Towards Performance Guarantees in Emerging Wireless Network Applications

A Dissertation presented

by

Arani Bhattacharya

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2019

Stony Brook University

The Graduate School

Arani Bhattacharya

We, the dissertation committe for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation

Samir R. Das
Professor, Department of Computer Science

Himanshu Gupta
Associate Professor, Department of Computer Science

Jie Gao
Professor, Department of Computer Science

Petar M. Djuric
Professor, Department of Electrical and Computer Engineering, Stony Brook University

Pradipta De
Assistant Professor, Department of Computer Sciences, Georgia Southern University

This dissertation is accepted by the Graduate School

Eric Wertheimer
Dean of the Graduate School

Abstract of the Dissertation

Towards Performance Guarantees in Emerging Wireless Network Applications

by

Arani Bhattacharya

Doctor of Philosophy

in

Computer Science

Stony Brook University

2019

The growing interest in mobile systems and Internet-of-Things (IoT) has engendered complex technical challenges ranging from efficient utilization of radio spectrum to developing applications on platforms widely varying in computational power and connectivity. Scalability is increasingly emphasized with exploding number of connected devices, complexity of applications and network data demands with proportionate pressure on limited radio spectrum resources. Our work picks two specific problem domains and explores algorithms that provide performance bounds while scaling to large problem instances. The domains we target are related to distributed radio spectrum monitoring and computation offloading.

In distributed spectrum monitoring we target the spectrum patrolling problem where unauthorized transmitters are localized using a distributed set of spectrum sensors. We specifically consider a crowdsourcing model where a large number of inexpensive sensors are deployed to monitor the radio spectrum. We address different versions of transmitter detection and localization problems, specifically considering the limited budget for the sensors. We develop algorithms to reduce the cost of running a crowdsourced spectrum monitoring system and improve the accuracy of transmitter detection and localization. We also develop FPGA-based spectrum sensors and benchmark

the performance improvements in terms of lower latency and energy consumptions than conventionally used sensors that use commodity embedded processor boards.

Our second problem domain is related to computation offloading from weakly powered devices to more powerful cloud servers. Because of the uncertainty inherent in wireless connectivity, and the presence of a large number of devices, deciding which part of the computation to offload or where to offload is challenging. To address this, we propose algorithms that optimize the process of offloading. We focus on providing probabilistic guarantees on the performance of offloaded applications in the presence of channel errors. We further suggest a technique to minimize the completion time of the offloaded application using a novel scheduling technique called task duplication. We show the effectiveness of our algorithm via trace-driven simulation.

Contents

Acknowledgments

The work in this thesis was made possible by the involvement of a large number of people. I am especially indebted to my advisor Samir R. Das for his guidance through the second half of my PhD. His help and support especially during moments of disappointments were extremely crucial in ensuring that I finish my PhD. I would also like to thank the other professors in the thesis committee – Himanshu Gupta, Pradipta De, Petar Djuric and Jie Gao. Himanshu thoroughly vetted and helped improve most of my work. Petar helped us improve this work by giving a number of useful suggestions. Pradipta advised me during the first part of my PhD. Finally, Jie actively participated in both my proposal and defense committees. I am grateful to each of them for their help.

I also had the opportunity to work on a number of projects involving other professors, namely Aruna Balasubramanian, Anshul Gandhi, Minh Hoai Nguyen, Ansuman Banerjee, Domenico Giustiniano and Jihoon Ryoo. Ansuman helped me with both ideas and analysis of the work related to offloading. I had a very enjoyable experience working with Domenico and and his graduate student Roberto during my internship. The ideas and support given by each of them helped me substantially improve this work. I also had the opportunity to work with and learn from a number of other students in our lab – Ayon Chakraborty, Vasudevan Nagendra, Sohee Kim Park and Caitao Zhan. Among them, I am especially indebted to Ayon for introducing to me the concepts of spectrum sensing and signal processing. I am grateful to have got the chance to work with a number of other students, like Snigdha Kamal, Jitendra Savanur, Hoyoung Kim and Roberto Calvo (at IMDEA Networks). I also received a lot of help from a number of students from other labs, such as Jian Xu, Hirak Sarkar, Shinyoung Cho, Hangil Kang, Duin Baek, Dan Zhang, Santiago Vargas, Shaifur Rahman, Darius Coelho, Ayush Kumar, Jayesh Ranjan and Ray Vo. I am especially grateful to Jian

and Shinyoung for letting me access their servers, which helped me get the experimental results much faster than what I thought was possible.

Finally, I would like to acknowledge the help of my mother who was supportive of my efforts throughout.

Publications

Publications Based on this Thesis

- **Arani Bhattacharya**, Caitao Zhan, Himanshu Gupta, Samir R. Das, Petar M. Djuric. "Selection of Sensors for Efficient Transmitter Localization", In Proceedings of IEEE Infocom held in 2020, Beijing, China
- **Arani Bhattacharya**, Ayon Chakraborty, Samir R. Das, Himanshu Gupta, Petar M. Djuric. "Spectrum Patrolling with Crowdsourced Spectrum Sensors", In IEEE Transactions in Cognitive Communications and Networking *(Extended version of IEEE Infocom 2018)*
- Ayon Chakraborty, **Arani Bhattacharya**, Snigdha Kamal, Samir R. Das, Himanshu Gupta, Petar M. Djuric. "Spectrum Patrolling with Crowdsourced Spectrum Sensors", In Proceedings of IEEE Infocom held in 2018, Honululu, Hawaii, USA
- **Arani Bhattacharya**, Han Chen, Peter Milder, Samir R. Das. "Quantifying Energy and Latency Improvements of FPGA-Based Spectrum Sensors", In Proceedings of IEEE Dynamic Spectrum Access Networks (DySPAN) held in 2018, Seoul, Korea
- **Arani Bhattacharya**, Ansuman Banerjee, Pradipta De. "Parametric Analysis of Mobile Cloud Computing Frameworks using Simulation Modeling", In Proceedings of the Workshop on Adaptive Resource Management Scheduling for Cloud Computing (ARMS-CC) in conjunction with ACM PODC held in 2015, Donostia-San Sebastián, Spain
- **Arani Bhattacharya**, Pradipta De. "Computation Offloading from Mobile Devices: Can Edge Devices Perform Better Than the Cloud?", In Proceedings of the Workshop on Adaptive Resource Management Scheduling for Cloud Computing (ARMS-CC) in conjunction with ACM PODC held in 2016, Chicago, USA

- **Arani Bhattacharya**, Ansuman Banerjee, Pradipta De. "Service Level Guarantee for Mobile Application Offloading in Presence of Wireless Channel Errors", In Proceedings of IEEE Global Tecommunications Conference (Globecom) held in 2016, Washington DC, USA
- **Arani Bhattacharya**, Ansuman Banerjee, Pradipta De. "Scheduling with Task Duplication for Application Offloading", In Proceedings of IEEE Consumer Communication and Networking Conference (CCNC) held in 2017, Las Vegas, Nevada, USA

Other Related Publications

- Mallesham Dasari, **Arani Bhattacharya**, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, Samir R. Das. "Streaming 360 Videos using Super-resolution", In Proceedings of IEEE Infocom held in 2020, Beijing, China
- Sohee Kim Park, **Arani Bhattacharya**, Zhibo Yang, Mallesham Dasari, Samir R. Das, Dimitris Samaras. "Advancing User Quality of Experience in 360-degree Video Streaming", In Proceedings of IFIP Networking held in 2019, Warsaw, Poland
- Vasudevan Nagendra, **Arani Bhattacharya**, Anshul Gandhi, Samir R. Das. "Scalable and Resource Efficient Control Plane for Next Generation Cellular Packet Core", In Proceedings of ACM Symposium on SDN Research (SOSR) held in 2019, San Jose, California, USA
- Mallesham Dasari, Bershgal Muhammad Atique, **Arani Bhattacharya**, Samir R. Das. "Spectrum Protection from Micro-Transmissions using Distributed Spectrum Patrolling", In Proceedings of Passive and Active Measurement (PAM) held in 2019, Puerto Varas, Chile
- Mallesham Dasari, Santiago Vargas, **Arani Bhattacharya**, Aruna Balasubramanian, Samir R. Das, Michael Ferdman. "Impact of Device Parameters on QoE of Internet-based Mobile Applications", In Proceedings of ACM Internet Measurement Conference (IMC) held in 2018, Boston, Massachusetts, USA
- **Arani Bhattacharya**, Pradipta De. "A Survey of Adaptive Techniques in Computation Offloading", In Journal of Network and Computer Applications, published in Volume 78, 2017

Chapter 1

Introduction

The Internet is witnessing an explosion in terms of data consumption and the number of connected devices. The Internet was originally designed to connect autonomous computers. A key feature of the Internet till the late 90's was that communication was slow and expensive. Thus, computers connected to the Internet typically tried to conserve bandwidth by limiting the data sent.

The rapid improvement in the capacity of both wired and wireless networks has changed the structure of the Internet. Devices connected to the Internet are no longer autonomous in nature, but can perform some dedicated task (such as taking videos or pictures) and then transfer the data to a server. Such devices are usually connected over a wireless network. This new Internet structure is commonly referred to as the Internet of Things (IoT). Since the number of such devices can be much larger than autonomous computers, this requires accommodating a large number of devices within a wireless network. Enabling smooth evolution of IoT requires allowing a massive number of wireless devices to connect and transfer large amounts of data over the wireless network at a low cost.

1.1 The Current State of the Internet

To further understand the current state of the Internet, we track the amount of data transferred over wireless networks all across the world from 2005. The amount of data transferred is reported by Cisco, as part of the Cisco Visual Networking Index reports. We summarize the collated data in Figure 1.1. We find that till 2005, the amount of data was less than 1 Petabyte

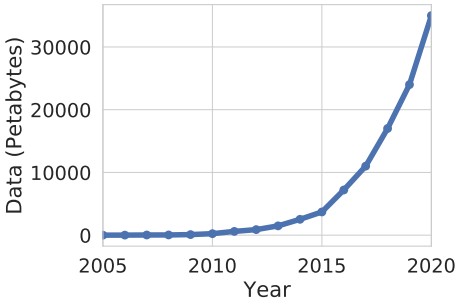


Figure 1.1: Amount of data transmitted over wireless networks each year all over the world.

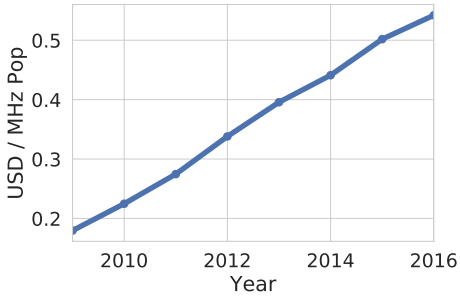


Figure 1.2: Median cost of 700-900 MHz spectrum all over the world from 2009 to 2016.

(PB). By 2015, this had risen over 1000 times to 1 Exabyte (EB), and this is expected to rise to 35 EB by 2020. This is a rise of over 35000 times in the last 15 years. Wireless networks need to evolve to handle such large increases in demand.

1.1.1 Rising Cost of Spectrum

A major consequence of this demand in data is the rise in the cost of spectrum paid by telecom operators to acquire licenses. Since this cost has to be raised by the telecom operators from consumers, this indirectly leads to an increase in cost of communication. Figure 1.2 shows the cost of spectrum from the year 2008 to 2016. Note that the cost of spectrum is measured in $ / MHz Pop, which refers to the cost in USD to get a license for 1 MHz of bandwidth per unit population. We find that the cost of spectrum license has increased from $0.18\$/MHz\ Pop$ in 2009 to $0.54\$/MHz\ Pop$ in 2016. This is a rise of three times in the last eight years. This rise, if not checked, can lead to increase in communication cost for consumers.

1.1.2 Presence of Low-Powered Connected Devices

A second challenge associated with the rising popularity of Internet of Things (IoT) is that many of the devices have low-power processors. They are also constrained by amount of energy available, due to them being dependent on batteries or some form of energy-harvesting. Users demand faster response times for their applications and longer battery lives. Thus, it is essential to

find ways of reducing application execution times and energy consumption to satisfy requirements of users.

1.2 Resource Management in Emerging Wireless Network Applications

The increase in amount of data and connected devices has made it essential to better manage network resources. Network resources such as spectrum and availability of compute power are finite resources and need to be intelligently managed.

1.2.1 Spectrum Patrolling

The increase in cost of spectrum has made it essential for it to be properly monitored and guarded against unauthorized users. Although statistics on unauthorized spectrum use are not publicly available, anecdotal evidence suggests that such unauthorized use is becoming increasingly common [105, 39]. Interference can also occur due to RF leakage from cable plants and connectors [76]. These incidents can all lead to complaints about quality-of-service from customers of telecommunication service providers.

Current strategies to protect spectrum used by the regulatory agencies and telecommunication service providers rely on wardriving using specialized equipments. This suffers from high equipment and human labor cost. To mitigate this problem, a number of recent studies propose deploying cheap (but less accurate) software-defined radios available in the market to monitor spectrum [24]. Such deployment can be readily done by crowdsourcing, where users may be given some incentives (financial or otherwise) to deploy sensors. These studies show that by deploying a large number of such cheap sensors, it is possible to accurate detect or localize the presence of such unauthorized transmissions.

However, running a large number of deployed sensors also has some operational cost. Apart from the cost of incentives, it also requires us to account for the cost of backhaul to send the data and energy to run the sensors. For crowdsourced spectrum monitoring to be used in practice, it is essential to reduce the cost of monitoring spectrum. Such operational cost can be reduced using two major techniques — intelligent selection of sensors

to select the ones that are most relevant, and improving the energy efficiency of spectrum sensors.

Our Contributions: Our first contribution is related to crowdsourced deployment of spectrum sensors to detect illegal transmitters and monitor usage. We propose algorithms to select the most relevant sensors and then show a technique of combining the individual sensors into a global decision. We also show that our global decision is optimal, i.e. given the individual local decisions, our algorithm provides the most accurate possible global decision. We show using both mathematical techniques and experiments through actual deployment of sensors that our system detects and/or localizes transmitters more accurately and has less cost than traditional techniques.

Our second proposed technique of reducing cost is to use FPGA-based spectrum sensors. Since the computation done by spectrum sensors is repetitive, utilizing FPGA's can significantly reduce both energy consumption and latency of computation. We benchmark the performance of our designed FPGA-based spectrum sensors, and show that it has an order of magnitude improvement compared to sensors based on smartphones and Raspberry Pi's.

1.2.2 Computation Offloading

Recently, the number and type of smart devices available in the market has increased manifold. A key effect of this increase is that more complex applications are being run on devices with limited compute capability or batteries with limited energy. One technique that is frequently used to speed up execution of applications or conserve energy is to offload some of their more compute-intensive components for execution on more powerful devices. For example, a smartwatch can offload to the user's own smartphone, whereas a smartphone can utilize to a cloud server.

However, offloading raises a number of major challenges. The first challenge is to decide which components of applications are sufficiently demanding to be offloaded. A second challenge is to deal with the uncertainty that is inherent in the wireless networks used by these devices.

Our Contributions: Our work has three major contributions. Our first contribution is to benchmark the performance of different applications through offloading to both edge and cloud. We study the performance gains obtained for different applications using cloud or edge device. We show that even edge devices such as desktops with relatively less powerful processors can significantly reduce latency.

Our second contribution deals with decide which components of applications to offload. Conventional techniques model the application as DAG and then partition it using either an optimization solver or a heuristic. In contrast, we propose an algorithm to identify the optimal execution position (local or remote server) of each application component, while consuming much lower resources. We show that this can be done in polynomial time, thus significantly reducing the overhead of partitioning the graph.

Our third contribution deals with offloading over a lossy network. A key challenge of offloading is that execution of latency-sensitive applications can miss their deadlines, leading to overall degradation in Quality of Experience (QoE). In our work, we model the number of retransmissions using a Binomial distribution, and then propose a heuristic that provides a soft guarantee of satisfying deadline constraints. We then show using trace-driven simulation that our technique provides mean lower execution time than conventional techniques.

1.3 Organization of this Thesis

We now explain the problem statements of each individual chapter of this thesis and our contributions.

- **Chapter 2**: We look at the problem of crowdsourced spectrum monitoring to detect illegal transmitters. An individual sensor reports its local decision about whether a transmitter is present in its vicinity. Our objective is to choose an optimal subset of sensors and their configurations to maximize the overall detection performance subject to given resource (cost) limitations. We present the challenges of this problem in crowdsourced settings and present a set of methods to address them. The proposed methods use data-driven approaches to model individual sensors and develops mechanisms for sensor selection and fusion while accounting for their correlated nature. We present performance results using examples of commodity-based spectrum sensors and show significant improvements relative to baseline approaches. A part of this work has been published in IEEE Transactions on Cognitive Communications and Networking [8] and in the proceedings of IEEE INFOCOM held in 2018 [22].

- **Chapter 3**: In this chapter, we design greedy approximation algorithms for the optimization problem of selecting a given number of sensors

in order to maximize an appropriately defined objective function of localization accuracy. The obvious greedy algorithm delivers a constant-factor approximation only for the special case of two hypotheses (potential locations). For the general case of multiple hypotheses, we design a greedy algorithm based on an appropriate auxiliary objective function—and show that it delivers a provably approximate solution for the general case. We develop techniques to significantly reduce the time complexity of the designed algorithms, by incorporating certain observations and reasonable assumptions. We evaluate our techniques over multiple simulation platforms, including an indoor as well as an outdoor testbed, and demonstrate the effectiveness of our designed techniques—our techniques easily outperform prior and other approaches by up to 50-60% in large-scale simulations.

- **Chapter 4**: We demonstrate a different technique of sensor selection, where the process of sensor selection and localization are done simultaneously. Such online selection of sensors can significantly reduce both latency and energy consumption than offline selection. We discuss an online selection algorithm, and show that it provides more accuracy than baseline techniques, while running faster than offline techniques.

- **Chapter 5**: In this chapter, we demonstrate that typical crowdsourced implementation using a low-cost software radio connected to a Raspberry Pi or a smartphone as host is not energy-efficient and incurs significant latencies. We propose use of field-programmable gate array (FPGA) to improve both metrics for the signal detection task. Our benchmarking shows significant improvements with FPGA platforms relative to using a Raspberry Pi or smartphone, upto a factor of 73 in terms of latency and a factor of 29 in terms of energy usage. A part of this work has been published in the proceedings of IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN) held in 2018 [9].

- **Chapter 6**: We note that offloading decisions in mobile cloud computing and edge computing are influenced by several parameters, like varying degrees of application parallelism, variable network conditions, trade-off between energy saved and time to completion of an application, and even user-defined objectives. In order to investigate the impact of these variable parameters on offloading decision, we present a detailed model

of the offloading problem incorporating these parameters. Implementations of offloading mechanisms in MCC frameworks often rely on only a few of the parameters to reduce system complexity. Using simulation, we analyze influence of the variable parameters on the offloading decision problem, and highlight the complex interactions among the parameters. A part of this work has been published in the proceedings of Second International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing held in 2015 [5].

- **Chapter 7**: Traditional offloading uses cloud data centers which has a high network latency. To mitigate the problem of network latency, recently offloading to computing resources lying within the user's premises, such as network routers, tablets or laptop has been proposed. In this paper, we determine the devices whose processors have sufficient power to act as servers for computation offloading. We perform trace-driven simulation of SPECjvm2008 benchmarks to study the performance using different hardware. Our simulation shows that offloading to current state-of-the-art processors of user devices can improve performance of mobile applications. We find that offloading to user's own laptop reduces finish time of benchmark applications by 10%, compared to offloading to a commercial cloud server. A part of this work has been published in the proceedings of Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing held in 2016 [10].

