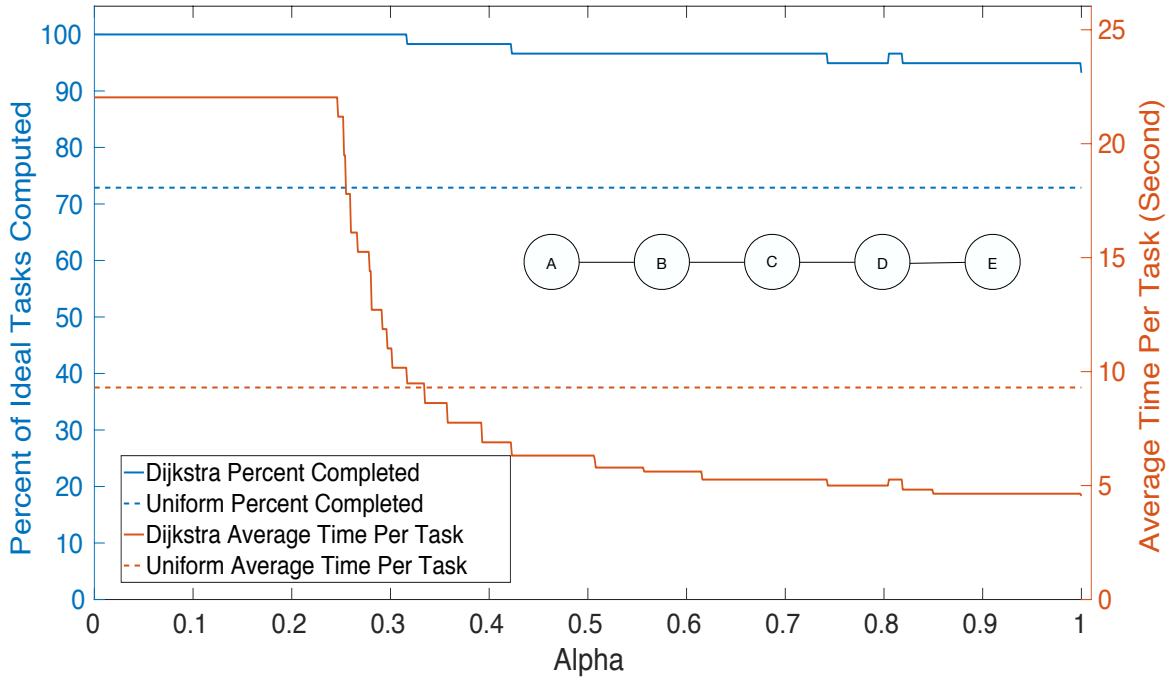


(a) Average time per task.

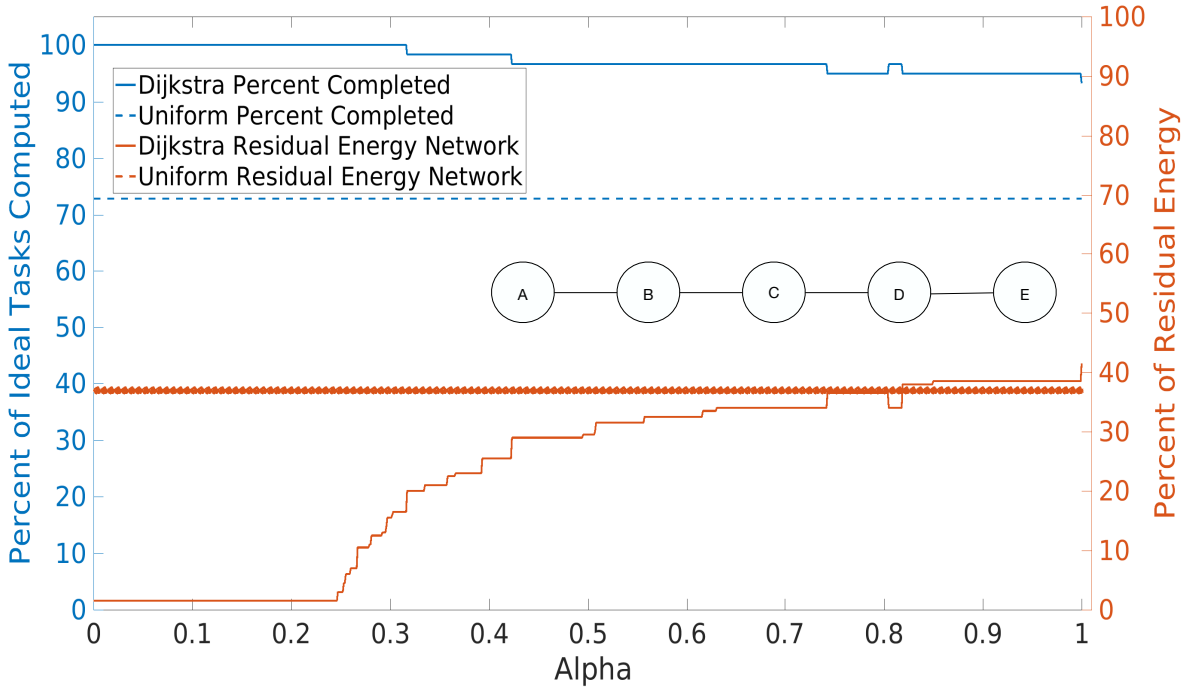


(b) Percentage of energy left in the network.

Figure 5.9: The effect α has on the iterative algorithm's ability compute the maximum number of tasks before network partition.

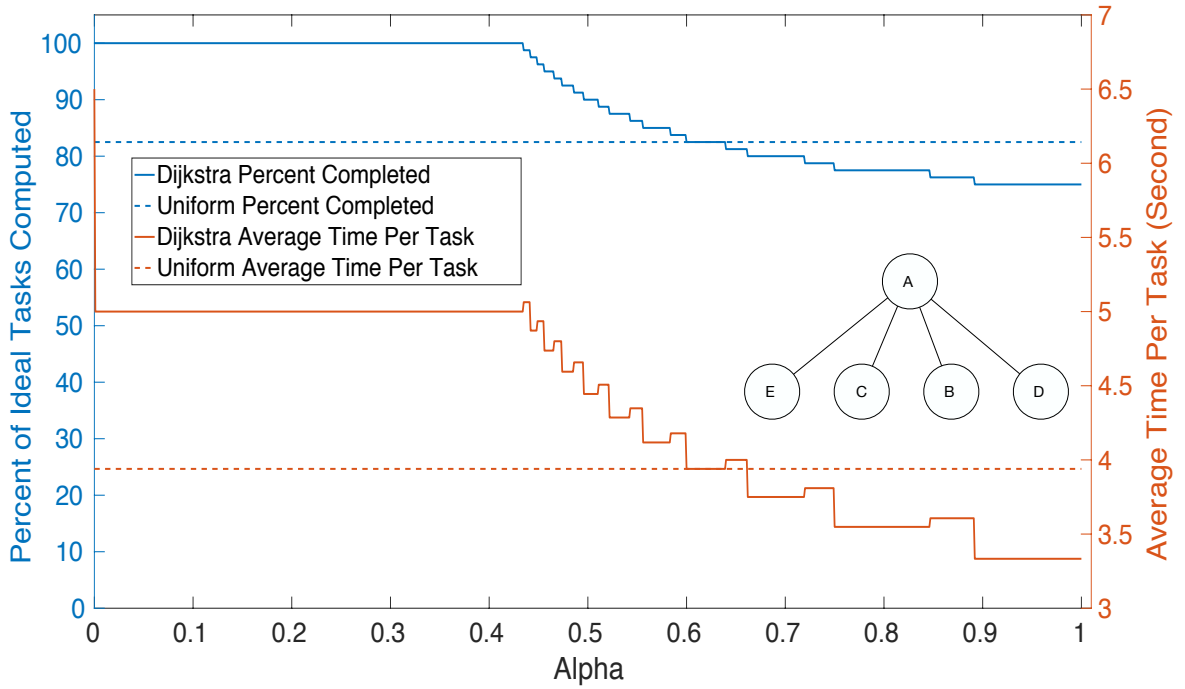


(a) Average time per task.