- **Chapter 8**: Quality of offloading decisions depend on network conditions and hence many offloading solutions assume that MAC layer retransmissions will tackle transient frame errors. This can lead to suboptimal solutions, as well as, degrade service level guarantee of reducing finish time compared to execution without offloading. In this work, we propose an error-aware solution that uses run-time channel conditions to adapt the offloading decisions. We guarantee that given a failure rate bound (ϵ), offloading decisions will achieve application execution in less time than that of local execution with a probability of (1-ϵ) while operating in networks with unpredictable error characteristics. Simulation results show that at channel error rate of 20%, our heuristic provides 90% guarantee of better performance than on-device computation and reduces the mean finish time by 18% compared to execution without any offloading.

A part of this work has been published in the proceedings of IEEE Global Communications Conference held in 2016 [7].

- **Chapter 9**: Computation offloading frameworks partition an application's execution between a cloud server and a low-powered mobile device to minimize its completion time. An important component of an offloading framework is the partitioning algorithm that decides which tasks to execute on mobile device or cloud server. The partitioning algorithm schedules tasks of a mobile application for execution either on mobile device or cloud server to minimize the application finish time. Most offloading frameworks partition parallel applications devices using an optimization solver which takes a lot of time. We show that by allowing duplicate execution of selected tasks on both the mobile device and the remote cloud server, a polynomial algorithm exists to determine a schedule that minimizes the completion time. We use simulation on both random data and traces to show the savings in both finish time and scheduling time over existing approaches. Our trace-driven simulation on benchmark applications shows that our algorithm reduces the scheduling time by 8 times compared to a standard optimization solver while guaranteeing minimum makespan. A part of this work has been published in the proceedings of IEEE Consumer Communications and Networking Conference held in 2017 [6].

- **Chapter 10**: We conclude in Chapter 10 with a brief discussion of our contributions and scope of our future work.

Part I

Low-Cost Spectrum Monitoring

Chapter 2

Spectrum Patrolling with Crowdsourced Spectrum Sensors

2.1 Introduction

With growing realization of mobile communication's impact on the nation's economic prosperity, RF spectrum has emerged as an important natural resource that is in limited supply [58]. While various spectrum sharing models are being developed to improve spectrum usage, 'spectrum patrolling' to detect unauthorized spectrum use is emerging as a critical technology [36]. Such unauthorized uses can take many forms, such as lower-tier devices accessing spectrum reserved for higher tier devices in a tired spectrum sharing model [43], unauthorized devices accessing licensed spectra using software radios, or various forms of denial of service attacks. Techniques must be developed to detect such unauthorized accesses and large-scale spectrum monitoring is one effective way to do this.

However, large-scale spectrum monitoring using lab-grade spectrum analyzers is not scalable, given that such devices cost anywhere from several thousands to tens of thousands of US$ depending on the exact capability and require availability of AC power. Several recent chapters have proposed to address this scalability issue by deploying low-cost, small form-factor, low-power spectrum sensors in large numbers perhaps using a crowdsourcing paradigm [24, 124, 16].[1] The overall monitoring performance achieved by a large number

[1] There is at least one commercially successful crowdsourced application of spectrum sensing. FlightAware [40] deploys low-cost sensors via crowdsourcing to detect signals

of such low-cost sensors can exceed that of a handful of lab-grade spectrum analyzers while costing several orders of magnitude less [24]. Due to this reason there is a growing body of literature in studying the performance characteristics of commodity-based inexpensive sensors [23, 87, 16].

Although using inexpensive, commodity-grade sensors in large numbers may provide a very encouraging cost-performance tradeoff, use of a crowdsourcing paradigm brings in certain management problems. Spectrum patrolling must involve signal detection. It is unlikely that all deployed sensors will be used in specific detection tasks [24]. Only a subset will be typically be employed ensuring that the required level of detection performance is achieved. This conserves the backhaul bandwidth and also energy when the sensors are battery operated (e.g., when mobile phones serve as spectrum sensors [23]). In case of multiple sensing needs in the same geographical space (e.g., detecting specific signals in multiple spectrum bands), sensors may need to be configured to engage in one specific task as their processing powers may not be sufficient for multiple concurrent signal detection tasks. The broad goal of this work is to *develop mechanisms to select the right set of sensors that optimizes the performance of detection task for a given cost.* There are two sub-problems that arise: 1) modeling individual sensor performance and cost for given configurations, 2) fusing data from multiple sensors and selecting the optimal subset to maximize detection performance subject to cost limitations (or, minimizing cost subject to a given detection performance). While these problems are not entirely new in a general sense, the specific nature of crowdsourced spectrum patrolling problem makes them challenging.

Challenge 1 – Modeling Individual Sensors: Fundamentally spectrum sensors must perform a signal detection task in form of a binary hypothesis testing (intruding transmitter present/absent). Detection performance is usually characterized by standard metrics like the *probability of detection* (P_d) or *false alarm rate* (P_{fa}). Assigning a specific sensor to a specific sensing task and choosing specific configurations, requires accurate estimation of its P_d and P_{fa} metrics and cost for such configurations. Modeling of the cost depends on the scenario and can include, e.g., energy cost, backhaul data cost or any form incentives to be paid to the owner of the sensor. However, given the heterogeneity and diversity of spectrum sensors in a crowdsensing paradigm estimating such metrics accurately is challenging. Existing literature extensively uses so-called first principles modeling approach

from aircrafts flying overhead.

that could miss various forms of imperfections (e.g., clock skew, I/Q imbalance, RF front end non-linearity) and noises common in commodity platforms. Even when they are able to account for those, they require knowledge of internal details of the sensor or separate calibration efforts. These are either not practical or do not scale well. More specifics of these issues will be discussed in Section 2.2.

Instead of relying on first principles models, we use a data-driven (blackbox) approach where models are created based on data from prolonged observation of the sensor. This type of approach is getting traction in other communities such as industrial process control where first-principles approaches are not practical for largely similar reasons (see, e.g., [119]). We abstract out the observable and easily quantifiable parameters of a sensor, its operating environment or runtime configuration. We use machine learning methods that treats the internal sensor hardware information (otherwise inaccessible) as hidden variables. This gives our methodology a direct and practical advantage over involved analytical models. Second, such models get richer with time and can easily accommodate new sensors without the need of explicitly calibrating them, an otherwise impossible task.

Challenge 2 – Sensor Selection and Fusion: Once individual sensors are modeled, we must select the subset of sensors (and their configurations if they are configurable) to achieve the best cost-performance tradeoff, i.e., the best detection performance for a given total cost (or minimum cost for a given desired performance). Here, the local sensor decisions (target present/absent) are to be combined into a global 'fused' decision. Thus, a fusion rule is needed. While there is a very rich literature on sensor fusion and developing optimal fusion rules much of techniques in literature assume that *sensor decisions are conditionally independent.* This is not true for spectrum sensors, where their decisions could be correlated depending on the sensor locations. The reason is that sensors located in the same neighborhood are likely to face the same fading environment, resulting in correlations in their observations/decisions. The case for correlated observations have been indeed studied (see, e.g., [64, 34, 113]). But these methods are either too complex computationally to implement in practical systems and/or requires prior knowledge of the correlation structure (e.g., in terms of higher-order moments of the sensor observations under each hypothesis [64] or spatial correlation coefficient [15], etc). Also, these techniques do not help addressing the sensor selection problem.

To handle this problem, we use a variant of sensor selection from machine

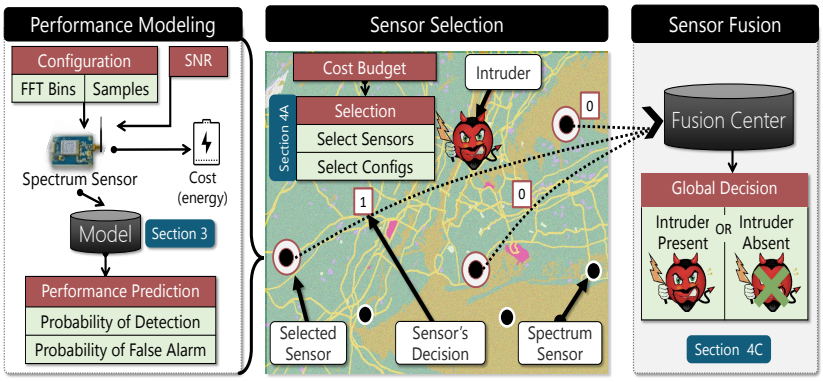


Figure 2.1: Overview of the proposed technique. Performance models of individual spectrum sensors are created first using a data-driven approach. Then sensor selection using a feature selection based approach. Finally individual sensor decisions are fused together to get a global decision. The figure indicates the different steps along with the section numbers where they are described.

learning literature called Maximum Relevance Minimum Redundancy (mRMR) [92]. This technique first measures the value of each sensor by considering both its probability of detection, and its correlation with the other sensors. It uses an adaptive greedy selection where the value of each sensor is computed at each step, and then the sensor with the highest value is taken. While this does not guarantee an optimal subset, experiments on a large variety of datasets have shown that it works well in practice. Our evaluation shows that it works significantly better than a baseline technique that does not take correlation into account.

Contributions Figure 2.1 pictorially describes the overall approach with pointers to various sections of the chapter. Overall, we make two sets of contributions. *First,* we develop a systematic approach for data-driven models of spectrum sensors engaged in signal detection (Section III). The model takes the sensor's configuration and SNR as input and estimates detection performance and cost (we use energy to model cost in this work). We precede this modeling approach by highlighting limitations of traditional first-principles based analytical modeling approaches (Section II) and demonstrate improved model performance using the proposed data-driven approach using actual spectrum sensor hardware. *Second,* we develop a technique for the

sensor selection and fusion problem taking into account the fact spectrum sensors are not conditionally independent (Section IV). The proposed feature selection based technique is suitable for crowdsourcing as it does not require information that is hard to obtain or estimate. We show that the overall detection performance improves significantly relative to baseline techniques.

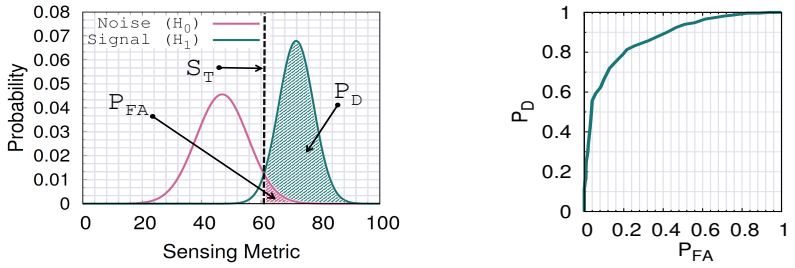
2.2 Modeling Detection Performance[2]

The spectrum sensor detects the absence or presence of an intruding transmitter's signal. The corresponding hypotheses are denoted as H_0 (absence) and H_1 (presence) respectively. Raw sensed samples from the sensor are fed to the corresponding detection algorithm on board of the sensor that computes a *sensing metric*. The sensing metric is compared against a threshold (S_T) to output a binary decision. This is the local decision of the sensor.

Performance Metrics Given H_1, the rate at which the sensor detects the transmitter is known as the *probability of detection* (P_d). Second, given H_0, the rate at which the sensor incorrectly flags the presence of a transmitter is known as the *probability of false alarm* (P_{fa}). Figure 2.2 demonstrates the basic working principle. The sensing metric has two different distributions under hypotheses H_0 and H_1. Under H_0, the distribution reflects noise. P_d and P_{fa} depends on the selection of S_T. Varying S_T varies both P_d and P_{fa} between 0 and 1. This produces the *receiver operating characteristics* (ROC) curve. Specifying P_{fa} (common case) also determines P_d as per the ROC curve. However, the ROC curve itself would look different if the distributions of the sensing metric shown in Figure 2.2(a) change. This is possible when the signal power from the transmitter changes (due to a different location, e.g.). More on this below.

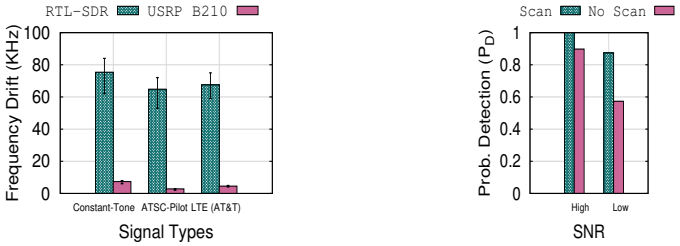
Challenges Estimating an optimal value of S_T is straightforward when the distributions of the sensing metric for H_0 or H_1 (Figure 2.2(a)) are known or can be accurately estimated. Unfortunately, this is not the case in practice. The distributions depend on a variety of factors including the detection algorithm, specifics of the sensor hardware, SNR or SINR at the sensor location, number of sensed samples, FFT resolution and so on. Common detection algorithms are energy-based, waveform or feature-based, autocorrelation

[2]Sections 2.2 and 2.3 are based on work done by Ayon Chakraborty. It has been added to make this chapter complete.



(a) Detector Basics (b) ROC Curve

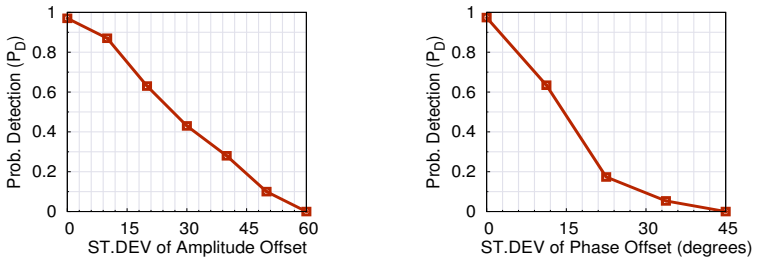
Figure 2.2: Working principle of a detector. S_T denotes the *threshold* of the sensing metric. Increasing S_T increases P_d but also increases P_{fa} as per the ROC curve.



(a) Higher clock-skew in cheaper devices (b) Clock skew deteriorates P_d

Figure 2.3: Unpredictable clock skew makes frequency offset calculation harder resulting in poorer signal detection performance.

or cyclostationary-based. Existing analytical techniques [114, 33, 26] can help model such algorithms to estimate an optimal S_T. However, such models typically result in significant estimation errors [114, 33]. The reasons are as follows. First, many of these models make idealistic assumptions about the distribution of the signal or noise or the noise associated with sensor hardware. For example, [13] shows that the performance of a sensor actually depends on both the signal parameters and the amount of RF front-end non-linearities of the sensors. Second, complex models do exist that take into account such factors [13, 45], but it is seldom possible to parameterize them correctly. This is due to the uncertainty in the hardware itself or inaccessible components that makes reliable measurements impossible. Third, even when such measurements are possible manual calibration of individual sensors does



(a) Effect of Amplitude Offset (b) Effect of Phase Offset

Figure 2.4: We demonstrate the effect of I/Q imbalance in deteriorating the performance of simple waveform based detector algorithm used in detecting an ATSC pilot tone.

not scale well, especially in the context of crowdsourcing.

We provide two sets of benchmarking experiments to highlight the challenges. ***Clock-skew*:** As an example, we study the clock skew associated with the local oscillator (LO) in the sensor. The frequency set in LO tunes the sensor to the desired frequency. However, the LO-frequency drifts giving rise to clock skew. To understand the nature of such drifts in commodity sensor hardware, we use two different spectrum sensors based on RTL-SDR and USRPB210. These sensors are chosen due to their low-power, small form factor nature [23]. They are both USB-powered and could be driven by an embedded CPU board or even a smartphone. Three test signals are used for detection. The first two are constant frequency tones in the 915 MHz band and the pilot tone of an ATSC signal (DTV band). In both cases we observe a non-trivial frequency drift that varies widely across individual sensor instances. For the third, we use an LTE downlink signal from a real network (AT&T) using these sensors and recorded the frequency correction needed in order to decode the synchronization signals. The results are summarized in Figure 2.3(a). In most cases RTL-SDR suffers from a appreciable clock skew which is less prevalent in more expensive hardware like USRP. In Figure 2.3(b) we show the impact of such clock-skew in detecting an ATSC signal. The ATSC signal has a pilot tone located at an offset of 310 KHz that is expected by our waveform based detector algorithm. We create two variations of the algorithm that expects the pilot tone (i) exactly at the 310 KHz offset and (ii) ≈100 KHz surrounding the expected location that it scans. In a low SNR scenario, scanning provides almost a 50% improvement in P_d compared to the detector that expects the pilot at

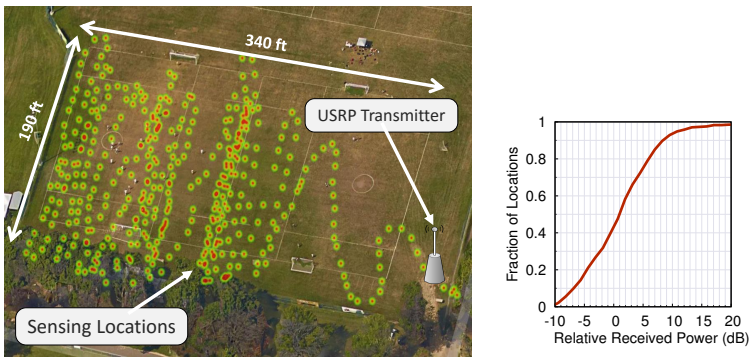
a fixed offset demonstrating the impact of the clock skew problem.
***I/Q imbalance*:** Apart from clock skew, I/Q imbalance and RF front-end non linearities are other prominent issues. I/Q imbalance is introduced as a result of mismatch between the in-phase (I) and quadrature (Q) signal paths of the RF receive chain. For example, phase difference between the I and Q components is not always exactly 90^0 which results in an amplitude and phase offset in an I/Q sample. Since we do not have direct control over the radio circuitry we simulate I/Q imbalance by adding amplitude and phase offsets to real I/Q traces obtained for an ATSC signal using a RTL-SDR device. For both cases, we use an offset drawn from a zero-mean Gaussian with a standard deviation as shown in Figure 2.4. We report the detection rate of the ATSC signal using a waveform based detector that identifies the ATSC pilot signal. As the I/Q imbalance becomes more prominent it becomes impossible to detect the signal. Although I/Q imbalance can be addressed directly in the hardware [45] we expect crowdsourced spectrum sensors may use inexpensive hardware unable to do such corrections.

As mentioned earlier, while such problems can be accounted for by applying models that 'corrects' for such errors, these models are based on the 'first principles' approach. These models can only be applied only after knowing specific sensor-specific parameters (e.g., characteristics of frequency drift, whether the algorithm scans, or nature of I/Q imbalance, etc). This information may not be available in a crowdsourcing scenario given significant possible heterogeneity.

2.3 Data-Driven Performance Modeling

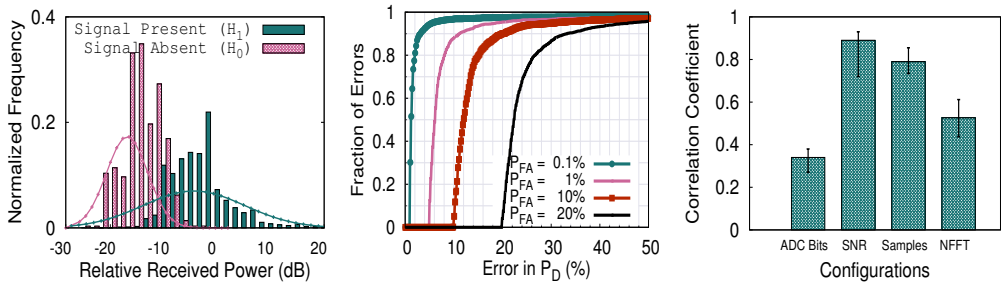
To address this problem of scalable modeling of heterogeneous sensors, we borrow from the concept of data-driven soft sensors utilized in industrial processes [119, 104]. Industrial processes find it impossible to use first principles models for their physical and chemical processes. These models are often idealized (e.g., assumes steady state behavior) or requires parameters that are hard to obtain. Instead, data-driven soft sensors models are gaining ground that takes an alternative blackbox approach where massive amount of collected data is used to model and predict the industrial process behavior in realistic conditions using statistical or machine learning techniques (see, e.g., [119, 104]).

In the following we present our approach for the data-driven analysis



(a) ≈1000 sensing locations (b) Received power CDF

Figure 2.5: Spectrum sensor data collection



(a) Empirical vs. Analytical Model for a fixed configuration (b) Error in estimating P_D (c) Factor analysis for the estimation error

Figure 2.6: Analysis of estimation errors associated with analytical models and its dependency on the sensor's operating environment or configurations. Median estimation error in P_D can be as high as 25%. Higher errors are highly associated to low SNR operating environments.

using an example dataset. We first present our dataset, quantify the errors associated with first-principles based analytical models and then present our data-driven performance model of spectrum sensors.

2.3.1 Dataset

We collect spectrum sensor measurements in an outdoor setting within the university campus. As shown in figure 2.5(a), we setup a USRP B210 based transmitter that transmits a constant tone in the 915 MHz band and collect sensing data (I/Q samples) using three RTL-SDR and two USRP B210 devices. We collect 1M samples at every location and our sensing area covers approximately 1000 locations within a 190×340 ft^2 region (Figure 2.5(a)).

The distribution (H_1) of the received power is also shown in Figure 2.5(b). We bias our data collection towards relatively lower SNR zones so as to have more variations in detection performance. This also presents a more challenging test case – detection is much easier when SNR is high. Using the same set of sensors we also collect a *noise dataset* by turning off the transmitter. This data corresponds to the distribution for H_0. Note that H_0 is agnostic to the sensor's location.

For every location we employ three different detection algorithms (energy, feature and autocorrelation based) [23] both on the signal and the noise dataset. We vary two key parameters of the algorithm that directly influences P_d–P_{fa} as well as energy cost in the sensor [23]: (i) N, number of sensed samples and (ii) $NFFT$, resolution of the FFT. N and $NFFT$ are varied from 32 (2^5) to 4096 (2^{12}) by repeated doubling with the constraint of $N \geq NFFT$ (36 configurations). We introduce heterogeneity in the resolution of sensed samples by changing the number of bits per sample. We produce additional data sets of 14, 12, 10 and 6 bit samples by ignoring least significant bits from the collected 16 bit samples. Note that this depends on the resolution of the ADC in the sensor and heavily influences the dollar cost.

2.3.2 Limitations of Analytical Models

Before directly delving into the internals of the data driven model, we first demonstrate the limitations of first-principles based analytical models using our dataset. Due to space restriction we are not able to explain individual variations of analytical models we use but will explain the general conclusions and trends. Figure 2.6(a) shows two histograms of the sensing metric corresponding to H_0 and H_1 obtained by using the energy-based detector algorithm (N = 2048, NFFT = 1024). We use the analytical model for energy-based detector to estimate the distributions for H_0 and H_1 for the same location. Figure 2.6(a) visually shows the difference between *ground truth* and *estimated* distributions. In Figure 2.6(b) we present the estimation errors for different values of P_{FA}. Note that the median error can be as high as 25% that in many cases. We observe that the errors are particularly higher in low SNR scenarios. We also show (Figure 2.6(c)) the correlation of such errors to the sensing configurations. Unlike other factors, the number of ADC-bits does not show a very high degree of correlation. This may be because we attempt to detect a simple tone at a constant power in this study.

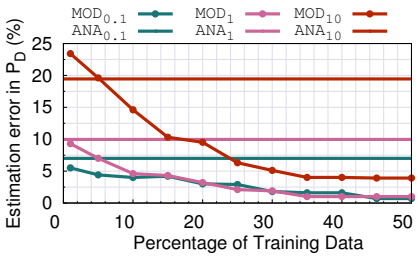


Figure 2.7: Performance of the data-driven models $MOD_{P_{FA}}$ compared to the analytical models $ANA_{P_{FA}}$ for different values of P_{fa}.

2.3.3 Data-Driven Performance Model

Given the relatively poor performance of parametric models, we make use of 'training data' collected from spectrum sensors to take a non-parametric data-driven approach. Essentially, the task of the model is to determine an optimal sensing threshold, S_T^{opt} that maximizes P_d for a given P_{fa}. For training the model we use feature vectors of the form V: <`Algorithm`, `N`, `NFFT`, `B`, `SNR`, $\texttt{P}_{FA}^{target}$>. P_{FA}^{target} is the allowable false alarm rate. `Algorithm` refers to the signal detection algorithm the sensor runs that uses `N`, `B`-bit samples and involves an `NFFT`-bin FFT. We use energy, waveform and autocorrelation based detection algorithms. `SNR` refers to the signal-to-noise ratio of the intended signal at the sensor's location. Every V_i is mapped to a corresponding $S_T^{opt_i}$ in the training examples. Note that we do not explicitly take into account internal hardware details unlike the involved analytical models [12, 44]. We explore off-the-shelf machine learning techniques to learn the estimator for S_T^{opt}. Out of several popular techniques we tried out, the Support Vector Regressors (SVR) works best in our case. We have also explored deep-learning methodologies [119] using convolutional neural networks (CNN), however the amount of training data required to get reasonable estimation performance is significant. This makes CNN impractical in our case and we adopt SVR for creating the performance model.

Validation: We validate the performance of our data-driven model in Figure 2.7. Given configuration of the sensor and the SNR it operates in, our model predicts the optimal threshold S_T^{opt} that maximizes P_d for a fixed P_{FA}. We use the sensor traces and the model predicted $\widehat{S_T^{opt}}$ to compute $\widehat{P_D}$ for a given P_{FA}. The relative error of $\widehat{P_D}$ with respect to P_d is reported. We show estimation error in P_d for P_{fa} equal to 0.1%, 1% and 10% respectively. The

data-driven models are indicated by $MOD_{P_{FA}}$ in Figure 2.7. We also present the estimation errors of the analytical models ($ANA_{P_{FA}}$) for the same set of data points (low/moderate SNRs). In all cases after our model is moderately trained we reduce our estimation error by a significant margin with respect to the analytical models. For instance, MOD_{10} outperforms ANA_{10} by $\approx 12\%$ for a training set of size 20%. With more training samples the estimation error of our model becomes negligible and we see a clear improvement over analytical performance models.

2.4 Sensor Selection and Fusion

The approach described in the previous section gives us the power to estimate the detection performance of an individual sensor deployed in the wild without explicitly calibrating it. In this section we use such models to optimize the (network-wide or global) detection rate. This is done by selecting an optimal set of sensors (and their configurations such as number of ADC bits, number of samples or FFT bins etc.) and fusing their local decisions into a *network-wide (global)* decision. This needs a simultaneous solution of sensor selection and sensor fusion problems. As discussed in Section 2.1, a wide body of literature exists that propose mathematical techniques to fuse sensor decisions to optimize certain detection performance metrics (typically Bayes risk). In a widely used method proposed by Chair and Varshney [20] that we will also use, an optimal fusion rule is developed to minimize the sum of false alarm and missed detection rates, but specifically for case when the sensors are conditionally independent.

As explained in Section 2.1, the conditional independence assumption does not hold for spectrum sensors and existing techniques to account for correlated sensor observations are hard to apply for case of crowdsourced spectrum sensors either due to complexity or unavailable parameters. We develop an alternative feature selection based approach below that we will demonstrate to perform well in practice.

2.4.1 Sensor Selection

To optimize performance under the constraint of a cost budget, we need to select a set of sensors $\mathbf{S}$ that collectively offers the best network-wide detection performance. Let $P_D(\mathbf{S})$ denote the probability that the set of

sensors $\mathbf{S}$ detect an intruder. We denote the selection of a sensor by setting the decision variable $z_i = 1$, otherwise we set $z_i = 0$. Let C_i denote the cost of utilizing sensor S_i. Our objective is to maximize the probability of detection while keeping the cost within a fixed budget B:

$$\text{Maximize } P_D(\mathbf{S}) \text{ subject to: } \sum_{S_i \in \mathbf{S}} z_i C_i \leq B. \tag{2.1}$$

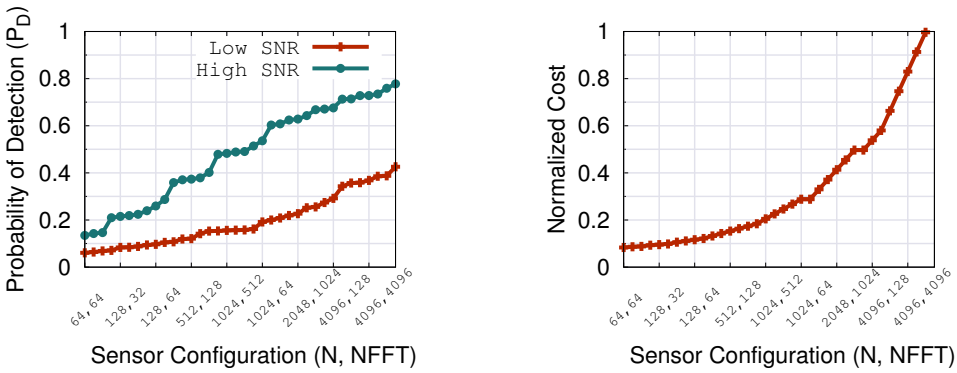
Sensor Ranking: Solving this optimization is a known NP-Hard problem, since the sensors are correlated. This is mainly because quantifying the effect of the correlations on performance of the set of sensors is difficult. To solve this optimization problem, we utilize a variant of a commonly used feature selection technique from the machine learning literature, known as Maximum Relevance Minimum Redundancy (mRMR) [92]. In this technique, the sensors are ranked based on their contribution to $P_D(\mathbf{S})$. However, a measurement of the contribution of a single sensor needs to take into account two distinct factors:

- **Relevance:** A sensor is more relevant if its data is more frequently used to detect an intruder. Based on the feature selection literature, we measure the relevance of a sensor by looking at the mutual information metric of the sensor readings and the presence of intruder. Let X_i be a random variable denoting the local decision given by sensor S_i. Also, let U be a random variable denoting if an intruder is actually present. Both X_i and U are binary random variables. Then, the relevance of S_i is measured by the mutual information between X_i and U, $I(X_i, U)$:

$$I(X_i, U) = \sum_{x_i \in \{0,1\}} \sum_{u \in \{0,1\}} P(X_i = x_i, U = u) log \frac{P(x_i, u)}{P(X_i = x_i)P(U = u)} \tag{2.2}$$

  The value of $I(X_i, U)$ is 1 if X_i and U are perfectly correlated, and 0 if they are completely independent. Thus, $I(X_i, U)$ is a measure of how relevant the sensor S_i is in detecting the presence of the intruder (denoted by U).

- **Redundancy:** Assuming we already have selected a set of sensors, we need a way to measure if adding a new sensor adds any new information. A common approach of measuring the redundancy of a sensor S_k with respect to a subset $\mathbf{T} \subseteq \mathbf{S}$ is to measure the amount of information



(a) Configuration vs. P_d for $P_{fa} = 0.1\%$ (b) Configuration vs. Normalized Cost

Figure 2.8: (a) P_d for different configurations of the sensor under low and high SNR. (b) Sensor cost model. N is number of samples, NFFT is number of FFT bins.

given by the new sensor about the output of the subset:

$$R(S_k, \mathbf{T}) = \frac{1}{|\mathbf{T}|} \sum_{S_i \in \mathbf{T}} I(X_i, X_k) \tag{2.3}$$

If two sensors S_i and S_k are spatially close to each other, then the mutual information among these two sensors will be high. In this case, selecting both the sensors leads to high redundancy of one sensor with a subset containing the other. Thus, the value of adding a sensor to a subset reduces if they have a high redundancy.

To account for both relevance and redundancy, the actual value (denoted by $V(S_k, \mathbf{T})$) of adding a sensor S_k is the difference between the mutual information and redundancy. Mathematically, we write this as:

$$V(S_k, \mathbf{T}) = I(X_k, U) - R(S_k, \mathbf{T}). \tag{2.4}$$

Sensor selection schemes: For each pair of sensor $S_i \in \mathbf{S}$ and subset of the sensor set $\mathbf{T} \subseteq \mathbf{S}$, we now have a fixed value $V(S_i, \mathbf{T})$. We also have a fixed cost C_i for each sensor $S_i \in \mathbf{S}$. This is a feature selection problem with linear cost constraints, which is in general NP-hard. We first look at solving it in the simple case where sensors are homogeneous in terms of configurations, and followed by heterogeneous configurations.

Homogeneous Sensors (HOMS): We assume all sensors are identical and have the same configuration. Hence their costs are equal and we assume unit cost for every sensor, i.e., $C_i = 1$. In this case we first iterate across all the sensors and select the sensor with the highest V_i. We add this sensor to the subset $\mathbf{T}$. In the next iteration, we recompute the values of V_i for all the remaining sensors, and again select the maximum. In this way, we keep selecting sensors until we reach the budget for the number of sensors allowed.

Heterogeneous Sensors (HETS): In this case the sensors have heterogeneous configurations that are preconfigured for every sensor and cannot be changed. Accordingly, the sensor's cost C_i is a function of its configuration as demonstrated in Figure 2.8(b). Depending on the sensor's configuration, C_i can vary anywhere from the minimum cost value to 1. In this case, we pick the sensor having the highest value-to-cost ratio V_i/C_i, and add it to the subset $\mathbf{T}$. We recompute the values of V_i/C_i again, and keep picking the sensor with the highest value and adding it to $\mathbf{T}$ until again we exceed the budget. We summarize this algorithm in Algorithm 1.

Algorithm 1 HETS: Heterogeneous Sensor Selection.

Input: Value of sensors V, cost of sensors C, cost budget B
Output: Optimal selection A

$R = 0$ /* R stores the cost of sensors selected so far. */
$\mathbf{T} \leftarrow \phi$
while $R < B$ **do**
    $j \leftarrow \arg\max_{i=1}^{N} V_i/C_i$
    $\mathbf{T} \leftarrow \mathbf{T} \cup \{S_j\}$
    $R \leftarrow R + C_j$
    $V_j \leftarrow 0$
return T

Reconfigurable Sensors (RES): Here, the sensor can adopt a specific configuration from a pool of available configurations. Here the task is not only to select the sensors but also determine the configuration of the sensor that it should adopt. We again compute the value-to-cost ratios V_i/C_i for each configuration, and select the one that provides the highest. However, in the next iteration, we repeat the procedure after excluding the sensor that has already been selected in the previous step. We repeat this procedure until no sensor can be selected within the budgeted cost.

Time Complexity: To understand the time complexity of our technique, we note that selecting a single sensor requires iterating over all the sensors to compute each sensor's relevance. This requires $O(|\mathbf{S}|)$ time. It also requires iterating over all the selected sensors. Since the number of selected sensors is always less than the budget B, this requires B time. Thus, a single selection requires $O(|\mathbf{S}| \times B)$ time. This needs to run B times to fill the budget, and so the total time complexity of our technique is $O(|\mathbf{S}| \times B^2)$.

2.4.2 Sensor Fusion

We now have a selection of sensors and their configurations. We use the Chair-Varshney optimal sensor fusion rule [20] that fuses the local decisions of the individual sensors into a global (fused) decision to minimize the error rate. However, Chair-Varshney sensor fusion rule assumes that the sensor decisions are conditionally independent. This is not true in practice in our case, since the intruder can arrive at any location within the area, which affect the sensor local decisions.

To resolve this limitation, we apply this fusion rule repeatedly for each possible location of the intruder. We note that for a particular location of the intruder, the sensor local decisions are conditionally independent. Formally, assume that $U_{i,L=j}$ is the local decision (1 or 0) of the sensor S_i, if the intruder signal is detected or not detected (respectively) by this sensor given the intruder is at location j. Using [20], we compute the fused decision $U_{L=j}$ of the sensors given this location of the intruder as:

$$U_{L=j} = \sum_{P_{Di,L=j} > P_{FAi}} [U_i \log \frac{P_{Di,L=j}}{P_{FAi}} + (1 - U_i) \log \frac{1 - P_{Di,L=j}}{1 - P_{FAi}}] \qquad (2.5)$$

The summation above is for all selected sensors. $P_{Di,L=j}$ is the probability of detection of sensor S_i for intruder location j. $U_{L=j} > 0$ indicates presence of the intruder (at location j), otherwise it is considered absent. Note that we only consider sensors with probability of detection higher than the probability of false alarm from a particular location L, since only those sensors are close enough to give meaningful information. To estimate the presence of an intruder anywhere, we first compute the values of $U_{L=j}$ for all possible locations j. We conclude that there is an intruder anywhere only if at least one of these $U_{L=j}$'s is positive. Otherwise, we conclude that no intruder is present.

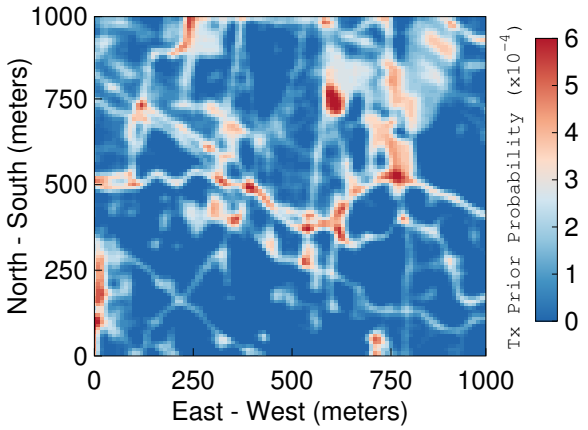
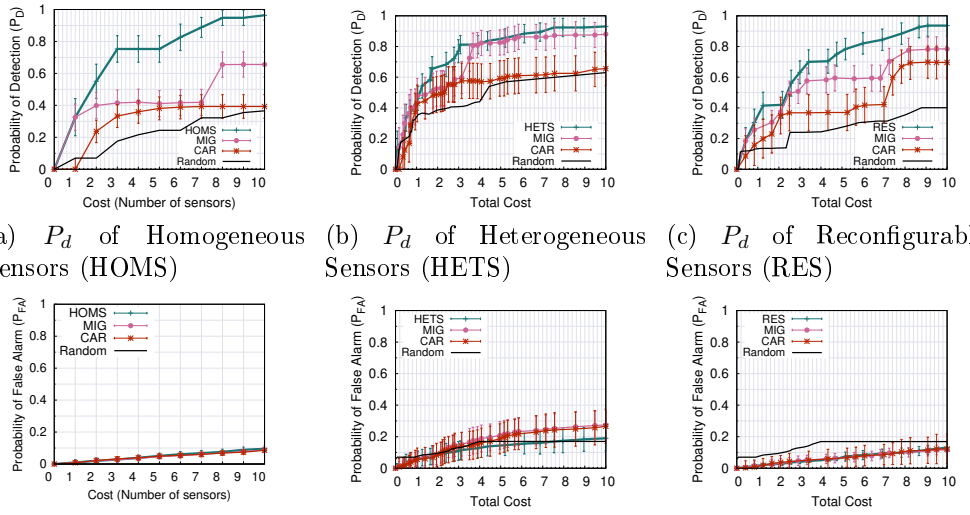


Figure 2.9: Transmitter prior map of the area obtained from a satellite image.