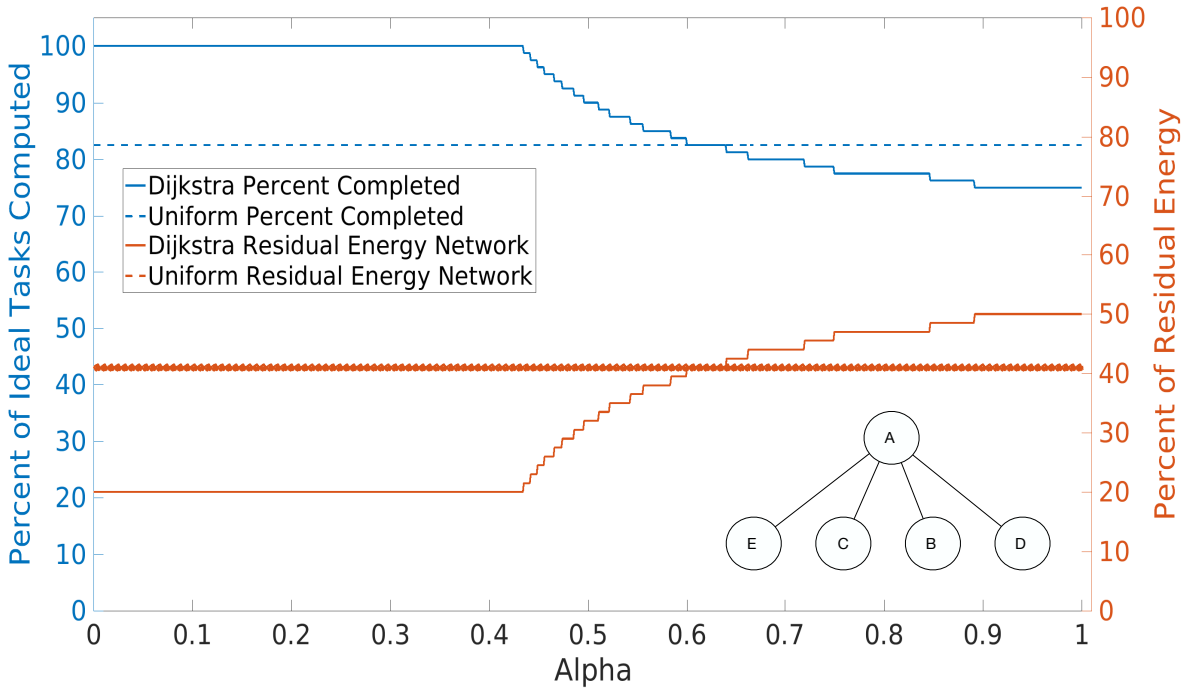


(b) Percentage of energy left in the network.

Figure 5.10: The effect α has on the iterative algorithm's ability compute the maximum number of tasks before network partition.



(a) Average time per task.



(b) Percentage of energy left in the network.

Figure 5.11: The effect α has on the iterative algorithm's ability compute the maximum number of tasks before network partition.

average time per task, and the percentage of residual energy left in the entire network, respectively. From these figures, we can see that for smaller values of α, the average time per task for the iterative algorithm is 81%, 137%, and 65% slower than the uniform case. Additionally, Figures 5.9(b)-5.11(b) indicate that smaller values of α, enable the iterative algorithm to utilize more of the network's energy, leaving the network with 24%, 1.5%, and 20% of its total energy. This is compared to the uniform algorithm, which consumes 45%, 37%, and 41% of the network's total energy. In the other extreme, Figures 5.9(a)-5.11(a) show the iterative algorithm is able to perform an average task either 43%, 51%, or 15% faster than the uniform algorithm for certain values of α. These values correspond to the instances where the iterative algorithm leaves the network with 43%, 44%, and 50% of its initial energy.

From these figures, it is evident that the routing metric is able to utilize the network fully when distributing tasks for many values of α. Additionally, these figures indicate there is a clear trade off between average time per task, and utilizing the network. As a result, the value of α should be chosen appropriately for the particular situation in which the network is operating. Finally, we can see that the performance heavily depends on the network topology. For instance, we can see that the network in Figure 5.11(a) computes fewer tasks than the uniform algorithm, compared to Figures 5.9(a) and 5.10(a) where the iterative algorithm always computes more tasks.

5.6.2 Performance Evaluation: Heterogeneous Tasks

Static Task Set

In the prior sections, we have made the case that offloading computation, using the proposed algorithm presented in Section 5.4, can be beneficial; however, the prior sections still rely on the strong assumption that the computation that is being shared and the devices participating in the computation are homogeneous. For the results in this section, we analyze the performance of our iterative algorithm, relative to the uniform algorithm and the algorithm proposed by Dantzig et al., for heterogeneous tasks. By varying the heterogeneity [5] of the tasks being distributed, we aim to complete our

[5] We vary the heterogeneity by initially distributing a set of homogeneous tasks, and slowly change the set until all of the tasks have a different cost.

analysis of offloading computation in ad hoc networks. We note that the NP nature of this problem only allows us to comment on the particular scenarios we have simulated and general trends exhibited by these simulations.

To this end, we analyze Figures 5.12(a) and 5.12(b), using the performance of the uniform algorithm as a baseline, we are able to understand the relative performance of both our iterative algorithm and the algorithm proposed by Dantzig et al. For these results, we simulated the network, specified in each figure, while distributing sets of 150 tasks with varying heterogeneity. These simulations were performed with a "low" communication cost $(10$ seconds to send and receive the task/result$)$. Additionally, Figure 5.12(a) starts with a homogeneous set of tasks, which require 10 seconds to compute, and varies the heterogeneity of the set by replacing tasks from the heterogeneous set, with tasks that require longer $($bounded to 1500 seconds$)$ to compute. Figure 5.12(a) demonstrates the effect that the heterogeneity of the task set has. More specifically, it is evident that the iterative algorithm is able to complete the set of tasks 47% faster than the uniform algorithm , when the set is homogeneous, and about 8% faster when the set is heterogeneous. On the other hand, the algorithm proposed by Dantzig et al. exhibits the same performance as the uniform algorithm in the homogeneous case , and is able to complete the set of tasks about 10% faster than the uniform algorithm for heterogeneous task sets. It is worth noting that the results in the homogeneous case corroborate the results from Section 5.6.1, due to the fact that the ratio between time to compute and time to communication is 1:1.

Similar to Figure 5.12(a), Figure 5.12(b) simulated the same network and communication cost; however, this figure changed the homogeneous set of tasks to be a set of tasks requiring 1500 seconds to compute. This particular simulation proceeded to substitute these tasks with shorter tasks $($bounded to 10 seconds$)$ as the heterogeneity changed. The results from this simulation indicate that both the iterative algorithm and the algorithm proposed by Dantzig et al. exhibit the same performance as the uniform algorithm in the homogeneous case and each algorithm performs about 10% faster as the set become more heterogeneous. These figures not only reinforce the conclusions drawn in Section 5.6.1, but also demonstrates that as a set becomes more heterogeneous, the algorithm proposed by Dantzig et. al is able to complete the set of tasks the fastest.

For completeness, Figure 5.6.2 simulates the same network, distributing the same sets of tasks,

with communication costing 1500 seconds to transmit and receive results. By comparing Figures 5.12(a) and 5.13(a), we observe that in the communication dominant system, our iterative algorithm is able to compute the set of tasks between 65% and 98% faster than the incumbent algorithms. Moreover, Figures 5.12(b) and 5.13(b) shows that our iterative algorithm is now able to compute the set of tasks between 48% and 65% faster than both the uniform algorithm and the algorithm proposed by Dantzig et al. We note that unlike Figures 5.12(a) and 5.13(a), which maintain similar trends, Figure 5.13(b) exhibits different behavior from Figure 5.12(b). This is likely due to the fact that in the homogeneous case for Figure 5.12(b), the cost to compute is 150 times greater than the cost to communicate, meaning that our iterative algorithm uniformly distributes the tasks to all of the participating devices, whereas in Figure 5.13(b) the cost to compute and the cost to communicate are equal. The examination of all of these figures allude to a trend that the relative cost of communicating impacts each of the algorithm's ability to efficiently distribute tasks. For instance, there exists a set of conditions for which the algorithm proposed by Dantzig et al. is able to distribute the set of tasks more efficiently than our iterative algorithm; however, due to the fact that this algorithm needs to order both the tasks and devices, it must know all of the tasks a priori. In contrast to Dantzig et al.'s algorithm, our iterative algorithm greedily assigns tasks to the device that has the lowest cost, at that particular moment. Furthermore, we note that the Hungarian algorithm [76] is another incumbent algorithm for task distribution schemes; however, this algorithm stipulates that the number of devices and tasks are equal and is therefore not presented in these findings.