2.5 Evaluation

We simulate a 1000 m×1000 m grid where we randomly deploy 100 spectrum sensors. The sensors can choose among 36 different configurations. Each configuration corresponds to the tuple (N, NFFT), N being the number of I/Q samples and NFFT, the resolution of the FFT in the sensor's detection algorithm. N, NFFT $\in \{2^5, 2^6, \cdots, 2^{12}\}$ such that N $\geq$ NFFT. For each sensor, we set $P_{fa} = 1\%$ (or 0.01) and obtain the P_d from our data-driven performance model (MOD_{10}). The sensors have a cost model as mentioned in Figure 2.8. Next, we simulate an intruder in the grid. The intruder is represented by a wireless transmitter with a transmit power of 10 dB. We use the log-normal model to compute RSS at all the sensor locations. We make the intruder's *prior map* realistic to account for different factors such as terrain information or proximity to residential or navigable areas. We create the *prior map* directly from a snapshot of Google map's satellite imagery data. To remove intricate details (e.g., buildings, texture) in the image, we apply Gaussian blur, a well known image filtering technique. Next we resize the image to a dimension of 100×100 to emulate our grid. We make the prior probability of the transmitter to be present in a certain cell $< i, j >$ proportional to the pixel intensity at $< i, j >$. Figure 2.9 shows our prior map. For all simulations we sample the intruder's location 10 K times from the *prior map* that we use to obtain weights for our sensor selection algorithms. Every time the intruder appears the selected sensors attempt to determine its presence with their respective values of P_D. The fused decision



(a) P_d of Homogeneous Sensors (HOMS) (b) P_d of Heterogeneous Sensors (HETS) (c) P_d of Reconfigurable Sensors (RES)

(d) P_{fa} of Homogeneous Sensors (HOMS) (e) P_{fa} of Heterogeneous Sensors (HETS) (f) P_{fa} of Reconfigurable Sensors (RES)

Figure 2.10: Comparison of performance for the three proposed schemes with *MIG*, *CAR* and *Random* baseline heuristics. For each data point, we show the mean value and the standard deviation. We do not show the standard deviation of *Random* scheme for clarity since it has a higher standard deviation.

is compared to the ground truth. We compute the detection rate for the given instance of selected sensor by simulating the intruder 1000 times. We also compute the false alarm rate by simulating another 1000 cases where no intruder is present.

2.5.1 Performance of Sensor Selection Algorithm

We compare the performance of our sensor selection algorithms with two baseline algorithms. As baseline, we first run a *random* selection algorithm where we pick the sensors randomly with uniform probability. We then run a greedy algorithm where we pick the best sensors (the ones with highest relevance) without accounting for their correlation. We refer this algorithm as **m**utual **i**nformation based **G**reedy (MIG). When the sensors are homogeneous, *MIG* selects the sensors for which the prior probabilities are the highest. For other cases, *MIG* selects sensors in decreasing order of their V_i/C_i ratios.

Finally, we also run the sensor selection algorithm proposed in [22], which first segments the entire grid into clusters, and then uses ranking of sensors across each cluster. We refer this technique as **C**lustering **a**nd **R**anking (CAR).

Observation Figure 2.10 shows the performance in terms of P_d and P_{fa} obtained by the sensors selected by our algorithms compared to baseline heuristics across different cost budgets. We show both the mean performance and the standard deviation at each of the data points. For HOMS, we consider the number of sensors as the cost, i.e., $C_i = 1$. However for HETS and RES, the cost $C_i \in [min_{cost}, 1]$. We note that our algorithms perform significantly better compared to *MIG*, *CAR* as well as *Random* schemes, especially at medium values of the budget. For all cases, till a budget of 1, our algorithms perform similar to the *MIG* scheme. This is because both of them select sensors only from the cluster with high prior probability. When we increase the budget above 2, the *MIG* method keeps selecting from the same cluster, since it does not consider the effect of correlation. For instance, at a budget of 5, 3 and 4, HOMS, HETS and RES outperform *MIG* scheme by 91%, 10% and 15% respectively. Note that our algorithm also performs much better than random selection in each of the cases. The lower gain in the case of HETS can be explained by observing that a larger number of lower cost sensors provides higher probability of detection than a fewer number of expensive sensors. For less expensive sensor configurations, the amount of correlation is also lower, since their individual P_D's fall more sharply with a reduction in power. Thus, our algorithms, because of its technique of removing redundancy, improves performance the most when the budget constraint requires intelligent selection of sensors.

We also note that the increase in the values of P_d also leads to increase in P_{fa}. However, this increase in the value of P_{fa} is relatively small, as it is always less than 0.1 in case of HOMS and RES, and less than 0.2 in case of HETS. Our algorithm also provides either lower or equal values of P_{fa} compared to each of the baseline techniques.

2.5.2 Performance of Our Fusion Rule

We compare the performance of the Chair Varshney fusion rule with a baseline technique. To compare, we run the same simulation and selection process as HETS, but run both Chair Varshney fusion rule and a baseline technique.

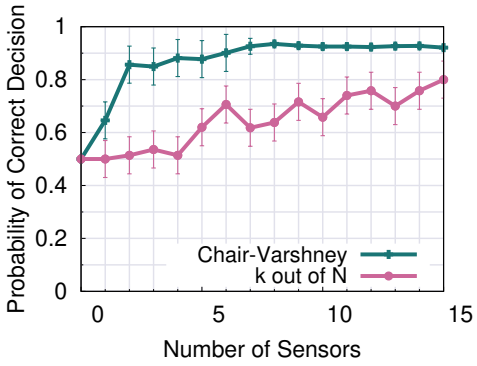


Figure 2.11: Performance of Chair-Varshney Fusion rule (our method) compared to a baseline k out of N rule.

Our baseline technique concludes that there is an intruder if a total of k out of $|\mathbf{S}|$ sensors give output 1, where the best value of k is chosen by simulation.

Observation Figure 2.11 shows the probability of detection using Chair Varshney and the baseline technique. We find that Chair Varshney performs better in all the cases, with the performance rising with increase in number of sensors. Thus, Chair Varshney technique is 91.2% accurate when just 8 sensors are present, whereas using k sensors just gives 65.8% accuracy. This is because Chair Varshney is able to consider the individual performance of each of the sensors, whereas the baseline technique always considers all sensors as equivalent. The contributions of the individual sensors need to be considered for good detection performance. This further confirms our claim that the Chair-Varshney rule is optimal. This also shows that having information about probability of detection of individual sensors is important for accurate sensor fusion. Thus, our data-driven technique of evaluating sensors behavior is necessary to improve the accuracy of detection.

2.6 Related Work

Shared spectrum architectures need to enforce suitable policies to control spectrum access among secondaries [90, 61]. Second, with the advent of cheaper radio hardware the licensed spectrum is prone to unauthorized use [39]. This makes the problem of spectrum patrolling important. Dutta and Chiang [36] introduce the concept of crowdsourced enforcement of spectrum policies. Vaze and Murthy [115] also localize transmitters using binary sensors similar to our study. However, unlike our work, they do not consider the effect of

correlation among sensors and do not consider the cost of utilizing sensors.
Performance of low cost spectrum sensors: The authors in [36] assume complete knowledge about the performance of crowdsourced sensors which is not practical. [36] also assumes the sensors to be homogeneous which is generally not true in a crowdsourced environment. Spectrum monitoring using cheap crowdsourced sensors is not new [24, 87, 16] but they do not provide any insights regarding performance or reliability of sensing. We also show that analytical techniques [111] that model the sensor's detection performance are often simplistic and error prone. [45, 13] builds upon the analytical techniques providing corrections for hardware related aspects like I/Q imbalance, RF front-end non-linearities etc. Inspired by [104, 119], we use a data-driven approach to create performance models of heterogeneous spectrum sensors.
Sensor Selection and Fusion: A good amount of literature exists that study the problem of selecting sensors and combining the decisions of multiple sensors. Joshi and Boyd [63] show a method of selecting sensors using convex optimization, and empirically show that their results are usually close to optimal. Shamaiah et al. [103] propose a greedy selection of sensors that is close to optimal. Unlike our work, these studies consider sensors that follow normal distribution. To select sensors in the presence of intruders, we utilize a feature selection technique commonly used in the machine learning literature. This technique, known as maximum relevance minimum redundancy (MRMR) [92], is widely used to select relevant features when the features are correlated.

Combining the data of multiple sensors is a well-known problem in sensor networks. We utilize the rule provided by Chair and Varshney [20] which optimizes the overall performance when the individual sensor outputs are conditionally independent of one another. Different techniques of fusing multiple sensor decisions are presented in [1]. Some studies have also looked at the problem of distributed spectrum monitoring. Ghasemi and Sousa [43] propose using collaborative sensing across multiple sensors to better monitor spectrum. Dasari et al. [32] showed that detection of intermittent transmitters can be significantly improved by fusing the decisions of multiple sensors. Our work builds upon these studies to focus on detecting the presence of spectrum intruder.

2.7 Conclusion

In this work we address the problem of spectrum patrolling using crowsourced heterogeneous sensors. To the best of our knowledge this is the first work that models the performance of a spectrum sensor in a data-driven way. Our model provides significant improvement over state-of-the-art 'whitebox' models. Next we address the problem of sensor selection and fusion of heterogeneous sensors deployed over a region of interest to improve intrusion detection performance within a cost budget. We investigate different scenarios of homogeneous, heterogeneous and reconfigurable sensors. Our sensor selection algorithms perform significantly better than reasonable baseline heuristics. We highlight challenges of the patrolling problem in a cost-effective fashion using crowdsourced sensors and develop mechanisms to address them.

Chapter 3

Selection of Sensors for Efficient Transmitter Localization

3.1 Introduction

Wireless transmitter localization via analysis of the received signal from multiple receivers or sensors is an important problem. While the problem has been widely explored, the problem exposes new challenges in many emerging applications due to the constraints of the application. In this work, we are specifically interested in a distributed monitoring system where a set of distributed RF sensors are tasked to detect and localize transmitters. These transmitters could be of various type. For example, in certain spectrum allocation scenarios, unknown primary transmitters need to be detected/localized. Or, in spectrum patrolling scenarios, unauthorized transmitters need to be detected/localized [22]. Recent work has explored new approaches for such monitoring where the RF sensors are crowdsourced, perhaps using various low-cost spectrum sensing platforms [66, 87]. The crowdsourcing makes densely deployed, fine grain spectrum sensing practical by creating suitable incentive mechanisms [24, 66].

Crowdsourcing makes the sensing *cost-conscious*. The cost here could be incentivization cost, cost of power, backhaul bandwidth on the part of the spectrum owner or the opportunity cost – being low-cost platform, the sensors may be able to only sense smaller spectrum bands at a time. Thus, involving only a small number of sensors or sensors with low overall cost budget (for a suitable cost model) for sufficiently accurate localization performance is

critical. Prior works [66] that discuss sensor selection in this context only presents heuristics without any performance guarantees.

We do not use geometric approaches which rely on hard-to-model mapping of received power to distance. Instead, we use a hypothesis-driven, Bayesian approach for localization [19]. We focus on the optimization problem of selecting a certain number of sensors from among the deployed sensors such that an appropriately defined objective of localization accuracy is maximized. This optimization problem can also be used to solve the dual problem of selecting a minimum number of sensors (or sensors with the minimum total cost budget) to ensure at least a given localization accuracy. We adopt the framework of a hypothesis-driven localization approach wherein each hypothesis represents a configuration (location, power, etc.) of the potential transmitters and then the localization is equivalent to determining the most-likely prevailing hypothesis. See Figure 3.1. The hypothesis-driven framework does not require an assumption of a propagation model, and works for arbitrary signal propagation characteristics. The framework does, however, require prior training to build joint probability distributions of observation vectors for each hypothesis.

Our Contributions. In the above hypothesis-based framework, we develop an overall approach that enables selection of sensors that are most relevant to localize transmitters. In particular, we develop algorithms that aim to maximize localization accuracy for a given budget of number of sensors to be used for localization. More specifically, we make the following contributions in the paper.

1. We design a greedy algorithm (GA) that selects sensors iteratively to maximize the objective function of localization accuracy, under the constraint of number of sensors selected. We prove that GA yields a constant-factor approximate solution for the special case of the problem wherein there are only two hypotheses.

2. For the general case of more than two hypotheses, we design an alternate greedy scheme (called AGA) based on maximizing an auxiliary objective function. We prove that AGA delivers a solution that has (i) an auxiliary objective value within a constant factor of the optimal auxiliary objective value, as well as (ii) a localization error within a certain factor of the optimal localization error.

3. We optimize the time complexity of our developed algorithms by a

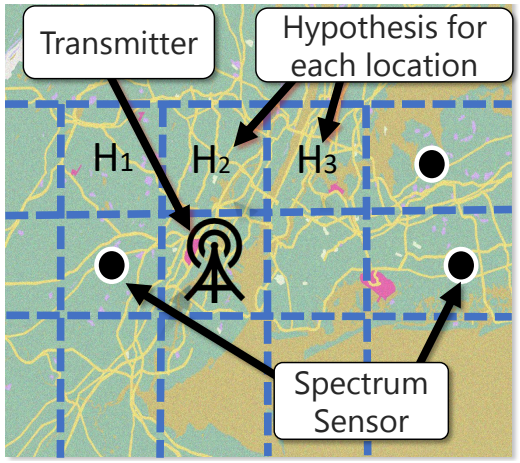


Figure 3.1: Hypothesis-driven localization. The figure shows the simple case of localizing a single transmitter with fixed power; thus, there is a hypothesis created for each potential location. Observations from deployed sensors are analyzed to determine the most likely prevailing hypothesis (and thus, location).

substantial factor, based on certain observations and reasonable assumptions. In addition, we generalize our techniques to more practical and useful settings.

4. We evaluate the performance of the developed algorithms over multiple evaluation platforms: (1) large-scale simulation using synthetically generated data using established signal propagation models, and (2) publicly available experimental data trace collected over an indoor WiFi network with 44 sensors, and (3) our own data collection using 18 outdoor software radio sensors in the 915 MHz band with a custom transmitter. Results show that our techniques outperform other state-of-the-art algorithm [66] substantially (up to a factor of 50-60%).

3.2 Background and Motivation

Problem Setting. The overall setting of the transmitter localization problem is as follows. Consider a geographic area, with a number of spectrum sensors deployed or available (if attached to mobile devices) at known locations. At

any instant, one or more transmitters are allowed to transmit signals (on a common frequency). Each deployed/available spectrum sensor senses and processes the aggregate received signal, and reports appropriate metric (i.e., total received power or signal strength) to a central server which estimates the location of the transmitter(s) using the maximum-likelihood hypothesis algorithm as described below. The overall objective of our paper is to develop techniques to select an optimal subset of sensors in order to accurately localize any present transmitters. Though our developed techniques naturally extend to the case of multiple transmitters, for simplicity, we implicitly assume at most a single transmitter present at any instant. We consider the extension to multiple transmitters in §3.3.5. We start with defining basic notations used throughout the paper.

Hypotheses, Observations, and Inputs. We discretize the given space into locations $l_1, l_2, \ldots.,$, and transmit power of a potential transmitter is similarly discretized into levels $p_1, p_2, \ldots$. We represent potential "configurations" of the possible transmitter by hypotheses $H_0, H_1, \ldots, H_m$, where each hypothesis H_i represents a configuration (l_i, p_i) of location l_i and transmit power p_i of a potential transmitter (see Figure 3.1). We use the convention that hypothesis H_0 corresponds to no transmitter being present. Localizing any potential transmitter is thus equivalent to determining the prevailing hypothesis. To do this, we use observations for a set of deployed sensors. We denote the observation vector of a subset of sensors $\mathbf{T}$ by $\mathbf{x_T}$ (we usually drop the subscript $\mathbf{T}$, as it is clear from the context).

Inputs. For a given set of sensors deployed over an area, we assume the following available inputs, obtained via a priori training, data gathering and/or analysis:

- Prior probabilities of the hypotheses, i.e. $P(H_i)$, for each hypothesis H_i.
- Joint probability distribution (JPD) of sensors' observations for each hypothesis. More formally, for each hypothesis H_j, we assume $P(\mathbf{x}_S|H_j)$ to be known for each observation $\mathbf{x}_S$ for the entire set $\mathbf{S}$ of deployed sensor. Note that this also gives us the JPD's of each subset $\mathbf{T} \subseteq \mathbf{S}$.

Maximum a Posteriori Localization (MAP) Algorithm. We use Bayes rule to compute the likelihood probability of each hypothesis, from a given

observation vector $\mathbf{x_T}$ for a subset of sensors $\mathbf{T}$:

$$P(H_i|\mathbf{x_T}) = \frac{P(\mathbf{x_T}|H_i)P(H_i)}{\sum_{j=0}^{m} P(\mathbf{x_T}|H_j)P(H_j)} \tag{3.1}$$

We select the hypothesis that has the highest probability, for given observations of a set of sensors. That is, the `MAP` Algorithm returns the hypotheses based on the following equation:

$$\arg\max_{i=0}^{m} P(H_i|\mathbf{x_T}) \tag{3.2}$$

The above `MAP` algorithm to determine the prevailing hypothesis is known to be *optimal* [35], i.e., it yields minimum probability of (misclassification) error. The above hypothesis-based approach to localization works for arbitrary signal propagation characteristics, and in particular, obviates the need to assume a propagation model. However, it does incur a one-time training cost to obtain the JPDs, which can be optimized via independent techniques [93].

Selection of Sensors for Localization. As mentioned above, in a typical setting, spectrum sensors may be deployed at pre-determined locations or available at certain locations (if part of mobile devices) to sense unauthorized signals and thus localize any unauthorized transmitters. Two immediate problems of interest in this context are: where to deploy given a number of sensors, and once deployed/available, which subset of sensors to select for localization. The latter problem of selection of sensors is motivated by the fact that, in most realistic settings, the sensors (or their mobile devices) are not tethered to AC power outlets and hence have limited energy resources. Moreover, spectrum sensors also incur cost in transmitting sensing data to the fusion/cloud center [88]. Thus, it is critical to optimize resources and costs incurred in localization of unauthorized transmitters, e.g., via the selection of an optimal set of sensors. Note that the sensor-selection problem can also be used to effectively *deploy* a given number of sensor, by assuming sensors available at all potential locations.