Online Task Sets

In Section 5.6.2, we demonstrated the expected performance of our iterative algorithm, the uniform algorithm, and the algorithm proposed in [77] when distributing a full set of tasks; however, seldomly are all of the tasks available at one. Figure 5.6.2 presents the simulated results for the topology detailed in 5.6.2. This simulation focused on the effect that distributing tasks in an "online" manner has on each of the algorithm's ability to efficiently distribute tasks. Therefore, we have focused on the effect that binning tasks has on the over all time required to distribute 150 heterogeneous tasks. We note that the algorithms were applied to each of the bins.

Similar to the prior sections, we have provided the results for both task dominant system, found in 5.14(a) and 5.14(c), as well as a communication dominant system, seen in 5.14(b) and 5.14(d). Figures 5.14(a) and 5.14(b) show the percent speed up our iterative algorithm and the algorithm proposed by Dantzig et al. has over the uniform algorithm. Likewise, Figures 5.14(c) and 5.14(d) plot the absolute time required for each of the algorithms' to compute 150 tasks.

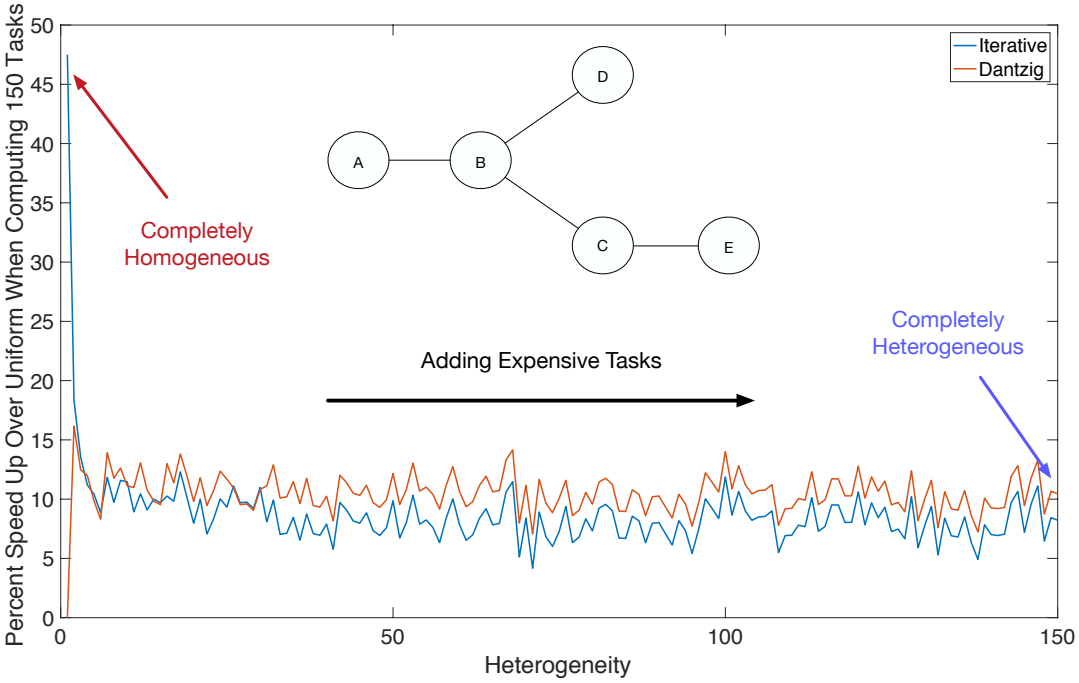
Figure 5.14(c) demonstrates that our iterative algorithm is able to to maintain consistent performance across all bin sizes. Our iterative algorithm is able to account for each of the participating device's current load, allowing for the system to maintain consistent performance. This is in contrast to the other algorithms, which effectively distribute each batch uniformly among the participating devices. We note that when the bin size is equal to 6, the uniform algorithm requires more time. this is due to the fact that the root node ends up calculating an additional task. This is in contrast to the algorithm proposed by Dantzig et al., where the root node computes the quickest task, from that bin, in addition to the longest, thereby acting as an upper bound to the uniform algorithm for this particular scenario.

Similar to the Section 5.6.2, Figure 5.14(b) and 5.14(d) further cement our iterative algorithm's ability to account for the impact that communication has on the system's ability to distribute tasks. For communication dominant systems, both the uniform algorithm and the algorithm proposed by Dantzig et al. incur higher costs due to the majority of the time being spent on communicating the task and result. This notion is reinforced when analyzing bin size of 6 for the communication dominant system, where we can observe a reduction in the amount of time required to compute 150 tasks.

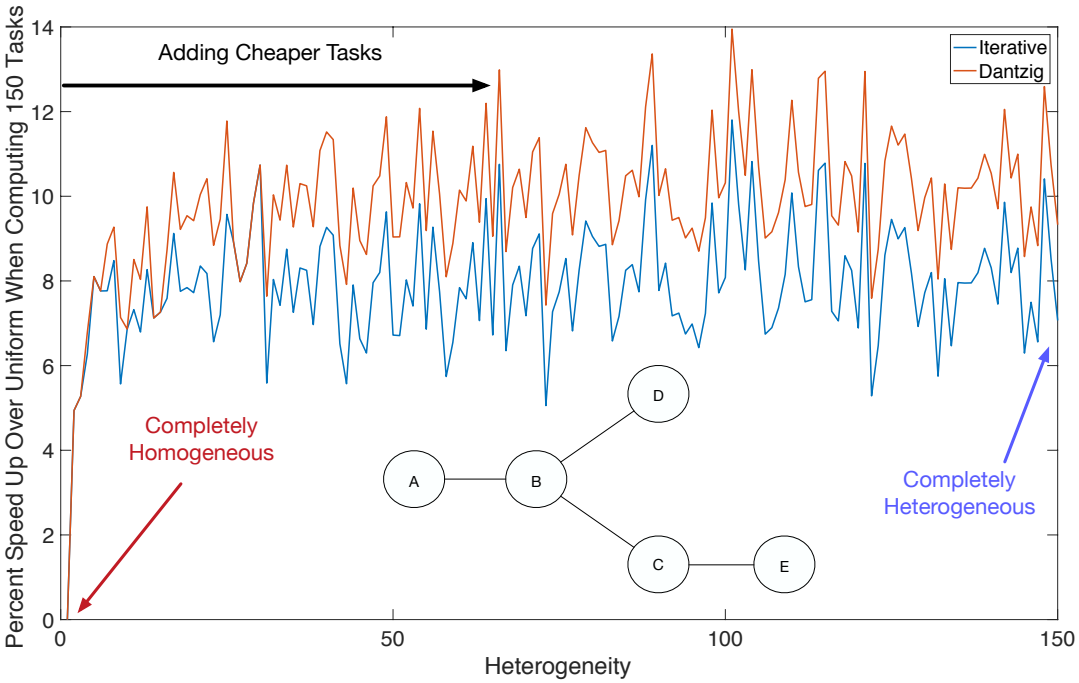
5.7 Conclusions

In this chapter, we explored the benefits of enhancing mobile to mobile computational offloading in multi-hop cooperative networks. By implementing a multi-hop computational offloading system, we were able to implement different task distribution algorithms and verify the accuracy of an analytic model. Using this model, we were able to show the overall benefit of enabling offloading to multi-hop neighbors in a network, which can be quite large when the time to compute is higher than the time to

communicate the data.

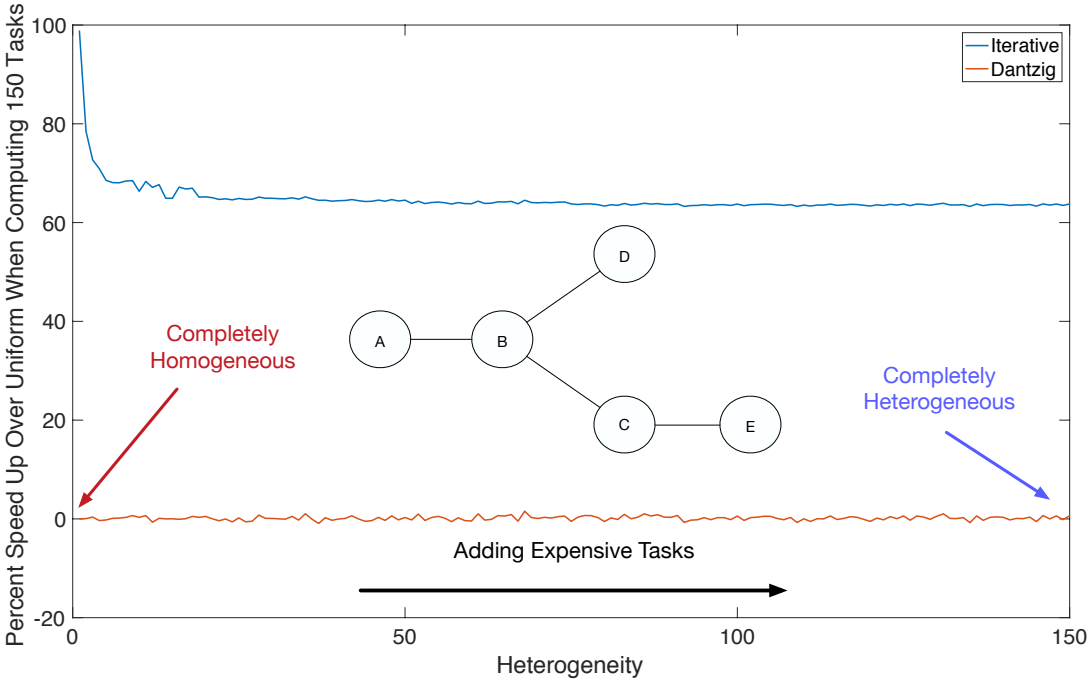


(a) Compute time increases with heterogeneity

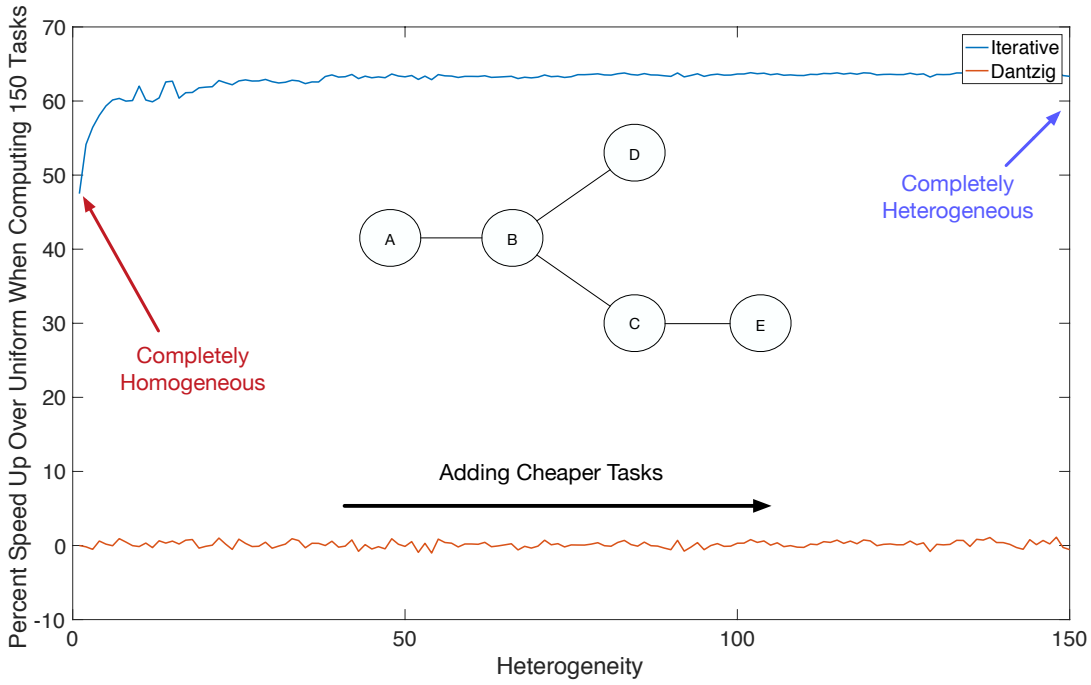


(b) Compute time decreases with heterogeneity

Figure 5.12: Percent speed up when distributing 150 tasks with varying heterogeneity and low communication cost (10s). The homogeneous case is a set of tasks requiring 10s to compute and the heterogeneous case has tasks $\in$ [10s, 1500s].