3.3 Optimal Sensor Selection for Intruder Localization

In this section, we address the problem of sensor selection for transmitter localization; informally, the problem is to select an optimal set of B sensors such that the overall probability of error of localizing a transmitter is minimized,

given appropriate JPDs as discussed in the previous section. We start with formulating the problem in the following subsection. In following subsection, we present a greedy algorithm for it and prove that it is guaranteed to deliver an approximation solution for the special case of two hypotheses. However, as shown, the greedy algorithm can perform arbitrarily bad for the general case of multiple hypotheses. Thus, we then modify our algorithm to use an "auxiliary" objective function and show that the modified algorithm delivers an approximation solution for the general case of multiple hypotheses albeit with a slightly worse approximation ratio. Finally, we discuss optimizing the computation complexity of the designed algorithms, certain extensions and other issues.

3.3.1 LSS Problem Formulation

We start with formally defining the optimization objective (probability of error or misclassification) for a given subset of sensors. Then, we formally define the sensor selection problem, hereto referred to as *Localization Sensor Selection (LSS)* problem. Throughout this section, we use hypotheses H_0 to represent the hypotheses with no transmitters present, and H_i to represent the hypotheses wherein a transmitter is present in i^{th} configuration.

Probability of Error ($\mathrm{P}_{\mathrm{err}}(\mathbf{T})$). Recall that, for a given observation vector, the MAP localization algorithm outputs the hypothesis that has the most likelihood among the given hypotheses. Thus, MAP can also be looked upon as a classification technique. Given a subset of sensors $\mathbf{T}$, we define the *probability of error* or misclassification as the probability of the MAP algorithm outputting a hypothesis different from the actual *ground truth* (i.e., prevailing hypothesis). The expected or overall probability of error is an expectation of the probability of error over all possible prevailing hypotheses and/or observation vectors $\mathbf{x_T}$ from $\mathbf{T}$. Our techniques generalize to the notion of distance-based localization error, as discussed in §3.3.5.

Formally, let $\mathtt{MAP}(\mathbf{x})$ be the output of the MAP algorithm on observation vector $\mathbf{x}$ from a given subset of sensors $\mathbf{T}$. Let $\delta_{\mathtt{MAP}(\mathbf{x})\neq i}$ be the binary predicate that denotes whether MAP algorithm outputs the hypothesis H_i or not; here, δ_p is the indicator function which is 1 if the predicate p is true and 0 otherwise. Given H_i as the ground truth and $\mathbf{x}$ as the observation vector, the probability of error $\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_i, \mathbf{x})$ can be written as:

$$\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_i, \mathbf{x}) = \delta_{\mathtt{MAP}(\mathbf{x})\neq i}. \tag{3.3}$$

If the observation vector $\mathbf{x}$ is not given, then the expected probability of error for a given ground truth H_i is just an expectation over the random variable $\mathbf{x}$. That is, $\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_i)$ can be written as:

$$\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_i) = \sum_{\mathbf{x}} \delta_{\mathtt{MAP}(\mathbf{x})\neq i} P(\mathbf{x}|H_i) = E_{\mathbf{x}|H_i}[\delta_{\mathtt{MAP}(\mathbf{x})\neq i}]$$

Since expectation of an indicator random variable is its probability, we can simplify the above equation as:

$$\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_i) = P(\mathtt{MAP}(\mathbf{x}) \neq i|H_i) \tag{3.4}$$

Above, the probability is over the random variable $\mathbf{x}$. Now, if the ground truth hypothesis is also not given, we can compute an expectation over all possible hypotheses. Thus, the (overall) *probability of error* for a given set of sensors $\mathbf{T}$ is given by:

$$\mathrm{P}_{\mathrm{err}}(\mathbf{T}) = \sum_{i} P(\mathtt{MAP}(\mathbf{x}) \neq i|H_i)P(H_i) \tag{3.5}$$

Localization Accuracy Function, $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$. To facilitate a greedy approximation solution, we formulate our sensor selection as a maximization problem—and thus, define a corresponding maximization objective. In particular, we define the *localization accuracy* $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$ as $1-\mathrm{P}_{\mathrm{err}}(\mathbf{T})$. Based on the above equation Eqn. 3.5, we get the expression for $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$ as:

$$\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) = 1 - \mathrm{P}_{\mathrm{err}}(\mathbf{T}) = \sum_{i} P(\mathtt{MAP}(\mathbf{x}) = i|H_i)P(H_i) \tag{3.6}$$

Localization Sensor Selection (`LSS`) Problem. Consider a geographic area with a set of sensors $\mathbf{S}$ deployed. Given a set of hypotheses and JPD's, as defined in previous section, the OSS problem is to select a subset $\mathbf{T} \subseteq \mathbf{S}$ of sensors with minimum probability of error $\mathrm{P}_{\mathrm{err}}(\mathbf{T})$ (or maximum localization accuracy $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$), under the constraint that $|\mathbf{T}|$ is at most a given budget B. The above formulation implicitly assumes a uniform cost for each sensor; we generalize our techniques to handle non-uniform sensor costs (see §3.3.5).

It is easy to show that the above `LSS` problem is NP-hard, via reduction from the well-known maximum-coverage problem. Thus, we develop approximation algorithms below; in particular, our focus is on developing greedy approximation algorithms. The key challenge lies in showing that the objective function satisfies certain desired properties that ensure the approximability of the algorithm.

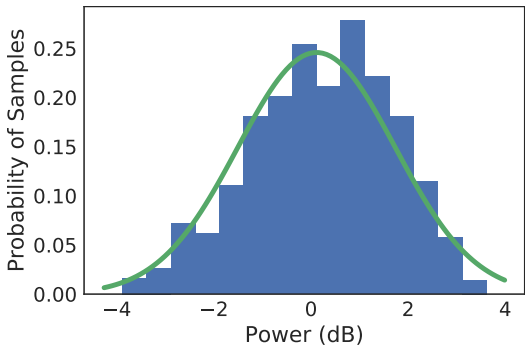


Figure 3.2: Distribution of the received power from a transmitter at an RTL-SDR sensor, and the Gaussian fit (green line) of the observed distribution.

3.3.2 Greedy Algorithm (GA)

In this subsection, we analyze a simple greedy approach and show that it delivers a constant-factor approximate solution for the special case of two hypotheses and Gaussian JPD's. In the next subsection, we present a modified greedy algorithm for the general case of more than two hypotheses.

Greedy Algorithm (GA): A straightforward algorithm for the `LSS` problem is a greedy approach wherein we iteratively select a single sensor at each stage. At each stage, we select the sensor that improves the localization accuracy $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$ the most. The algorithm iterates until the given budget B is reached. We call this algorithm Greedy Algorithm (GA); see Algorithm 2 for the pseudo-code.

Constant-Factor Approximation for 2 Hypotheses. We observe that when the spectrum sensors are deployed outdoors, the joint probability distribution (JPD) of the observation vectors is approximately Gaussian. See Figure 3.2, which shows the distribution obtained by a single RTL-SDR [99] based spectrum sensor and a USRP-based transmitter. This assumption of Gaussian JPDs allows us to derive close-form expressions for the objective functions, at least for the case of 2 hypotheses, and thus prove a performance guarantee of 63%. The result is stated in Theorem 1 below.

Theorem 1. *For the special case of two hypotheses and Gaussian JPDs, GA gives a subset* $\mathbf{T}$ *of sensors whose localization accuracy is at least 63% of the optimal.* □

We defer the proof of the above theorem to Appendix A.1, but the

performance guarantee of the greedy approach holds because the localization accuracy function $\mathrm{O}_{\mathrm{acc}}()$ can be shown to be *"monotone" and "submodular"* for the above special case. The function $\mathrm{O}_{\mathrm{acc}}()$ being *monotone* signifies that for a given $\mathbf{T}$ and a sensor $s \notin \mathbf{T}$, $\mathrm{O}_{\mathrm{acc}}(\mathbf{T} \cup \{s\}) \geq \mathrm{O}_{\mathrm{acc}}(\mathbf{T})$. Intuitively, the monotone property means that adding a sensor to a set of already selected sensors can never decrease the localization accuracy. Also, $\mathrm{O}_{\mathrm{acc}}()$ being *submodular* signifies that for any subsets $\mathbf{T}_1$ and $\mathbf{T}_2$ such that $\mathbf{T}_1 \subseteq \mathbf{T}_2$, we can show that for any sensor $s \notin \mathbf{T}_1, \mathrm{O}_{\mathrm{acc}}(\mathbf{T}_1 \cup \{s\}) - \mathrm{O}_{\mathrm{acc}}(\mathbf{T}_1) \geq \mathrm{O}_{\mathrm{acc}}(\mathbf{T}_2 \cup \{s\}) - \mathrm{O}_{\mathrm{acc}}(\mathbf{T}_2)$. Intuitively, the submodular property means that the "benefit" of adding a sensor s decreases over GA's iterations, i.e., as the selected set of sensor grows (from $\mathbf{T}_1$ to $\mathbf{T}_2$, here). It is well known that if an objective function is both monotone and submodular, then a greedy approach that iteratively maximizes the objective function will return a constant-factor approximate solution [85].

Algorithm 2 Greedy Algorithm (GA).

INPUT: Set of available sensors $\mathbf{S}$, budget B, objective $\mathrm{O}_{\mathrm{acc}}$
OUTPUT: Subset of sensors $\mathbf{T}$

$\mathbf{T} \leftarrow \phi$
while $|\mathbf{T}| \leq B$ **do**
  $L \leftarrow \mathrm{O}_{\mathrm{acc}}(\mathbf{T})$
  max $\leftarrow 0$
  for all $s \in \mathbf{S} \setminus \mathbf{T}$ **do**
    $M = \mathrm{O}_{\mathrm{acc}}(\mathbf{T} \cup \{s\}) - L$
    if $M >$ max **then**
      max $\leftarrow M$
      $r \leftarrow s$
  $\mathbf{T} \leftarrow \mathbf{T} \cup \{r\}$
return $\mathbf{T}$

Performance of GA for more than two Hypotheses. For the case of more than two hypotheses, GA no longer provides a constant-factor approximation. In fact, we show in Appendix A.2 via a counter-example that the $\mathrm{O}_{\mathrm{acc}}()$ is not submodular for more than 2 hypotheses, even if the given JPDs are Gaussian.

3.3.3 Auxiliary Greedy Algorithm (AGA)

In the section, we design an approximation algorithm for the general case of multiple hypotheses based on an auxiliary objective function. To do so, we first analyze the proof of Theorem 1 and see why it doesn't generalize if the number of hypotheses is greater than 2. This insight helps in defining an "auxiliary" objective function that is the key to designing the approximation algorithm for the general case.

Auxiliary Function. Let us consider a special case of MAP algorithm, viz., $\texttt{MAP}_{ij}$ which compares the likelihood of only two hypothesis H_i and H_j and returns the one with a higher likelihood. It is easy to formulate the objective function $\mathrm{O}_{\mathrm{acc}}$ in terms of $\texttt{MAP}_{ij}$ too. From Equation 3.6, we easily get:

$$\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) = \sum_{i=0}^{m} P(\bigcap_{j \neq i} \texttt{MAP}_{ij}(\mathbf{x}) = i | H_i) P(H_i) \tag{3.7}$$

$$\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) = \sum_{i=0}^{m} [1 - P(\bigcup_{j \neq i} \texttt{MAP}_{ij}(\mathbf{x}) = j | H_i)] P(H_i) \tag{3.8}$$

Above, $\mathbf{x}$ represents the observation vector for the set of sensors $\mathbf{T}$. For the case of two hypothesis, the above expression is just $\sum_{i=0}^{1}[1 - P(\texttt{MAP}_{ij}(\mathbf{x}) = j|H_i)]P(H_i)$ where j is 1 if i is 0 and vice-versa; Theorem 1 essential shows that the term $P(\texttt{MAP}_{ij}(\mathbf{x}) = i|H_i)$ is submodular. However, for the case of multiple hypothesis, computing the probability for a union of events involves product (and sum) of appropriate probability terms. Note that product of submodular functions need not be submodular, while sum of submodular functions is submodular. Thus, we *approximate* the above $\mathrm{O}_{\mathrm{acc}}$ () expression as follows, so that it is a sum of submodular terms. In effect, in defining the auxiliary objective $\mathrm{O}_{\mathrm{aux}}()$, we estimate the probability of union of events in the above equation by just taking a summation of the probability of events, i.e., we ignore the other terms involving subsets of events. Formally, we define the auxiliary objective $\mathrm{O}_{\mathrm{aux}}()$ for a set of sensors $\mathbf{T}$ as:

$$\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) = 1 - \sum_{i=0}^{m} \sum_{j \neq i} P(\texttt{MAP}_{ij}(\mathbf{x}) = j | H_i) P(H_i) \tag{3.9}$$

The above auxiliary objection function is submodular if the JPDs are Gaussian, as it is a sum of submodular functions ($P(\texttt{MAP}_{ij}(\mathbf{x}) = i|H_i)$ is submodular,

as per Theorem 1's proof). Note that, for a competitive algorithm for the original LSS problem, we also need to show that maximizing $\mathrm{O}_{\mathrm{aux}}$ () also somewhat maximizes the original objective function $\mathrm{O}_{\mathrm{acc}}$ ().

Auxiliary Greedy Algorithm (AGA). We now modify our Greedy Algorithm (Algorithm 2) to iteratively maximize the auxiliary objective $\mathrm{O}_{\mathrm{aux}}$ () instead of the original objective $\mathrm{O}_{\mathrm{acc}}$ (). We call this algorithm as Auxiliary Greedy Algorithm (AGA). From the submodularity of the $\mathrm{O}_{\mathrm{aux}}$ () for Gaussian JPDs, it is easy to see that AGA delivers a solution $\mathbf{T}$ s.t. $\mathrm{O}_{\mathrm{aux}}$ ($\mathbf{T}$) is within 63% of the optimal $\mathrm{O}_{\mathrm{aux}}$ () possible. The following lemma states that maximizing $\mathrm{O}_{\mathrm{aux}}$ also maximizes $\mathrm{O}_{\mathrm{acc}}$. See Appendix A.3 for a proof.

Lemma 1. *Let* $\mathbf{T}$ *be a subset of sensors already selected by AGA at some iteration. We claim that* $\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) \leq \mathrm{O}_{\mathrm{acc}}(\mathbf{T}) \leq 1-\frac{1}{k}(1-\mathrm{O}_{\mathrm{aux}}(\mathbf{T}))$*, where* k *is a value less than* m *that decreases as* $\mathbf{T}$ *grows (i.e., over AGA's iterations).* □

We empirically evaluate the value of k defined above in §3.4. The above lemma easily yields the below result.

Theorem 2. *For Gaussian JPDs, AGA delivers a subset* $\mathbf{T}$ *of sensors such that*

$$P_{err}(\mathbf{T}) \leq 0.37 + 0.63kP_{err}(\texttt{OPT}),$$

where k *is as defined in the above Lemma and* OPT *is the optimal solution.* □

3.3.4 Optimizing AGA's Computation Cost

In a straightforward implementation of AGA (akin to Algorithm 2 for GA), $\mathrm{O}_{\mathrm{aux}}$ function is computed (using Eqn. (3.9)) Bn number of times where n is the total number of sensors. Eqn. (3.9) requires m^2 computations of the expectation $P(\texttt{MAP}_{ij}(\mathbf{x}) = j|H_i)$, which, for Gaussian distributions, effectively requires computing the Eqn. A.1 and thus takes $O(B^2)$ time as it involves matrix multiplication of the observation vector of dimension B with the covariance matrix of dimension $B \times B$. Thus, the overall time complexity of a straightforward implementation of AGA is $O(m^2nB^3)$. As mentioned before, the number of hypotheses m can be large due to the large number of potential transmitter locations and power values; however, we can reduce the time complexity to $O(Bn)$ as discussed below, based on some observations and optimizations.

Reducing Number of Comparisons. Consider a sensor s whose benefit is to be computed in the `for` loop of Algorithm 2. Below, we show that effectively we only need to consider a constant number of (H_i, H_j) pairs in Eqn. (3.9) when computing s's benefit, and thus removing the m^2 factor from the time complexity. We implicitly assume a single transmitter in the below discussion, and later extend our argument to multiple transmitters. Let us use R to denote the maximum transmission "range" of the transmitter; formally, R is such that, beyond R, the probability distribution of the signal received from the transmitter is similar to the signal received when there is no transmitter present. We stipulate that any observation x_s at s, $P(x_s|H_{i1}) = P(x_s|H_{i2})$ for any two hypotheses H_{i1} and H_{i2} whose corresponding locations l_{i1} and l_{i2} are more than R distance away from s. The implication of the above observation is that, for a given sensor s, we can group all the hypotheses H_i with corresponding location l_i more than R distance away from s into one single "super" hypothesis H_s. Then, if the total number of hypotheses with corresponding locations within a distance of R from s is say G_R, then the total number of comparisons between pairs of hypotheses in Eqn. (3.9) is effectively only $(G_R + 2)^2$, involving G_R hypotheses, H_0, and H_s. The above brings down the overall time complexity of AGA to $O(G_R^2 nB^3)$. Note that G_R is essentially equal to the number of grid locations within a circle of radius R times the total number of power levels, and thus, can be considered as constant (does not vary across problem instances)—which reduces AGA's time complexity to $O(nB^3)$.

Independent Sensor Observations. If we assume that the observations across sensors are conditionally independent, the JPDs can be instead represented by independent probability distributions at each sensor location. In this case, the covariance matrix is purely diagonal, which allows us to "incrementally" compute the benefit of a sensor from one AGA iteration to another and thus reduce AGA's time complexity by an additional factor of B^2—and thus to $O(nB)$. See Appendix A.4 for details.

3.3.5 Generalizations

Weighted (Distance-Based) Objective Function. The probability of error $\mathrm{P_{err}}$ function penalizes *uniformly* for each misclassification. However, in general, it would be useful to assign different penalties or weights for

different misclassifications. E.g., Eqn (3.9) should be generalized to:

$$\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) = 1 - \sum_{i=0}^{m} \sum_{j \neq i} w_{ij} P(\mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i) P(H_i)$$

Above, w_{ij} is the weight function. We note that our techniques and proofs of performance guarantees generalize easily to the above generalized function, irrespective of the weight function. In particular, weight w_{ij} can be the Euclidean distance between the locations l_i and l_j corresponding to the hypotheses H_i and H_j. For the general case of multiple transmitters, where H_i and H_j may represent configuration of multiple transmitters, a minimum-cost matching based objective can be used to define the weight w_{ij}; if the number of transmitters in H_i and H_j are different, then an appropriately penalty for misses or false-alarms can be added to the weight.

Non-Uniform Sensor Cost. Another generalization of interest is to allow non-uniform cost for sensors, e.g., to prefer sensors with more (remaining) battery resources. Here, each sensor s may have a different cost $c(s)$, and the `LSS` problem constraint becomes: total cost of the selected set of sensors must be less than a given cost budget. For this version of the `LSS` problem, our algorithms need to be slightly modified in that we should pick the sensor that offers the highest improvement in the objective function per unit cost. To ensure a theoretical performance guarantee, we also need to use the "knapsack trick," i.e., to pick better of the two solutions: one returned by the modified algorithm, and the other the best one-sensor solution [67]. It can be shown the overall algorithm still offers a theoretical performance guarantee for submodular functions, but the performance ratio is reduced by a multiplicative factor of 2. The above model is useful when designing a "load-balanced" strategy to maximize network lifetime of a system—therein, the sensor-selection algorithm must be run periodically based on the remaining battery resources.