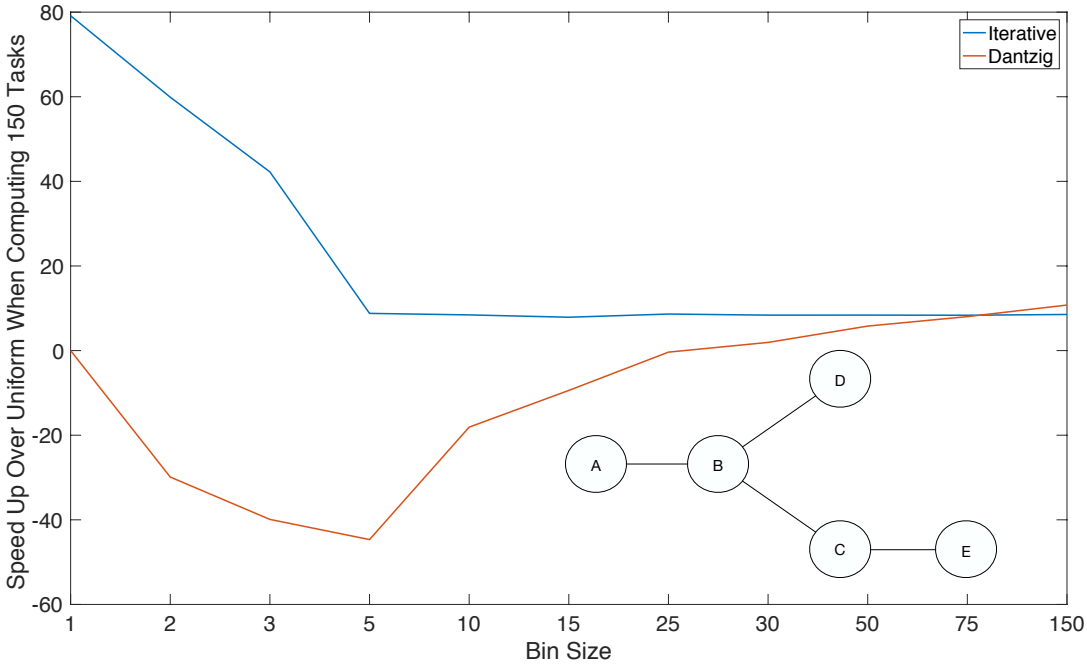


(a) Compute time increases with heterogeneity

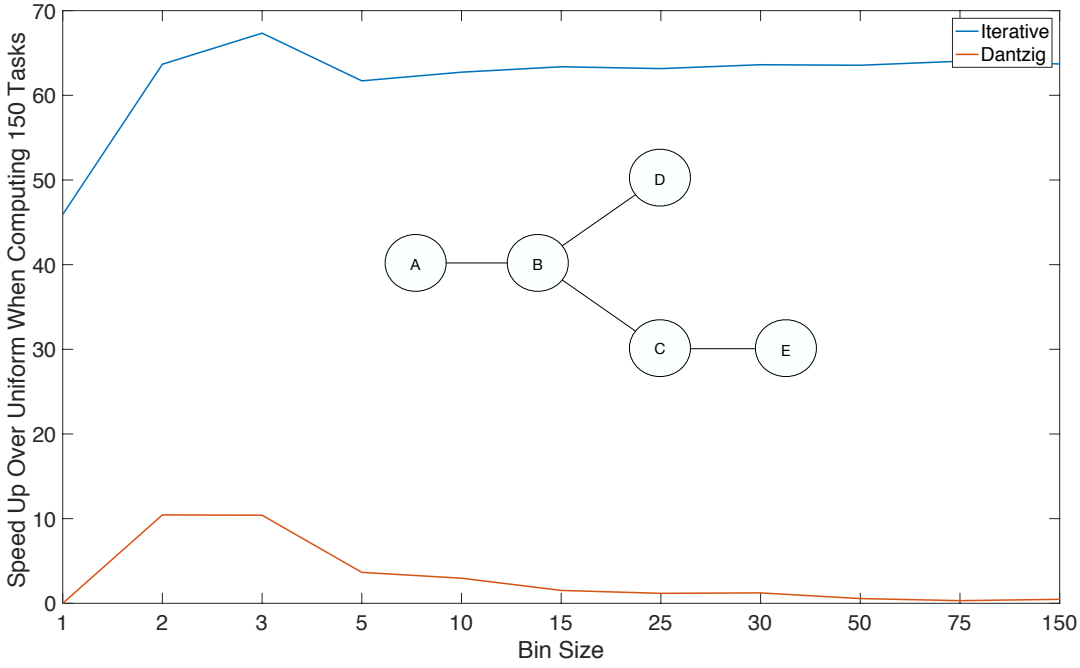


(b) Compute time decreases with heterogeneity

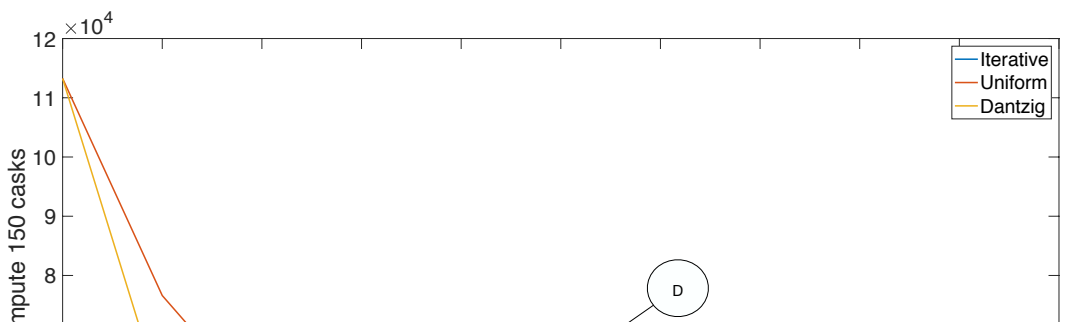
Figure 5.13: Percent speed up when distributing 150 tasks with varying heterogeneity and high communication cost (1500s). The homogeneous case is a set of tasks requiring 10s to compute and the heterogeneous case has tasks $\in$ [10s, 1500s].



(a) Relative performance compared to uniform for low communication cost



(b) Relative performance compared to uniform for high communication cost



Chapter-6

Visualizing Mobile Computing in Ad Hoc Networks

6.1 Introduction

Although an ad hoc network's independence from infrastructure based communication makes it ideal for facilitating communication in military and disaster relief scenarios, these very situations benefit greatly from high situational awareness, which is often not readily available in ad hoc networks. Hence we have created a system to visualize pertinent network information, allowing network operators to quickly identify bottlenecks, locate lost or disconnected nodes, and re-assign network roles in response to a node's impending death due to lack of energy. By coupling this information with geographical information, i.e., displaying the nodes and connections on a map, operators can more easily trouble shoot errors that arise from environmental conditions.

Given these objectives, our visualization system includes the following:

- We display link information, such as received signal strength indication (RSSI) and link speed, by drawing a colored line between two connected nodes. By coloring the link a particular color, based on link information, operators are able to quickly determine if there are environmental conditions that could inhibit communication even if nodes are geographically close together. This provides a good starting point for creating a heat map, which would allow for operators to determine any bottlenecks in a network.

- We display, with an approximation to the nearest quartile, the node's battery level, and we display disconnected nodes in a separate color, with no battery life. By doing so, operators are

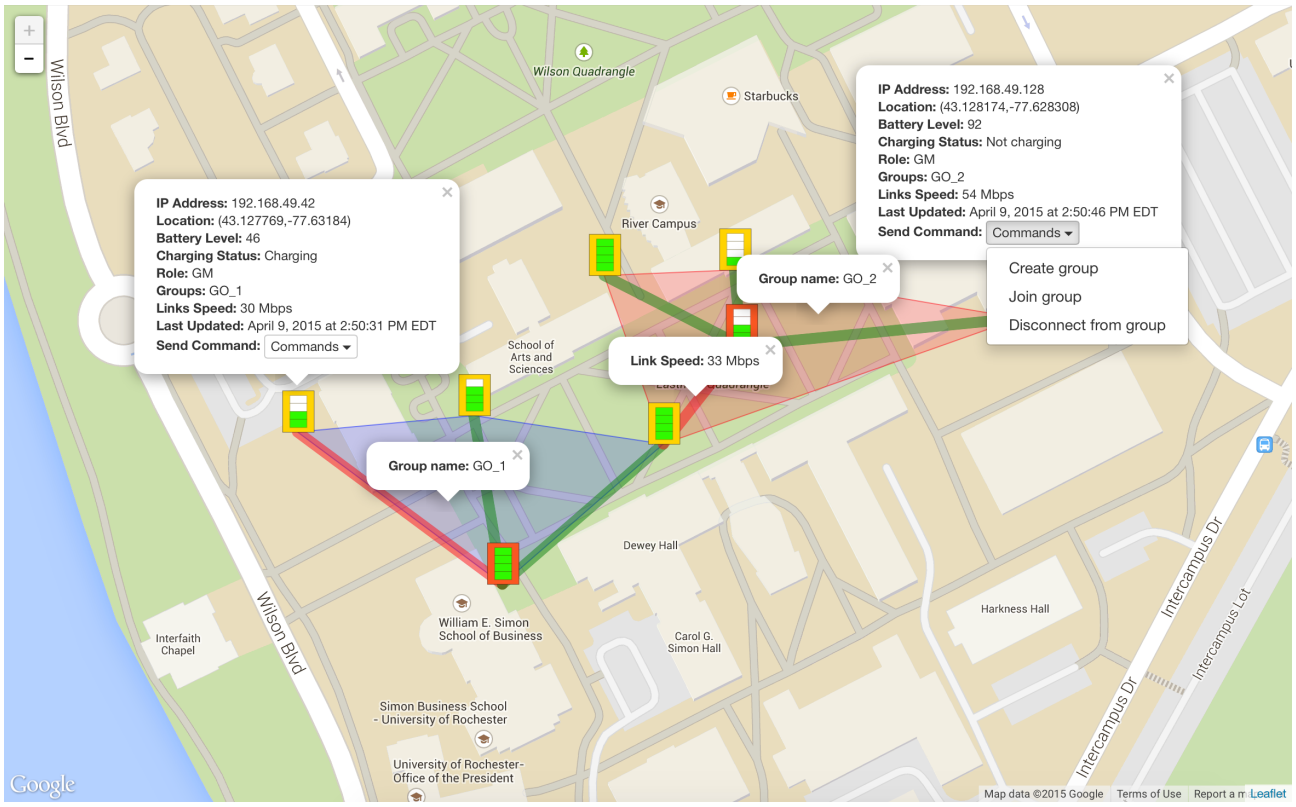


Figure 6.1: Screen shot displaying two groups and various link qualities.

able to visually see which nodes are on the verge of depleting their battery, as well as have a last known location of nodes that have lost contact with the network.

- We provide a method of displaying individual subnets, groups, or clusters, which make up the larger network. We have done so by using a convex hull to connect the various nodes that are members in each subnet. This in turn allows operators to determine which nodes are capable to become a new group owner, in the event that the current one will deplete its energy, or accommodate nodes that may exist outside of the current network.

In order to gather all of this information, we have developed an Android service, which collects the specified information and periodically adds and updates its information in a database that can exist both locally and over a longer latency link. By doing so, any node with privileges to access

Figure 6.2: Screen shot displaying last known location of a node that has lost connection with the network.

the database is able to visualize the network as we have described above. A sample of our system is shown in Figures 6.1 and 6.2.

6.2 Implementation

To bring these goals to fruition, we have developed an Android application to collect node information and route data and commands through our ad hoc network. We have additionally developed a web application to visualize the data received from our ad hoc network.

6.2.1 Android

We start by constructing an ad hoc network using Android's implementation of WiFi Direct. Using Android's WifiP2pManager and WifiManager API, we are able to collect a node's IP address, link speed, RSSI, and role in the network. Additionally, we use the BatteryManager API to collect the battery level as well as Location API to determine the node's GPS coordinates. All of these metrics are combined into a node object, which is in turn sent to each node's respective group owner. The group owner is then able to construct a routing table, which is used to route commands and data, as well as pass the information towards the desired data sink, where the sink uploads all of the network information to a mongoDB data base. This process is executed periodically, with the period specified by the Android program.

6.2.2 Web Application

With the goal of presenting pertinent network information in an easily accessible and visual manner, we developed an application to fulfill these requirements. Our web application must first be able to access the mongoDB database, in order to visualize the stored data. We use Leaflet to handle the map and ploting the nodes on the map. Node.js is used to calculate the convex hull of each group, which is used to help color coordinate each individual group, as well as determine what level of opacity should be used, which is dependant on the amount of time a node has been absent from the network, as well as coloring the links.

Chapter-7

Conclusions and Future Work

7.1 Conclusions

This dissertation addresses some of the fundamental challenges in creating and distributing tasks in mobile clouds comprised of multi-hop ad hoc networks. Our results have shown that mobile clouds can benefit greatly from using all computing resources available in a given network. By constructing suitable task assignments, based on available network resources, we aim to further enhance mobile cloud computing. The contributions of this dissertation are summarized as follows:

- I extended volunteer computing platforms through the use of ad hoc networking protocols and practices. In particular, I have analyzed the impact that task distribution schemes, namely $Proxy$ and $Batch$, have on these systems, as well as created a closed form analytical model of these systems.

- I have designed and provided an in-depth analysis of different approaches for implementing multi-group WiFi Direct networks to enable multi-hop communication in ad hoc mobile clouds. My approaches include designs that will work with off-the-shelf mobile devices as well as optimized communication solutions that require modifying the Android operating system.

- I have explored the opportunities and challenges in offloading computation in a multi-hop wireless network. Additionally, I have found that the proposed iterative algorithm is provably optimal under certain conditions.

- I have developed a system to visualize basic parameters of an ad hoc mobile cloud. This system is able to control the network topology as well as track the location and movement of the participating nodes. By doing so, administrators are not only able to tailor routes and topologies, to avoid communication bottlenecks, but they are also able to anticipate and avoid network partitions, by monitoring the status of crucial relay nodes.

As a result, the work presented here has further enhanced mobile clouds by exploring the necessary techniques required to establish multi-hop computational offloading, as well as providing optimization, which proved not only the viability of offloading to nearby mobile devices, but also provided intuition into the trade-offs in time spent computing and time spent communicating when utilizing multi-hop computational offloading.

7.2 Future Work

While the work presented here provides an understanding of enabling multi-hop wireless computational offloading systems, several directions can be investigated to further improve the findings presented here.

- When enabling multi-hop wireless networks, generalizing the network structure from a "tree" like structure can provide more options for sharing the computation. This more general graph structure can provide more routing options to either avoid partitions or have the network "heal" from partitions.

- For multi-hop computational offloading systems, studying the effect that device mobility has on the task assignment will create a more realistic understanding of the impact of computational offloading in multi-hop networks. Although works like Serendipity [29] have provided a study on the effects mobility has on throughput for Disruption Tolerant Networks, Serendipity does not address any techniques to re-form the network as devices move. The results of this expansion can also be used to provide insight into the benefits of re-forming the network, to keep mobile devices connected, or let the devices leave the system.

- When expanding this work to include more commercial scenarios, privacy and incentive structures become the primary factors for resource sharing among mobile devices to succeed.

- Studying the effects and dependencies associated with distributed computation will also provide further insight as to when and where computation should be offloaded. For instance, can techniques used by out of order processors benefit the overall computation, or does the criticality of a given task affect the assignment?

In conclusion, the contents of this thesis have provided the basis for understanding how computational offloading in ad hoc networks affects the network as a whole. Furthermore, by providing details on how to implement these systems in practice, others can expand this work to new fields and applications.

Bibliography

[1] "International Data Corporation (IDC)." [Online]. Available: http://www.idc.com

[2] "Gartner Inc." [Online]. Available: http://www.gartner.com

[3] "Geekbench 4: Cross-platform processor benchmark," Last time accessed: Febuary 2017. [Online]. Available: http://www.primatelabs.com/geekbench/

[4] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of EuroSys*. New York, NY, USA: ACM, 2011, pp. 301–314.

[5] "Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: http://aws.amazon.com/ec2/

[6] C. Funai, C. Tapparello, H. Ba, B. Karaoglu, and W. Heinzelman, "Extending volunteer computing through mobile ad hoc networking," in *Prof. of IEEE GLOBECOM*, Dec. 2014, pp. 32–38.

[7] "Attached Resource Computer (ARCNET)." [Online]. Available: http://www.arcnet.com/

[8] "The Stone SouperComputer." [Online]. Available: http://www.extremelinux.info/stonesoup/

[9] "Microsoft OneDrive." [Online]. Available: https://onedrive.live.com/

[10] "IBM Cloud." [Online]. Available: http://www.ibm.com/cloud-computing/us/en/

[11] "Great Internet Mersenne Prime Search." [Online]. Available: http://www.mersenne.org

[12] L. Gong, "Jxta: a network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001.

[13] G. Fedak, C. Germain, V. Neri, and F. Cappello, "Xtremweb: a generic global computing system," in *Proc. of IEEE/ACM CCGrid*, 2001.

[14] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. of IEEE/ACM GRID*, Pittsburgh, PA, Nov. 2004.

[15] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[16] "BOINC," BOINC's Homepage, Last time accessed: October 2014. [Online]. Available: http://boinc.berkeley.edu/

[17] T. Phan, L. Huang, and C. Dulan, "Challenge: Integrating mobile wireless devices into the computational grid," in *Proc. of ACM MobiCom*, Atlanta, Georgia, USA, Sept. 2002.

[18] Bluetooth Group, "Specification of the Bluetooth system," June 2010.

[19] Wi-Fi Alliance, P2P Task Group, "Wi-Fi Peer-to-Peer (P2P) Technical Specification, Version 1.2," Dec. 2011.

[20] E. E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Sept. 2009.

[21] J. R. Eastlack, "Extending volunteer computing to mobile devices," Oct. 2011.

[22] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013.

[23] D. C. Chu and M. Humphrey, "Mobile ogsi.net: Grid computing on mobile devices," in *Proc. IEEE/ACM GRID*, Pittsburgh, PA, Nov. 2004.

[24] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Computing while charging: Building a distributed computing infrastructure using smartphones," in *Proc. of ACM CoNEXT*, Nice, France, Dec. 2012.

[25] K. B. Parmar, N. N. Jani, P. S. Shrivastav, and M. H. Patel, "jUniGrid: A simplistic framework for integration of mobile devices in heterogeneous grid computing," *International Journal of Multidisciplinary Sciences and Engineering*, vol. 4, no. 1, pp. 10–15, Jan. 2013.

[26] H. Xu, M. Bilec, L. Schaefer, A. Landis, and A. Jones, "Ocelot: A wireless sensor network and computing engine with commodity palmtop computers," in *Proc. of IGCC*, Arlington, VA, USA, June 2013.

[27] S. Schildt, F. Busching, E. Jorns, and L. Wolf, "Candis: Heterogenous mobile cloud framework and energy cost-aware scheduling," in *Proc. of IEEE GreenCom*, Beijing, China, Aug. 2013.