Multiple Transmitters. Till now, we have implicitly assumed that a single transmitter was present. Our techniques naturally generalize to the case of multiple transmitters, if we represent each *combination* of configurations of present transmitters by a separate hypothesis. Since the `MAP`, GA, and AGA algorithms are formulated in terms of hypotheses, they generalize naturally to localization of multiple transmitters. However, the key challenge arises due to the large number of hypotheses—exponential in the number of potential transmitters— and thus, the high time complexity of AGA. Fortunately, the

techniques discussed in previous subsection can be extended for the case of multiple transmitters as follows.

The key observation is that, for a given hypothesis H_i, the probability distribution of observations at a sensor s depends only on the configuration of transmitters in H_i that within a distance of R of s. I.e., for any observation x_s at a sensor s, $P(x_s|H_{i1}) = P(x_s|H_{i2})$ for any two hypotheses H_{i1} and H_{i2} that have the same configuration (locations and powers) for transmitters that are within a distance of R of s. The implication of the above observation(s) is that, for a given s, we can group the given hypotheses into *equivalence classes* based on the configuration of transmitters close of s, and to compute the benefit of a sensor s with AGA's iteration, we only need to compare pairs of equivalence classes (rather than the original hypotheses). The number of such equivalence classes is easily seen to be equal to G_R^T where G_R is the number of locations (grid cells) within R times the number of power levels, and T is the maximum number of transmitters possible/allowed within a range R of s (or any location). Thus, computation of benefit of s requires consideration of G_R^{2T} pairs of equivalence classes. If we assume T to be a small constant, then the overall time complexity of AGA reduces to $O(nB^3)$ as before, and to $O(nB)$ if we assume independence of sensor observations.

3.4 Evaluation

In this section, we evaluate the performance of our algorithms developed in the previous sections. We start with a description of the evaluation platforms used in our experiments.

3.4.1 Implementation

Implementation Technique. To evaluate whether AGA runs sufficiently fast to be feasibly used, we implement two distinct versions of AGA using python. The first version, called AGA-Basic, does not utilize the optimizations discussed in Section 3.3.4. The second version, called AGA-OPT, includes these optimizations. Each version utilizes multiple cores of a CPU using joblib library [62] to compute the gain of each available sensor in parallel. It also uses the numpy library to vectorize operations wherever possible to make execution as fast as possible. We run three different instances of AGA – with 100, 1600 and 4096 hypotheses. Each of these instances have 100

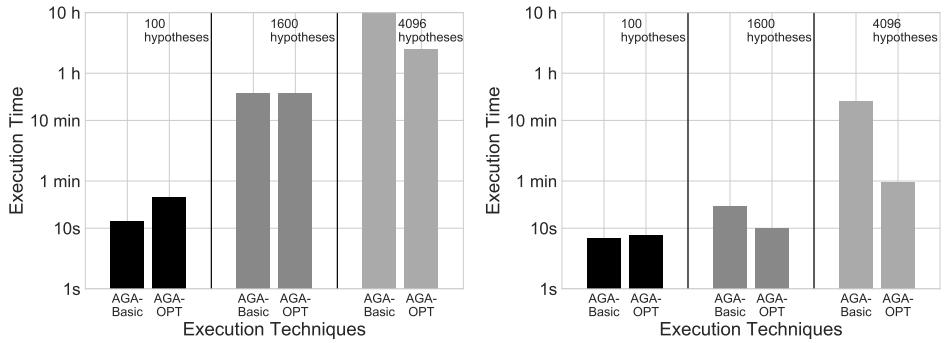


Figure 3.3: Execution time of AGA both with and without the optimizations on a (i) CPU and on a (ii) GPU.

available sensors and a budget of 20. We execute this on a Core i9-7900X CPU having a frequency of 3.30GHz and 20 cores.

Implementation on CPU. Figure 3.3(i) shows the execution time of these three instances. We note that for small instances, the execution time is relatively small. For example, for 100 hypotheses, AGA-basic only takes 13s to execute. However, this rises to 28 minutes for 1600 hypotheses and to over 10 hours for 4096 hypotheses. We also find that for small instances, the optimizations do not lead to much improvement due to the overhead of maintaining the data structures. However, there is a large improvement for 4096 hypotheses, where we get an execution time of 150 minutes using the optimized version.

Implementation on GPU. Although execution on CPU's using our optimizations is feasible, we further note that the bulk of execution time is spent on matrix operations. This suggests that execution on a GPU can lead to much better utilization of data-level parallelism, and further speed up execution. To evaluate this, we optimize the computation of the gain of the sensors using numba library [74] to execute it on a GPU. We utilize an nVidia GTX 1080 GPU having 2560 cores with a processor clock of 1.733 GHz. We then note the execution time for each of the three instances of both AGA-Basic and AGA-OPT.

Figure 3.3(ii) shows the execution times on a GPU. We note that execution is much faster on a GPU than on a CPU. For example, AGA-OPT now runs in only 8s, 10s and 57s for 100, 1600 and 4096 hypotheses respectively. This shows that AGA can run very fast on a system with GPU, with a speedup of up to 155 times on the large instances.

3.4.2 Evaluation Platforms

We use the following three evaluation platforms with varying fidelity of signal propagation characteristics, to demonstrate the performance of our techniques.

- *Simulation based on synthetic data.* To demonstrate the scalability of our techniques and the sensitivity of our algorithms to changes in settings, we consider a large geographic area of 4km by 4km, with signal path-loss values generated using the SPLAT! application for the Longley-Rice [25]. We use the noise in the sensor measurements (measured independently) to generate the required JPDs. We assume observations to be conditionally independent, thus representing the JPDs as set of probability distributions, one for each sensor and intruder configuration pair. Unless otherwise stated, for this large-scale platform, we use 100m x 100m grid cells giving 1600 potential locations, randomly deploy a transmitter at the height of 30m at a random power between 27-33 dBm which corresponds to roughly 250-750m of transmission range. We randomly deploy 100 spectrum sensors in the area.

- *Indoor Data.* We use publicly available data [91], which deploys transmitters and receivers at 44 locations at an indoor building of an area of $14m \times 14m$. Here, we use 1m x 1m grid cells, thus giving us a total of 196 potential locations and hypotheses. The transmitters transmit at a frequency of 2.4GHz.

- *Outdoor Testbed.* Finally, to evaluate our techniques in a more practical outdoor setting, we deploy our own testbed in a parking area of dimension $10m \times 10m$. Each grid cell has size of 1m x 1m. We place a total of 18 sensors on the ground. The sensors consist of single-board computers such as Raspberry Pi's and Odroid-C2's connected to an RTL-SDR dongle. The RTL-SDRs use dipole antennas. We collect raw Inphase-Quadrature (I/Q) samples from the RTL-SDRs [99], while transmitting data using a USRP-based transmitter from each grid cell at a height of 1.5m. We perform FFT on the I/Q samples with a bin size of 256 samples to get the signal power values, and then utilize the mean and standard deviation of the power reported for each of the sensors.

Metrics We evaluate the performance of a localization strategy in terms of two key metrics: (i) Localization accuracy, i.e., $O_{acc}(\mathbf{T})$, and (ii) Localization

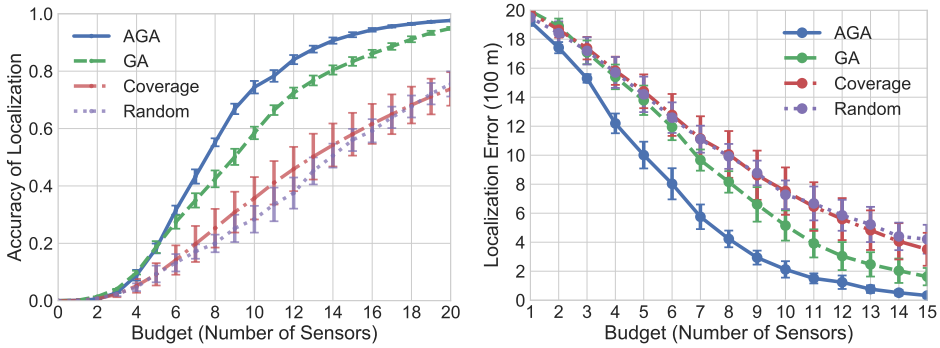


Figure 3.4: Comparison of various techniques for (i) Localization accuracy (O_{acc} ()), and (ii) Localization error, for varying available budget (number of sensors).

error, which weights the misclassification error by the Euclidean distance between the actual and the predicted location (§3.3.5).

Compared Algorithms. We implement both of our designed algorithms, AGA and GA. We also implement two other techniques for comparison purposes. The first technique, called `Coverage`, is the selection heuristic from the recent work [66], which essentially tries to maximize the "coverage" of the sensors in a greedy manner.[1] We also implement a `Random` algorithm which selects the required sensors randomly. We implement these algorithms in python, with extensive use of *numpy* library for vectorized operations. To ensure that our results are statistically significant, we run each of the algorithms 100 times; the range of values is plotted in each of the figures.

3.4.3 Simulation Based on Synthetic Data

Varying Budget. Figure 3.4 shows the performance of our techniques for budgets varying from 1 to 20 sensors. We observe that AGA and GA easily outperform other two algorithms in terms of both metrics, with AGA outperforming even GA quite significantly. For example, AGA outperform `Coverage` by up to 39% and 56% for localization accuracy and error respectively, while outperforming GA by 15% and 50% for the two metrics respectively.

Varying Number of Hypotheses. We now show the performance of our algorithms in terms of localization accuracy, for varying number of hypotheses.

[1] Their approach *Metropolis* performs worse than their greedy approach in open areas [66], and hence, not compared.

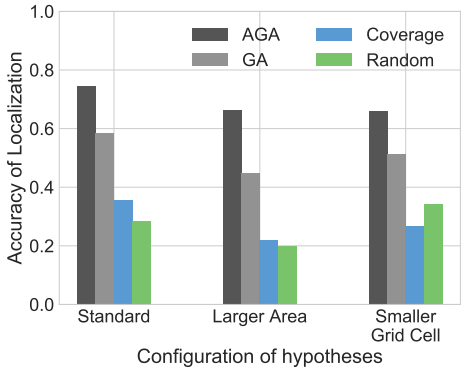


Figure 3.5: Comparison for configurations with different number of hypotheses, with a fixed budget of 10 sensors.

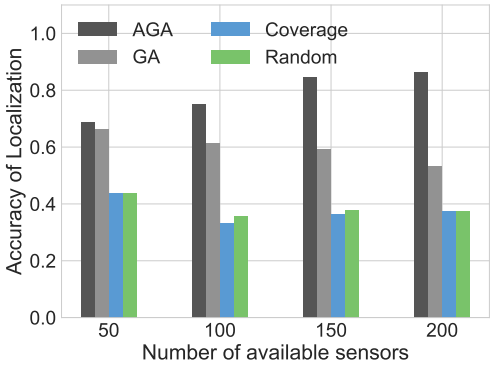


Figure 3.6: Comparison for varying number of available sensors, with a fixed budget of 10 sensors.

In Figure 3.5, we plot three different cases: (i) the default configuration of 100m by 100m grid cells, (ii) a larger area of 6km by 6km with 100m by 100m grid cells giving 3600 potential locations, and finally (iii) a configuration with default 4km by 4km area, but smaller 62.5m by 62.5m grid cells. First, we observe that AGA continues to outperform other techniques significantly across different cases, with the performance gap between AGA and others increasing with increase in number of hypotheses. Also, as expected, with increase in area and thus number of hypotheses, the accuracy of each of the algorithms falls, but deterioration in AGA's accuracy is much less compared to other techniques.

Varying Sensor Density, and Non-Uniform Sensor Costs. Figure 3.6 shows the accuracy of localization for varying sensor density (i.e., number of available sensors) with a fixed budget of 10 sensors. We observe that AGA continues to outperform other techniques, with localization accuracy of AGA significantly improving with increase in number of sensors. Surprisingly, however, the performance of other techniques reduces slightly with increase in number of sensors, across these specific set of experiments. We also evaluate performance of techniques under the setting of sensors with non-uniform cost. See Figure 3.7. We observe that AGA continues to outperform the other techniques in both metrics.

Empirical Evaluation of k Value. We now evaluate the k value as defined in Lemma 1. In particular, the performance guarantee of AGA depends on the value of k, with better performance guarantee for lower k (ideally, k

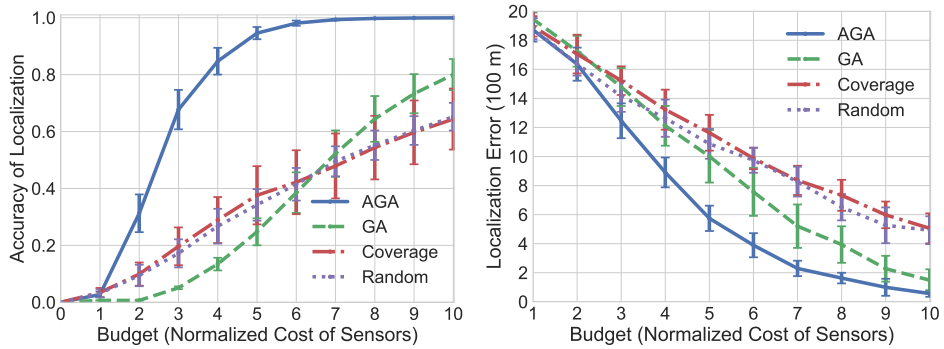


Figure 3.7: Comparison of various techniques, for sensors with non-uniform cost.

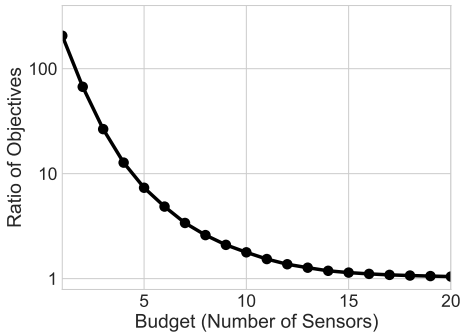


Figure 3.8: Values of k (from Lemma 1) for varying budget.

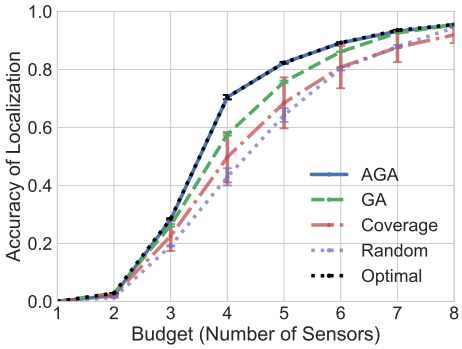


Figure 3.9: Comparison with an optimal algorithm, for small instances of the problem.

should be equal to 1). Figure 3.8 shows the value of k for varying budget. We observe that for a very low budget, the value of k is very large, but it reduces rapidly with increase in budget. In particular, for budgets of 10 and 15 sensors, the value of k is 1.78 and 1.19 respectively. This shows that AGA's performance guarantee as per Theorem 2 reaches its near-best for a moderately small budget.

Comparison with Optimal in Small Instances. To confirm AGA's performance with respect to optimal, we consider small instances of the problem and compare AGA with an optimal algorithm (exhaustive search). See Figure 3.9. We observe that AGA and optimal perform near-identically, with the optimal algorithm yielding at most 0.7% higher localization accuracy

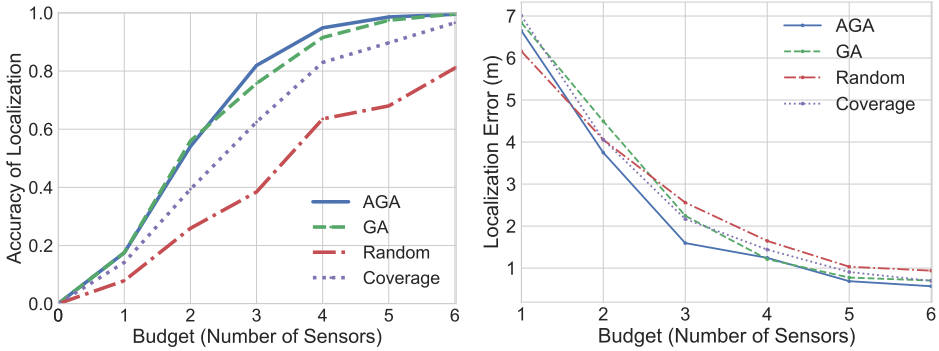


Figure 3.10: Performance over public indoor data.

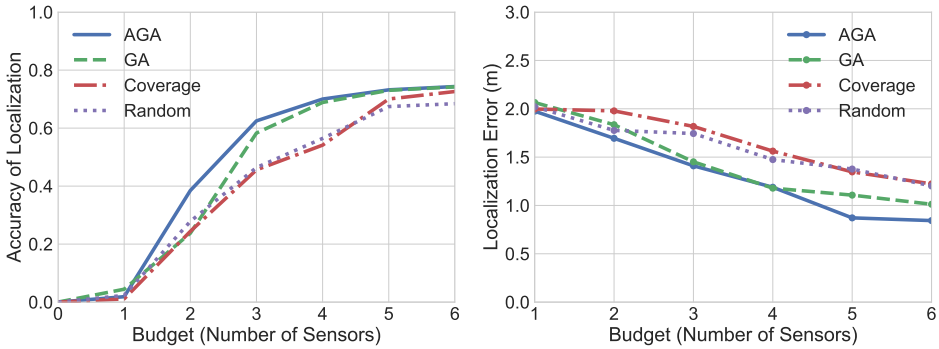


Figure 3.11: Performance over outdoor testbed data.

than AGA.

3.4.4 Evaluation in Indoor and Outdoor Testbeds

Indoor Data. We now evaluate our techniques over a publicly available data-trace taken in an indoor environment, as described in the previous subsection. See Figure 3.10. We again observe similar performance trends as in previous experiments, for both the performance metrics. The relatively smaller performance gap between AGA and GA is likely due to smaller a number of hypotheses.

Outdoor Testbed Figure 3.11 shows the performance of various algorithms over our outdoor testbed described in the previous subsection. Observe that AGA again performs the best among all techniques in both the metrics. As in the indoor testbed, the performance gap between the AGA and GA is less compared to the large-scale simulations due to a small number of hypotheses. Note that because of the noise in the dataset, the localization

accuracy reaches a maximum of only 75% even with all the 18 sensors.

3.5 Related Work

Sensor Selection for Transmitter Localization. A large number of works have developed techniques for detecting and localizing transmitters or intruders that emit radio signals [79, 123]. Note that the transmitter localization problem is slightly different from another well-studied problem of indoor localization [122], wherein a user is localized based on signal *received* from multiple transmitters; herein, the issue of selecting transmitters to localize the user has no incentive, and hence not been addressed before. To the best of our knowledge, none of these prior works on transmitter localization either have addressed the optimization problem addressed in the paper. The closest related works are [22] and [66] as discussed next. The work [22] focuses on *detection* of unauthorized transmitters using low-cost sensors in the context of shared spectrum systems; they consider the problem of selection of sensors in this context, and propose a heuristic with no performance guarantees. The key difference of our work from theirs is that they focus on detection of transmitters, which is a much simpler problem than localization of transmitters. In addition, [66] considers selection of sensors for transmitter localization, but with a objective of maximizing the "coverage" of the region by the sensors. They present heuristics without any performance guarantees. Nevertheless, we implement their approach and compare with our techniques (§3.4).