[28] P. Datta, S. Dey, H. Paul, and A. Mukherjee, "ANGELS: A framework for mobile grids," in *Proc. of AIMoC*, Kolkata, India, Feb. 2014, pp. 15–20.

[29] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proc. of MobiHoc*, Hilton Head, SC, USA, June 2012.

[30] N. Fernando, S. Loke, and W. Rahayu, "Honeybee: A programming framework for mobile crowd computing," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2013, vol. 120, pp. 224–236.

[31] P. Jassal, K. Yadav, A. Kumar, V. Naik, V. Narwal, and A. Singh, "Unity: Collaborative downloading content using co-located socially connected peers," in *Proc. of IEEE PERCOM*, San Diego, CA, USA, Mar. 2013.

[32] R. Agarwal, "DRAP: A decentralized public resourced cloudlet for ad-hoc networks," Mar. 2014.

[33] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[34] "BOINC on Android," Last time accessed: October 2014. [Online]. Available: http://boinc.berkeley.edu/trac/wiki/AndroidBoinc

[35] "Boincoid." [Online]. Available: http://boincoid.sourceforge.net

[36] "AndroBOINC - BOINC manager for Android phones." [Online]. Available: https://code.google.com/p/androboinc/

[37] "NativeBOINC." [Online]. Available: http://http://nativeboinc.org

[38] M. Black and W. Edgar, "Exploring mobile devices as grid resources: Using an x86 virtual machine to run BOINC on an iPhone," in *Proc. of IEEE/ACM Grid Computing*, Banff, AB, Canada, Oct. 2009.

[39] "HTC Power to Give." [Online]. Available: http://www.htc.com/us/go/power-to-give/

[40] "The 10th BOINC workshop - BOINC/Android status and plans," Sept. 2014. [Online]. Available: http://boinc.berkeley.edu/trac/attachment/wiki/WorkShop14/boinc_on_android_2014.pdf

[41] E. Cuervo, P. Gilbert, B. Wu, and L. Cox, "CrowdLab: An architecture for volunteer mobile testbeds," in *Proc. of COMSNETS*, Bangalore, India, Jan. 2011.

[42] A. D. Zayas and P. M. Gómez, "A testbed for energy profile characterization of ip services in smartphones over live networks," *Mob. Netw. Appl.*, vol. 15, no. 3, pp. 330–343, June 2010.

[43] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen, "Phonelab: A large programmable smartphone testbed," in *Proc. of ACM SENSEMINE*, Roma, Italy, Nov. 2013.

[44] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Seattle: A platform for educational cloud computing," in *Proc. of ACM SIGCSE*, Chattanooga, TN, USA, Mar. 2009.

[45] Y. Zhuang, A. Rafetseder, and J. Cappos, "Experience with Seattle: A community platform for research and education," in *Proc. of GREE*, Salt Lake City, UT, USA, Mar. 2013.

[46] H. Ba, W. Heinzelman, C.-A. Janssen, and J. Shi, "Mobile computing - A green computing resource," in *Proc. of IEEE WCNC*, Shanghai, China, Apr. 2013.

[47] E. Chen, S. Ogata, and K. Horikawa, "Offloading Android applications to the cloud without customizing Android," in *Proc. of IEEE PerCom*, Lugano, Switzerland, Mar. 2012.

[48] T. Soyata, R. Muraleedharan, S. Ames, J. Langdon, C. Funai, M. Kwon, and W. Heinzelman, "Combat: mobile-cloud-based compute/communications infrastructure for battlefield applications," in *Proc. of SPIE*, Baltimore, USA, Apr. 2012.

[49] G. Hiertz, S. Max, Y. Zang, T. Junge, and D. Denteneer, "IEEE 802.11s MAC fundamentals," in *Proc. of IEEE MASS*, 2007.

[50] E. Ferro and F. Potorti, "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison," *IEEE Wireless Commun.*, vol. 12, no. 1, pp. 12–26, Feb. 2005.

[51] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with Wi-Fi Direct: overview and experimentation," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 96–104, June 2013.

[52] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. of ACM MobiSys*, Low Wood Bay, Lake District, UK, June 2012.

[53] G. P. Perrucci, F. H. P. Fitzek, and J. Widmer, "Survey on energy consumption entities on the smartphone platform," in *Proc. of IEEE VTC*, Budapest, Hungary, May 2011.

[54] R. Friedman, A. Kogan, and Y. Krivolapov, "On power and throughput tradeoffs of WiFi and Bluetooth in smartphones," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1363–1376, July 2013.

[55] "Asus Nexus 7 (2013)." [Online]. Available: http://www.asus.com/Tablets_Mobile/Nexus_7_2013/

[56] "Arduino uno." [Online]. Available: http://arduino.cc

[57] D. Rakhmatov and S. Vrudhula, "Energy management for battery-powered embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 2, no. 3, pp. 277–324, Aug. 2003.

[58] M. Coleman, C. K. Lee, C. Zhu, and W. Hurley, "State-of-charge determination from EMF voltage estimation: Using impedance, terminal voltage, and current for lead-acid and lithium-ion batteries," *IEEE Trans. Ind. Electron.*, vol. 54, no. 5, pp. 2550–2557, Oct 2007.

[59] "Iperf." [Online]. Available: https://iperf.fr

[60] 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 *Proc. of ACM MobiSys*. New York, NY, USA: ACM, 2010, pp. 49–62.

[61] "Android Developers - Wi-Fi Peer-to-Peer." [Online]. Available: http://developer.android.com/guide/topics/connectivity/wifip2p.html

[62] "Android API Guides." [Online]. Available: https://developer.android.com/guide/index.html

[63] "RFC1122, Requirements for Internet Hosts – Communication Layers." [Online]. Available: http://tools.ietf.org/html/rfc1122

[64] C. Casetti, C. F. Chiasserini, L. Curto Pelle, C. Del Valle, Y. Duan, and P. Giaccone, "Content-centric routing in Wi-Fi Direct multi-group networks," *ArXiv e-prints*, Dec. 2014.

[65] M. Conti, F. Delmastro, G. Minutiello, and R. Paris, "Experimenting opportunistic networks with WiFi Direct," in *Wireless Days (WD)*, Valencia, Spain, Nov. 2013.

[66] C. Funai, C. Tapparello, H. Ba, B. Karaoglu, and W. Heinzelman, "Mobile to mobile computational offloading in multi-hop cooperative networks," in *Prof. of IEEE GLOBECOM*, Dec. 2016.

[67] M. T. Flynn, M. F. Pottinger, and P. D. Batchelor, "Fixing intel: A blueprint for making intelligence relevant in afghanistan," DTIC Document, Tech. Rep., 2010.

[68] B. Raj, T. Jayakumar, and B. Rao, "Non-destructive testing and evaluation for structural integrity," *Sadhana*, vol. 20, no. 1, pp. 5–38, 1995.

[69] C. Funai, C. Tapparello, and W. Heinzelman, "Enabling multi-hop ad hoc networks through wifi direct multi-group networking," in *Prof. of IEEE*, Jan. 2017.

[70] H. Kellerer and U. Pferschy, "A new fully polynomial time approximation scheme for the knapsack problem," *Journal of Combinatorial Optimization*, vol. 3, no. 1, pp. 59–71, 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1009813105532

[71] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot generalized task assignment problem," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 4765–4771.

[72] M. Racer and M. M. Amini, "A robust heuristic for the generalized assignment problem," *Annals of Operations Research*, vol. 50, no. 1, pp. 487–503, 1994. [Online]. Available: http://dx.doi.org/10.1007/BF02085655

[73] D.-M. U. Derigs and U. Zimmermann, "An augmenting path method for solving linear bottleneck assignment problems," *Computing*, vol. 19, no. 4, pp. 285–295, 1978.

[74] N. Fernando, S. Loke, and W. Rahayu, "Mobile crowd computing with work stealing," in *Proc. of NBiS*, Melbourne, Australia, Sept. 2012.

[75] T. Penner, A. Johnson, B. Van Slyke, M. Guirguis, and Q. Gu, "Transient clouds: Assignment and collaborative execution of tasks on mobile devices," in *Prof. of IEEE GLOBECOM*. IEEE, 2014, pp. 2801–2806.