Sensor Selection in Sensor Networks. Sensor selection is a natural problem to address in the context of wireless sensor networks deployed to detect and/or localize an event or phenomenon (see [98] for a survey). Many of these works have leverage the submodularity property to develop greedy approximation algorithms. The closest work among these is that of [72] which shows approximability of the greedy approach for the problem of minimizing uncertainty in estimating a spatial phenomenon (e.g., temperature). However, in general, the key difference of our work with these works is our desired objective function (O_{acc} or P_{err})—and thus, the making the proof of monotonicity and/or submodularity of the objective function very different. In our case, we had to even circumvent the non-submodularity of the objective function O_{acc} by considering an appropriate auxiliary objective function.

Online Selection of Sensors. An alternate formulation of our sensor

selection problem could be to select sensors adaptively *based* on the observations of previously selected sensors. This online problem is similar to the adaptive stochastic optimization problem addressed in other contexts [118, 27, 71, 46]. However, in online selection, a sensor is selected based on analysis (which will incur non-trivial latency) of observations of previous sensors. This makes localization based on *near-simulatenous* sensor observations, required to localize intermittent transmitters, infeasible. Also, note that online selection needs to be done anew for each localization, which may be performed very frequently (e.g., every second or fraction of a second) in many applications, e.g., spectrum patrolling.

3.6 Conclusion

In this work, we have considered the hypothesis-driven approach for localization of transmitters, and developed techniques to optimize the localization accuracy under a constraint of limited resources. Developed techniques have been shown to yield provably approximate solutions. Our work can be instrumental in maximizing the network lifetime of a spectrum monitoring and/or patrolling system. Our future work focusses on improving our theoretical performance guarantee results, and developing similar sensor selection approximation algorithms for other localization approaches that are not hypothesis-driven.

Chapter 4

Online Selection of Sensors for Transmitter Localization

4.1 Introduction

In the previous chapter, we focused on offline selection of sensors (a priori selection), i.e. selection of sensors was done before beginning the process of localization. In practice, it is reasonable for an iterative algorithm (that selects sensors one at a time) to gather observations from already selected sensors, and select remaining sensors based on the gathered observations. Such an "online" algorithm should yield a higher detection accuracy than an offline algorithm, for the same number of sensors or budget cost. This second technique has the advantage of having much lower energy and backhaul bandwidth, albeit at the cost of higher latency. This online sensor selection problem is akin to the recently studied problem of active learning in machine learning [27].

In this version, our algorithm picks a sensor for probing, and its output is used to select the next sensors. This allows more accurate sensing while keeping the cost of running the sensors low, since they are switched on only intermittently. We use a greedy approach to select sensors based on an information theoretic metric. This algorithm also guarantees a constant-factor approximation of the optimal.

Policy: We encode the strategy of picking sensors in the form of a policy. Formally, a policy π is a rule specifying which sensor to pick next for probing based on the outputs of the sensors picked so far. For example, if we pick B

number of sensors for probing, then on completion of all probes the policy returns a sequence of the form: $\psi_\pi : \{(s_{\pi,1}, \mathbf{x}_1), \ldots, (s_{\pi,B}, \mathbf{x}_B)\}$. Since the actual values returned by the sensor probe $\mathbf{x}_k$ depends on noise, the sequence is also stochastic in nature. Thus, we need a technique of quantifying policies that give us more information about the transmitter.

Quantifying the information given by a policy: Given the hypotheses set $\{H_0, \ldots, H_m\}$, let $\mathcal{H}$ denote the true hypothesis. We define the entropy $\mathbb{H}(\mathcal{H})$ of the distribution as:

$$\mathbb{H}(\mathcal{H}) = E[-\log_2(P(\mathcal{H} = h))] = -\sum_{h=0}^{m} P(\mathcal{H} = h) \log_2 P(\mathcal{H} = h)$$

When a policy returns a sequence, the probabilities of each hypothesis change, and thus the entropy also changes. Since the sequence is stochastic, we compute the expected entropy and refer to it as the entropy of a policy. Mathematically, the entropy of a policy is defined as:

$$\mathbb{H}(\mathcal{H}|\pi) = E_{\psi_\pi}[\mathbb{H}(\mathcal{H}|\psi_\pi)] = -\sum_{h=0}^{m-1} P(\mathcal{H} = h|\psi_\pi) \log_2 P(\mathcal{H} = h|\psi_\pi)$$

We define the mutual information $I(\pi, \mathcal{H})$ between a policy π and $\mathcal{H}$ as the change in entropy:

$$I(\pi, \mathcal{H}) = \mathbb{H}(\mathcal{H}) - \mathbb{H}(\mathcal{H}|\pi) \tag{4.1}$$

Note that a policy having higher mutual information is better, since it reduces the entropy even more. The mutual information between π and $\mathcal{H}$ quantifies the amount of information given by the policy π about the ground truth in the average case. We now utilize the notion of mutual information to formally define the online sensor selection problem.

Online Sensor Selection (NSS) Problem Formulation. Formally, we define online sensor selection (NSS) problem as follows. We define an optimal policy π_{OPT} as one such that the mutual information between π_{OPT} and $\mathcal{H}$ is maximum. Then, our goal is to design an algorithm that generates π_{OPT} for any given prior distribution of $\mathcal{H}$. We must also ensure that the length of the sequence of a policy does not exceed the budget B. Formally, we represent this as:

$$\textbf{Maximize } I(\pi_{OPT}, \mathcal{H}) \text{ subject to } |\psi_\pi| \leq B. \tag{4.2}$$

We now show a technique of obtaining a policy to maximize the mutual information.

Algorithm 3 Online Greedy Algorithm (NGA) to select sensors.

INPUT: Set of available sensors $\mathbf{S}$, budget B
OUTPUT: Subset of sensors $\mathbf{T}$

$\mathbf{T} \leftarrow \phi$
while $|\mathbf{T}| \leq B$ **do**
  L $\leftarrow I(x_{\mathbf{T}}, \mathcal{H}) = \sum$ using Eq (4.1)
  for all $s_k \in \mathbf{S} \setminus \mathbf{T}$ **do**
    Compute $I(x_{\mathbf{T} \cup \{s_k\}}, \mathcal{H})$
    if $I(x_{\mathbf{T} \cup \{s_k\}}, \mathcal{H}) > L$ **then**
      $L \leftarrow I(x_{\mathbf{T} \cup \{s_k\}}, \mathcal{H})$
      $s \leftarrow s_k$
  $\mathbf{T} \leftarrow \mathbf{T} \cup \{s\}$
  Recompute $P(H_i|x_{\mathbf{T}})$ $\forall i = 0, \ldots, m$
return T

Online Greedy Algorithm (NGA): The optimization problem defined in Eq (4.2) is NP-Hard [27]. We therefore use a greedy algorithm that probes the sensor that increases the mutual information the most at each step. The algorithm is shown in Algorithm 3. At each step, once the sensor is probed, we recompute the probabilities of each $P(\mathcal{H} = h)$ and then picks the sensor for probing that has the highest mutual information. We continue picking sensors in this way until we reach the budget.

To show that NGA performs well, we first state a theorem from [27], and then modify it to make it applicable to NGA.

Theorem 3 ([27]). *The relationship between mutual information (denoted by I) obtained using the greedy method and the optimal (denoted by I_{OPT}) is given by:*

$$I \geq (I_{OPT} - \delta)\big(1 - \exp[-\frac{B'}{B\gamma \max\{\log_2(m+1), \log_2 \frac{1}{\delta}\}}]\big), \forall 0 < \delta < 1, \quad (4.3)$$

where B' and B are the number of probes in the optimal and greedy algorithm respectively. The other constant γ is defined as:

$$\gamma = \min_{i,j} \frac{7}{Q((\mathbf{p}_j - \mathbf{p}_i)^T \Sigma^{-1} (\mathbf{p}_j - \mathbf{p}_i))}$$

□

In our case, since the number of hypotheses is fixed, and $B = B'$, we see from Theorem 6 effectively gives us a bound that depends on γ. However, we note that γ increases with an increase in the amount of noise in the sensor outputs. Since samples drawn from the same sensor are independent of one another, the value of γ reduces with an increase in the number of samples drawn during a single probe. Thus, γ represents a tradeoff between the accuracy of our algorithm and the latency of a single probe.

Time complexity. We now look at the time complexity of our algorithm. We note that computing the entropy requires computation of the probability of all $m + 1$ hypotheses. Applying the Bayes formula in Eq (3.1) requires $O(m|S|)$ time. Computing this for each hypothesis, therefore, requires $m \times O(m|S|) = O(m^2|S|)$ time. Since we select the sensors over a total of B number of iterations, this gives us a total time complexity of $O(m^2|S|B)$.

Heterogeneous Sensors. For heterogeneous sensors, we modify our algorithm as follows. At each step, we select the sensor with the highest ratio of mutual information and cost if it is within the budget. If we exceed the budget, then we iterate over the sensors with the next highest ratio but with lower cost. We continue this until we either select a sensor or no sensor can be selected without exceeding the budget. While this algorithm for heterogeneous sensors does not provide any performance guarantee, we show using experiments (§4) that it works well in practice.

4.2 Evaluation

In this section, we first explain the evaluation setting and then compare the performance of our algorithms.

4.2.1 Evaluation Setting

Dataset: We deploy a USRP-based transmitter and an RTL-SDR based spectrum sensor [99] in a 50×50 m^2 field. We move around both the transmitter and the sensor in the field and collect data from over 150 pairs of transmitter and sensor location. At each pair of transmitter and sensor location, the sensor collects raw I/Q samples. We perform FFT over the data collected, and compute the mean and standard deviation for each pair of transmitter-sensor location. We assume that the distributions of samples from different sensors are conditionally independent.

Experiment Setup: To define the hypotheses, we build a grid of dimension 64×64. The center of each cell represents a possible location of a transmitter or sensor. A transmitter can be located at any cell, so there are 4096 (64×64) different hypotheses, i.e. $m = 4096$. We randomly generate 1000 sensors within the grid with uniform probability. When the parameters are not available from the dataset directly, we interpolate to obtain the parameters (mean and standard deviation) of the distribution as explained in [24].

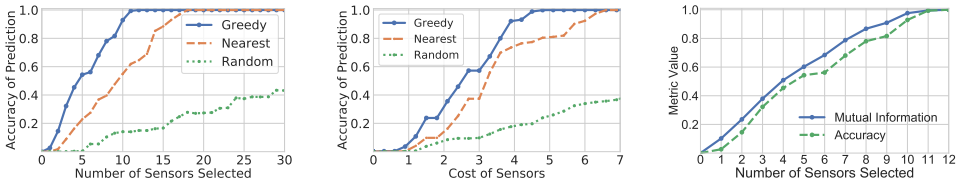
Evaluation Metric: We use localization accuracy to evaluate the algorithms. We use the metrics $\mathrm{O}_{\mathrm{acc}}$ and O_w respectively for the non-weighted and weighted cases. For the online algorithms we use prediction accuracy. The accuracy is calculated by doing a simulation of 100 tests, where in each test a subset of sensors try to localize a transmitter randomly generated with uniform probability. We then count the number of times the transmitter's location is correctly classified.

Evaluation Platform: We run the CPU-based algorithms on a machine using dual processor motherboard, where both processors are Intel Xeon E5-2640V4 with 10 cores and 20 threads [59]. Each core has a clock frequency of 2.4 GHz. Our algorithms are all implemented in python 3.6. Wherever possible, we use numpy to speed up the numerical computations and joblib [62] to use multiple processors in parallel.

Sensor Cost: For homogeneous sensors, we use the number of sensors as a metric of the cost. For heterogeneous sensors, we use the energy consumed as an estimate of the cost of running a sensor. To compute this, we perform multiple FFTs with different bin sizes from the set $\{2^8, 2^9, \ldots, 2^{15}\}$. We obtain the energy cost of FFT computation by running it on each of these configurations 1000 times on an Intel Core i5 processor, and use Intel RAPL drivers [55] to obtain their energy consumption. We then normalize the costs by taking the ratio of each energy value with the maximum and use the normalized energy as the cost of utilizing the sensor.

4.2.2 Accuracy of Online Greedy Algorithm (NGA)

Baseline Algorithms: For online sensor selection, we also use two baselines. The first baseline is random selection, as in OSS which does not update the priors at each iteration. The second baseline, called nearest, selects a sensor that is closest to the location having highest probability of transmitter being present. Intuitively, a sensor close to a hypothesis with high probability



(a) Homogenous Sensors (b) Heterogeneous Sensors (c) Mutual Information as Objective

Figure 4.1: Performance of our greedy online selection algorithms with 4096 hypotheses and 1000 sensors. For (a) and (b), we compare the accuracy of prediction with baseline algorithms. For (c), we compare the value of mutual information with accuracy of localization.

should be more likely to be selected. If the sensors are heterogeneous, we also weigh the gain of choosing it with their cost.

Observation: Figure 4.1 shows the performance of our online sensor selection, in terms of the accuracy of detection. We compare our algorithm along with the two baselines. We observe that our online greedy algorithm performs better than the baselines by a large margin except for the online nearest in the heterogeneous case. For homogeneous sensors, our algorithm accurately identifies the transmitter location in 12 sensors, compared to 18 and over 80 for coverage-based and random respectively. For heterogeneous sensors, our algorithm accurately identifies the transmitter location at a cost of 4.8, compared to 6.6 and over 21 for coverage-based and random respectively. We also note that online greedy selection uses just one-third the number of sensors compared to offline to achieve over 98 percent detection accuracy.

Performance of Mutual Information as Objective: We now confirm that mutual information is an appropriate objective to maximize the accuracy of classification. Figure 4.1(c) shows a comparison of the mutual information with the accuracy of detection. We note that an increase in accuracy closely correlates to higher accuracy of classification. This validates our claim that maximizing mutual information leads to a high accuracy of detection.

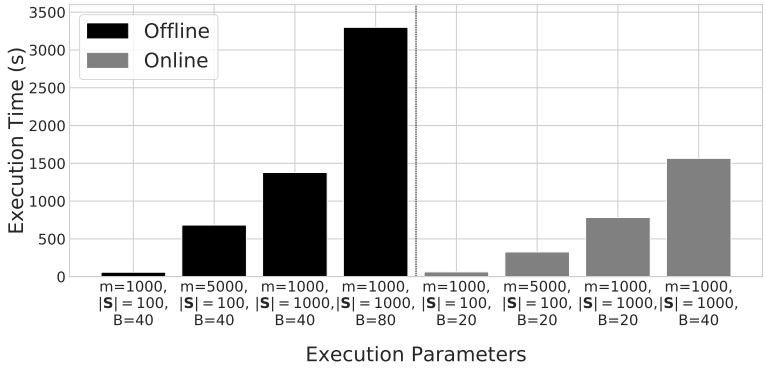


Figure 4.2: Execution time of both offline and online greedy algorithms for different numbers of hypotheses (m), number of sensors ($\mathbf{S}$) and budget (B).

4.2.3 Scalability of NGA

We further study the scalability of our algorithms. Figure 4.2 shows the execution time of our algorithms for different parameters of the problems. We show the execution time of both AGA and NGA. Note that for NGA, we choose a lower budget value of 20, as compared to 40, because NGA requires selection of fewer sensors than AGA to achieve similar levels of accuracy. Note that we only utilize a single core to run NGA, since it is fast enough that the overhead of parallelization leads to slower execution. We find that the execution time of NGA for even thousands of sensors and hypotheses is less than 15 minutes. Moreover, even on a single core implementation, it is much faster than AGA. This is both because AGA has higher complexity than NGA, and the budget required for NGA is usually lower. This shows that both AGA and NGA scale well to large problem instances, and can be used in practice.

4.3 Conclusion

In this chapter, we proposed a framework of online selection of sensors. We showed how online sensor selection can significantly reduce the cost of spectrum patrolling by switching on the most relevant sensors. We also compared the performance of our online algorithm with the offline algorithm introduced in the previous chapter. We showed that our online algorithm performs better than the baseline techniques.

Chapter 5

Quantifying Energy and Latency Improvements of FPGA-Based Spectrum Sensors

5.1 Introduction

In the last two chapters, we discussed techniques of intelligent sensor selection to reduce the cost of crowdsourced spectrum monitoring. An orthogonal approach of reducing energy consumption is to reduce the cost of running a single sensor. Current crowdsourced spectrum monitoring proposals (e.g., [17, 23]) utilize a single board computer such as Raspberry Pi or smartphone connected to a software-defined radio like RTL-SDR [99] or LimeSDR [80]. These devices are often energy-constrained in nature, especially in the case of outdoor deployments. Current state-of-the-art techniques can optimize energy either by reducing the duty cycle or using only a subset of the available sensors, both of which hurt accuracy of detection. Moreover, it is not possible to reduce latency of detection without significantly improving the compute power of the processor used. Since Raspberry Pis and smartphones have limited compute power, reducing latency requires using a more expensive processor board. Thus, a spectrum sensor that can significantly reduce both energy consumption and latency can improve the overall performance of spectrum monitoring.

In this work, we explore the use of field-programmable gate arrays (FPGAs) in spectrum sensors. An FPGA is a digital chip based on programmable logic.

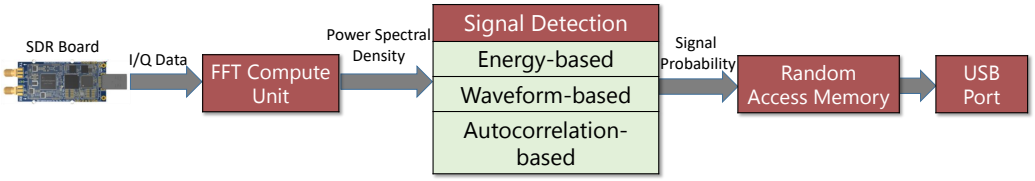


Figure 5.1: An overview of our FPGA-based spectrum sensor.

FPGAs allow algorithms to be implemented directly in hardware, producing significant performance and energy improvements. At the same time, their reconfigurability allows FPGAs to be adapted for use in differing conditions or to solve new problems without manufacturing new hardware. FPGAs can be viewed as a midway point between processor-based systems and the use of application-specific integrated circuits (ASICs). ASICs exhibit even higher efficiency than FPGAs, and with a lower per-part cost. However, the use of ASICs requires an expensive and time-consuming design and manufacturing process. This combined with their inflexibility mean that, although ASICs likely represent the best solution for a stable large-scale deployment of spectrum sensors, FPGA-based sensors can be better suited for the purpose of research.

We first observe that, when using a processor, computation of the power spectral density (PSD) and detection algorithms consume around 40 % of total energy cost, and incur 95 % of total latency. (Details of these experiments are explained in Section 5.2.) Thus, we implement these components in hardware on the FPGA to reduce energy and latency. By implementing a flexible system that allows easy use of several different FFT sizes and detection algorithms, we have produced a framework that enables a systematic evaluation of the real-world energy and latency costs of spectrum sensing on FPGA and in software on smartphones and the Raspberry Pi.

Using this system, we perform a set of measurements to understand the performance improvement and energy savings of our FPGA-based sensor. We benchmark both energy-based and autocorrelation-based detection using our FPGA. We find that the FPGA is able to achieve similar detection performance with 73 times lower latency than using a smartphone or a Raspberry Pi, while consuming up to 29 times less energy.

We summarize our contributions as follows:

- We demonstrate that computation speed is a performance bottleneck

on existing Raspberry Pi and smartphone-based sensors.

- We implement an FPGA based spectrum sensing system.
- We run a set of benchmarks using multiple parameters and algorithms, showing that the FPGA system has 73 times lower latency and 29 times lower energy consumption compared to a Raspberry Pi based sensor.