[76] H. W. Kuhn, *The Hungarian Method for the Assignment Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 29–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68279-0_2

[77] G. B. Dantzig, "Discrete-variable extremum problems," *Operations Research*, vol. 5, no. 2, pp. 266–288, 1957. [Online]. Available: http://dx.doi.org/10.1287/opre.5.2.266

7.3 Proof of Theorem 5.4.1

Theorem 2. *We will prove this in two steps. In the first step, we consider the case where Γ^K is the only optimal task assignment for K tasks, while in the second step, we consider the case of multiple K task assignments that have the same optimal cost.*

For the case where there is a unique optimal solution to distribute K tasks, let's assume by contradiction that Γ^{K+1} determined according to the above definition is not optimal. This, in turn, means that there exists a $K+1$ tasks assignment Θ^{K+1}, such that $\Theta^{K+1} \notin \Psi^{K+1}$ and

$$\mathrm{F}(\Theta^{K+1}) < \mathrm{F}(\Gamma^{K+1}). \tag{7.1}$$

Moreover, we can consider Θ^{K+1} to be derived from a K tasks assignment Θ^K, such that $F(\Gamma^K) < \mathrm{F}(\Theta^K)$ and

$$\mathrm{F}(\Theta^K) \leq \mathrm{F}(\Theta^{K+1}). \tag{7.2}$$

As a result, by combining Eq. (7.1) *and Eq.* (7.2)*, we obtain*

$$\mathrm{F}(\Theta^K) < \mathrm{F}(\Gamma^{K+1}). \tag{7.3}$$

On the other end, assuming that $F(\Theta^K) = \Theta_m^K C_m$ (i.e., device m is the bottleneck device that determines the overall cost of the non-optimal task assignment Θ^K), we have that $\Gamma_m^K < \Theta_m^K$. Thus, $\Gamma_m^K + 1 \leq \Theta_m^K$. Now, if $\Gamma_m^K + 1 < \Theta_m^K$, then $\mathrm{F}(\Gamma^K + e^m) = \mathrm{F}(\Gamma^K) < \mathrm{F}(\Theta^{K+1})$, which contradicts the hypothesis in Eq. (7.1)*. If $\Gamma_m^K + 1 = \Theta_m^K$, instead, $\mathrm{F}(\Gamma^K + e^m) = \mathrm{F}(\Theta^K) \leq \mathrm{F}(\Theta^{K+1})$, which still contradicts the hypothesis in Eq.* (7.1)*. As a result, if Γ^K is the unique optimal K tasks assignment, then the $K+1$ task assignment Γ^{K+1} obtained by Theorem 5.4.1 is an optimal $K+1$ tasks assignment.*

For the case where $\mathrm{F}(\Gamma^K) = \mathrm{F}(\Theta^K)$, there exist at least one Γ_m^K, $m = 1, \ldots, N$ such that $\Gamma_m^K < \Theta_m^K$ since $\Gamma^K \neq \Theta^K$. Thus, $\Gamma_m^K + 1 \leq \Theta_m^K$ which, following an argument similar to the one describe above, results in $\mathrm{F}(\Gamma^{K+1}) = \mathrm{F}(\Gamma^K + e^m) \leq \mathrm{F}(\Theta^{K+1})$, which contradicts the hypothesis in Eq. (7.1)*.*

Therefore, Γ^{K+1} obtained by Theorem 5.4.1 is an optimal $K+1$ tasks assignment.

7.4 Proof of Theorem 5.4.2

Theorem 3. *In what follows, we will prove by induction on the number of tasks M that the solution determined by Algorithm 1 is an optimal solution to the optimization problem defined in Eq. (5.5). For the case $M = 1$, the set of all possible tasks assignment is represented by Φ = {$[1, 0, ..., 0]$, $[0, 1, ..., 0]$, ..., $[0, 0, ..., 1]$}; as a result, we can easily see that*

$$C^1_{opt} = \min_{\phi \epsilon \Phi} \left[\max_{i=1,...,N} \{\phi_i C_i\} \right] = \min(\overrightarrow{1} \circ \mathbf{C}), \tag{7.4}$$

where $\overrightarrow{1}$ is a vector of all ones. Thus, C^1_{opt} can further be simplified to

$$C^1_{opt} = C_i, \tag{7.5}$$

where $i = \text{argmin}(\overrightarrow{1} \circ \mathbf{C})$.

Now to prove that our algorithm is capable of yielding an optimal solution for $M = 1$ task, let C_{algo} be the cost of the solution produced by the algorithm, defined as

$$C^1_{algo} = \max_{i=1,...,N} \left\{ X^1_i C_i \right\}. \tag{7.6}$$

Since $\mathbf{X^0}$ is initialized to a vector of zeros, as shown in step 2 of our algorithm, C^1_{algo} simplifies to

$$C^1_{algo} = \max \left[\{(X^0_i + 1) C_i\} \bigcup \{(X^0_j + 0) C_j | \forall j \neq i\} \right], \tag{7.7}$$

where X^0_i is the device which is chosen at steps 4 and 5 of our algorithm, and $\mathbf{X^0_j}$ is the subset of $\mathbf{X^0}$ that were not chosen at step 4 (i.e., by the argmin function). Eq. (7.7) can further be rewritten as

$$C^1_{algo} = \max \left[\{1 \cdot C_i\} \bigcup \{0 \cdot C_j | \forall j \neq i\} \right] = C_i, \tag{7.8}$$

which, compared with Eq. (7.5)*, shows that* C^1_{algo}=C^1_{opt}.

Let's now assume that $C^K_{algo} = C^K_{opt}$ *up to* $M = K$*, and that* $\boldsymbol{X}^K = \Gamma^K$*, where* Γ^K *is the optimal solution for* $M = K$*. For the case where tasks* $M = K + 1$ *we can write*

$$C^{K+1}_{opt} = \min_{\phi \epsilon \Phi} \left[\max_{i=1,...,N} \{\phi_i C_i\} \right]. \tag{7.9}$$

However, as shown in Theorem 5.4.1, we can restrict the search of the optimal solution to the set Ψ^{K+1}*, which has reduced cardinality to* N*. This means that we can rewrite Eq.* (7.9) *as*

$$C^{K+1}_{opt} = \min_{\psi \epsilon \Psi^{K+1}} \left[\max_{i=1,...,N} \{\psi_i C_i\} \right], \tag{7.10}$$

which in turn can be expanded and ultimately yields three possible outcomes, i.e.,

$$C^{K+1}_{opt} = \min \begin{bmatrix} \Gamma^K_g C_g | \forall l S.T. (\Gamma^K_l + 1) C_l = \Gamma^K_g C_g \\ (\Gamma^K_g + 1) C_g \\ (\Gamma_h + 1) C_h | \forall h S.T. (\Gamma^K_h + 1) C_h > \Gamma^K_g C_g \end{bmatrix}, \tag{7.11}$$

where $\Gamma^K_g C_g$ = C^K_{opt}*, which can be rewritten as*

$$C^{K+1}_{opt} = \max_{i=1,...,N} \left[\{(\Gamma^K_i + 1) C_i\} \bigcup \{(\Gamma^K_j + 0) C_j | \forall j \neq i\} \right], \tag{7.12}$$

where $(\Gamma^K_i + 1) C_i$*,* $i = \text{argmin}((\Gamma^K + 1)\mathbf{C})$.

Now for the solution determined by the algorithm, we can write:

$$C^{K+1}_{algo} = \max\{X^{K+1}_i C_i | i = 1, ..., N\}, \tag{7.13}$$

which in turn can be simplified to

$$C^{K+1}_{algo} = \max_{i=1,...,N} \left[\{(X^K_i + 1) C_i\} \bigcup \{(X^K_j + 0) C_j | \forall j \neq i\} \right], \tag{7.14}$$

which, combined with Eq. (7.12)*, returns* $C^{K+1}_{algo} = C^{K+1}_{opt}$*. Thus, according to Theorem 5.4.1, the* $K + 1$ *tasks assignment determined by the algorithm is guaranteed to be optimal, which concludes*

the proof.