The rest of this paper is organized as follows. In Section 5.2, we explain the working of spectrum sensors and the motivation behind using an FPGA. Section 5.3 presents the design of our FPGA sensor. In Section 5.4, we describe the measurement results. Section 5.5 discusses related work, and we conclude in Section 5.6.

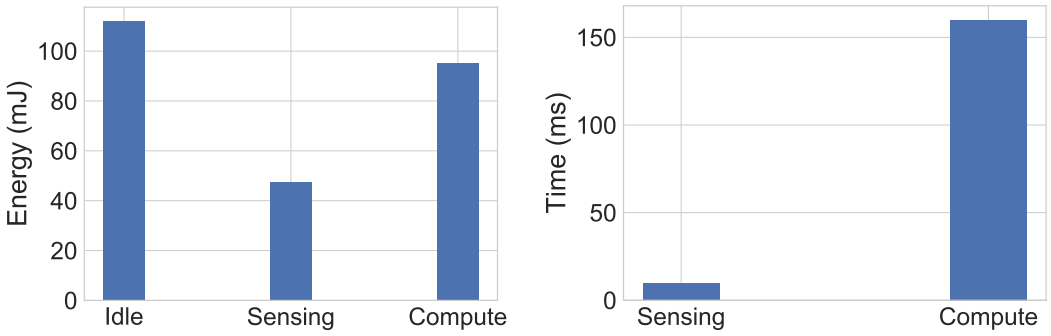
5.2 Background & Motivation

In this section, we describe the working of a spectrum sensor and explain the motivation behind using an FPGA.

5.2.1 Inside a Spectrum Sensor

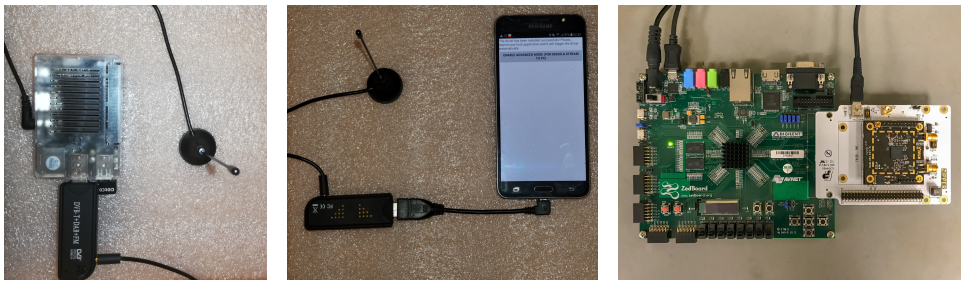
We first explain the process of signal detection. The signal is captured by the radio front-end (sensing unit) as complex numbers, representing discrete amplitude and phase values, in terms of I and Q samples. A detection algorithm directly operates on these samples to detect the presence of a signal on an attached compute platform (compute unit). The Fast Fourier transform (FFT) on the I/Q samples is a common operation used by many such algorithms.

Many types of detection algorithms are possible. We will focus on two common ones – energy-based and autocorrelation-based detection, also used in prior benchmarking studies [23]. In energy-based detection, the power of the signal within a given channel is compared with a threshold. If it is greater than the threshold, then the sensor considers the signal to be present. The fundamental computation here is an FFT on the I/Q samples to compute the Power spectral density (PSD), followed by summing of the squared magnitudes of each frequency bin in the FFT. The accuracy of energy-based detection depends on the number of I/Q samples and the number of frequency bins in the FFT. Increasing these parameters improves



(a) Energy Consumption (b) Latency

Figure 5.2: Energy consumption and latency of each unit of a Raspberry-Pi based spectrum sensor.



(a) Raspberry-Pi based sensor (b) Smartphone based sensor (c) FPGA based sensor

Figure 5.3: Our experimental testbed consisting of three different types of sensors.

the accuracy of detection by providing higher frequency resolution. However, this also costs time and energy, resulting in a trade-off between accuracy and cost.

In autocorrelation-based detection, we leverage the fact that many signals can have a periodic component and thus could be correlated over time, but noise is always uncorrelated. In this technique, we calculate the correlation of the signal with delayed copies of itself at specific periodic intervals. As the signal is correlated, the same patterns should get repeated at some lags, and thus a high degree of correlation should be found at these lags. In this way, it is possible to detect very weak signals, even in cases where energy-based

detection fails.

5.2.2 Motivation

We motivate the need for FPGA-based spectrum sensors using the following observations:

1. **High resource cost of computation**: To understand the proportion of energy consumed in computation, we plot the energy consumed by a Raspberry Pi in its different stages. We use a Monsoon Power Monitor [83] to measure the energy consumed when performing 1024-point FFTs on 100K samples of data. (Details of the experimental setup are described in Section 5.4.) Figure 5.2 shows the energy consumed individually by: (i) a Raspberry Pi if it is lying idle, (ii) by the sensing unit (SDR), and (iii) by the compute unit (when it runs autocorrelation-based detection) for the same amount of time taken by the compute unit. We obtain the idle energy by measuring the energy consumed during the entire amount of time taken to compute the power spectral density if no computation is performed. We compute the energy consumed by the sensing unit and the compute unit by subtracting the energy value obtained from the power monitor by the idle energy. We note that the energy cost of the compute unit is overall 37 % of the total cost. Thus, a significant amount of energy is spent on computation.

   We now look at the compute latency. We measure the latency of different portions of the algorithm by printing the timestamps at different stages of our software. We plot the latency of sensing and computing incurred while running an energy-based detector on 100K samples (Figure 5.2). The compute latency includes the time to perform the FFT and to run the detection algorithm. The sensing latency refers to the time to read data from the SDR front-end and send it to the buffer, assuming the SDR is already on. We again find that around 95 % of the latency is caused by the compute time. This demonstrates that faster computation can significantly speed up signal detection.

2. **Repetitive execution**: A second key observation is that the computations performed (running the FFT and the signal detection algorithm) are repetitive in nature. In other words, the same process is repeated many times during execution. This represents an ideal situation for hardware

acceleration, where a carefully-crafted hardware design can speed up the specific "hot spots" of the algorithm.

5.3 Our FPGA-based Spectrum Sensor

This section presents the design of our FPGA-based spectrum sensor, as shown in Figure 5.1. Our system is implemented using a Xilinx ZedBoard FPGA development board connected to a Myriad-RF1 transceiver board [84]. The ZedBoard uses a Xilinx Zynq-7000 All Programmable SoC (XC7Z020-CLG484-1); this chip combines a standard FPGA reconfigurable fabric with an embedded ARM processor, which we are using only for debugging. The Myriad-RF1 board includes a Lime Microsystems LMS6002D transceiver, which covers a frequency range of 300 MHz to 3.8 GHz, and uses a 12-bit ADC. The RF1 connects to the ZedBoard FPGA system via a Myriad-RF Zipper board, which provides a convenient physical interface to the FPGA.

As shown in Figure 5.1, I/Q samples flow out of the Myriad-RF1 (labeled "SDR Board") and into the FPGA. First, we designed logic to buffer the raw input data and synchronize it with the FPGA's internal clock rate. Then, our system feeds the data into an FFT core. We create the FFT modules using the Spiral FFT generator [81], a system that produces customized hardware for FFTs, parameterized by the FFT size, data precision, and other options. We used Spiral to produce different configurations, optimized for different FFT sizes. The FFT core feeds its result (the frequency domain representation of the input signal) into the signal detection unit.

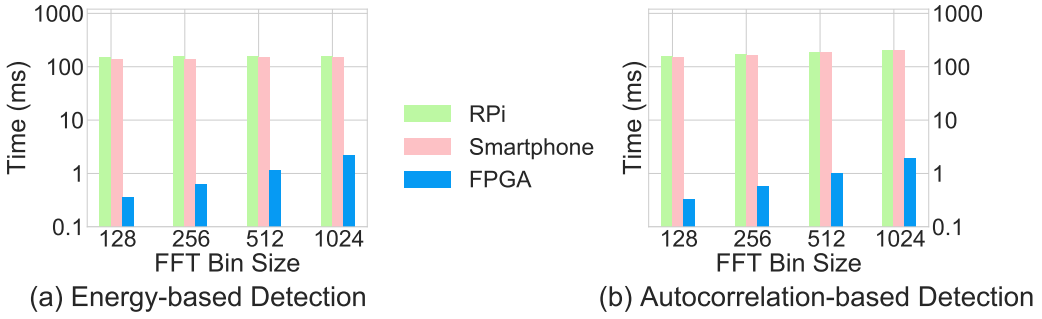


Figure 5.4: Comparison of latency of the compute unit of our FPGA, Raspberry-Pi and smartphone based sensors.

The signal detection unit can be pre-configured to run a detection algorithm based on energy detection, waveform detection, or autocorrelation. For the energy detection algorithm, we compute the energy over a specified (and configurable) range of the frequency spectrum. For the autocorrelation-based algorithm, we designed the hardware module to compute the autocorrelation in the frequency domain. Each of these algorithms outputs a stream of data that denotes the probability that a signal of interest is present.

For convenience, we can store the stream of result data in RAM using a Direct Memory Access (DMA) module or output it directly to pins. From memory, the data is then sent out to a computer or other device using either USB or UART ports.

5.4 Measuring Resource Usage

We focus on two distinct metrics of resource usage—(i) compute latency and (ii) energy consumption. We perform separate experiments on our Raspberry Pi, smartphone and FPGA-based sensors. Figure 5.3 shows photos of our testbeds: a Raspberry Pi 3 (model B) [97], a Samsung Galaxy S4 smartphone with an RTL-SDR based sensor, and the Xilinx ZedBoard FPGA with Myriad-RF. The FPGA system is constructed as described in Section 5.3; for the Raspberry Pi and smartphone, we have implemented the algorithms in C.

5.4.1 Computation Latency

We first compare the computation latency of the three types of sensors. For our FPGA, the latency is the time incurred from the first I/Q sample arriving in the FPGA board, to the output of the signal detection units. Because the FPGA system uses our custom hardware design, its latency is a deterministic value dependent only on the number of clock cycles required by the computation hardware and the FPGA's internal clock frequency. For the Raspberry Pi and smartphone implementations, we measure the average latency by timing the C code operating on a file of previously-stored I/Q data.

Figure 5.4 shows the latency of all three sensors executing the energy-based and autocorrelation-based detection algorithms. We find that the FPGA results in significant latency improvements– for energy-based detection, FPGA has around 73× lower latency than the Raspberry Pi and 69× less

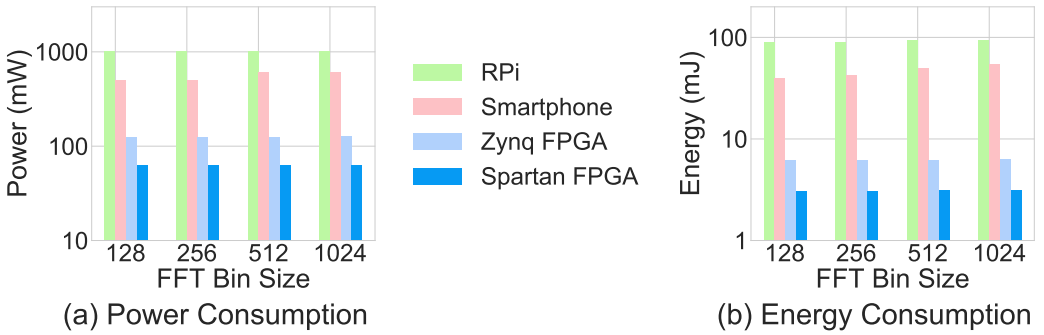


Figure 5.5: Comparison of power and energy consumption of our FPGA, Raspberry-Pi and smartphone based sensors.

than our smartphone. This is expected, since the FPGA is configured to use specialized parallel computational structures that can execute the detection algorithms more efficiently than the general-purpose processors used in the other systems. We also note that the latency increases by a similar absolute value with an increase in the FFT bin size. However, as the number of samples is constant, this increase in latency is not significant because the number of FFT computations reduces with an increase in the FFT bin size.

5.4.2 Energy Consumption

To measure the power and energy consumption of the Raspberry Pi and smartphone, we use a Monsoon Power Monitor [83]. We subtract the power consumption reported by the power monitor by the idle power to get the power consumed in computation. For all systems, we find the energy and power consumed by the system while running the energy-based detection algorithm with FFT bin sizes ranging from 128 to 1024. For both smartphone and Raspberry Pi, we avoid running other compute-intensive processes to avoid interference. For the Raspberry Pi, we run the algorithms remotely over a console by connecting it over a wired network.

Quantifying the exact power and energy consumption of the FPGA is non-trivial. The ZedBoard FPGA system is a development board that contains a variety of components (such as DRAM, flash memory, and an ARM processor) which are unused in our spectrum sensor. For this reason, physically measuring the power of the entire development board does not give a realistic measure

of the power consumed by the necessary components of the sensor. Instead, to obtain the power consumption of only the FPGA (without considering the unnecessary components of the development board) we use the FPGA vendor's power simulation tool (part of Xilinx Vivado) to produce estimates of the power consumed by the FPGA, which are labeled *Zynq FPGA* in our results.

Lastly, we note that the ZedBoard's Zynq FPGA is greatly over-provisioned for this application; our largest design uses $< 20\%$ of the chip's reconfigurable logic and only two-thirds of its arithmetic units. To quantify the further improvement available from using a smaller FPGA, we re-implemented the exact same functionality (running at an identical speed) on a smaller and lower-power Xilinx Spartan 7 FPGA (XC7S50FTGB196). Vivado's power simulation tool shows that the same design on the Spartan 7 FPGA requires approximately one half of the power as the Zynq FPGA. This design is labeled *Spartan FPGA* in our results.

Figure 5.5(a) shows the power consumption of these systems. We observe that the Zynq FPGA consumes about 8 times less power than the Raspberry Pi. Moreover, the power consumption can be further reduced by another 50 % by using the smaller Spartan FPGA, whose size and logic capacity are more appropriate for the application. Our results show that it is possible to reduce power consumption using an FPGA, but a careful design of the logic as well as the overall FPGA system and its board is necessary.

To measure the energy consumption, we run PSD on 100K samples with the sensor sampling rate set at 2 million samples per second. We then measure the energy consumed in computing the PSD of these 100K samples by integrating the result given by the Monsoon Power Monitor over the entire period. Figure 5.5(b) shows the energy consumption of the three sensors. Note that because of the effect of pipelining, the energy consumption is not necessarily equal to the product of power and latency. In general, we find a similar trend as seen in power consumption. The energy consumption of the Zynq FPGA and the Spartan FPGA are 14 and 29 times smaller respectively than the Raspberry Pi at an FFT bin size of 256. This confirms that a carefully designed FPGA can lead to significantly lower energy consumption.

5.5 Related Work

As discussed in previous chapters, there has been a lot of interest in spectrum monitoring using low-cost spectrum sensors. This includes our own work Specsense discussed in Chapter 2, as well as other works such as Electrosense [17] and Radiohound [69]. All of them utilize a large number of RTL-SDRs, with each connected via USB to a Raspberry Pi. Snoopy [125] proposes attaching a frequency converter to smartphones and utilizing individual smartphones as a spectral analyzer. In [23] authors benchmark latency and resource usage in similar Raspberry Pi and smartphone-based spectrum sensors.

The use of FPGAs for spectrum sensing is attractive due to FPGAs' flexibility, power efficiency, and computational abilities [109, 21]. However, the relative difficulty of FPGA implementation poses a significant barrier to implementation and to understanding important tradeoffs. Prior work has made a case for using FPGAs for spectrum sensing or related problems, but these are limited in the scope of their hardware considerations. For example, the FPGA system described by [14] considers only one algorithm (energy detection) with a single FFT size. In contrast to these studies, we focus on performing a systematic performance comparison of an FPGA-based sensor with the more widely available smartphone and Raspberry Pi-based sensors.

5.6 Conclusion

In this work, we systematically compare the performance of spectrum sensors for signal detection tasks, where the compute part of the sensors is based on embedded platforms such as Raspberry Pi, smartphone, vs FPGAs. We first made the observation that computation is the most energy-intensive process in spectrum sensing. We then described our implementation of FPGA-based sensor, which efficiently runs the computation entirely in hardware. We then compared its power consumption and latency with the Raspberry Pi-based and smartphone-based sensors. Our measurements show that the FPGA-based sensor consumes up to 29 times lower energy, and has around 73 times lower latency than a Raspberry Pi-based sensor.

Part II

Efficient Computation Offloading

Chapter 6

Parametric Analysis of Mobile Cloud Computing Frameworks using Simulation Modeling

6.1 Introduction

Mobile Cloud Computing (MCC) presents the opportunity to utilize unlimited resources of cloud based infrastructure to augment resource constrained mobile devices. Prototype implementations of MCC frameworks have demonstrated that offloading computation can significantly reduce energy consumed to execute an application on a mobile device [31, 29]. The key principle in MCC is to profile the energy footprint of individual tasks in an application, and then utilize the information to offload execution of energy hungry tasks to a cloud server to optimize energy usage on the mobile device. Task partitioning and task offloading decisions are constrained by several factors, like communication energy to offload the program states to cloud, network latency affecting application completion time, and tasks, involving sensors, which must be executed natively on the device. There are implementations that trade-off among these constraints [70, 121]. However, practical operating environment of a MCC framework is more complex due to several variable parameters, like network conditions, runtime workload, and hardware characteristics.

The key challenge in designing practical MCC frameworks is to adapt to changes in the operating environment. Variations in network conditions is one of the hardest to cope with. It has been shown that dynamically adapting

offloading decision based on varying network bandwidth can improve performance [75, 106]. Similarly, Zhang et al. showed the benefits of dynamically adapting data transmission rate to the cloud in presence of stochastic wireless channel errors [126]. Application workload is another source of variability to address while making offloading decisions. Exploiting dynamic execution patterns of an application can lead to better offloading decisions [41]. Barbera et al. implemented a tightly coupled device-cloud operating system that can overcome variations at different levels [3]. Even the diversity in smartphone hardware can lead to different offloading choices. For example, Lin et al. proposed the use of coprocessors, like GPUs in handheld devices, to arrive at better offloading decisions than those shown before [77]. The recurring theme in these works is that dynamic adaptation plays a crucial role in making better offloading decisions in MCC systems.

We observe that although system implementations have been effective in delivering performance gains, there is still a lack of in depth understanding of how individual parameters impact performance, as well as, influence each other. Given the complexity of these parameters, it is difficult to design controlled experiments in real environments. Therefore, we propose a simulation model that incorporates the parameters in a single model. This enables us to understand the interactions among different parameters that affect the offloading decision problem.

We summarize our contributions in this chapter as follows:

- We propose a formal model that incorporates different parameters that influence the task partitioning and task offloading in MCC systems.
- We analyze how various parameters used in offloading decision affect the performance of MCC systems. We report how optimization objectives, viz. energy consumed on a mobile device, and application execution time, are affected by various parameters, like application and cloud server features, degree of parallelism exploited and network characteristics.

The rest of this chapter is organized as follows. Section 6.2 discusses the working of an MCC offloading system. Section 6.3 explains the formulation. Section 6.4 shows the experiments and the corresponding inferences drawn. Section 6.5 concludes this chapter.

6.2 System Model

In this section, we present the architecture of a mobile cloud computing (MCC) system. The models of different components of the system, such as mobile application, communication network and the cloud system, are based on this architecture.

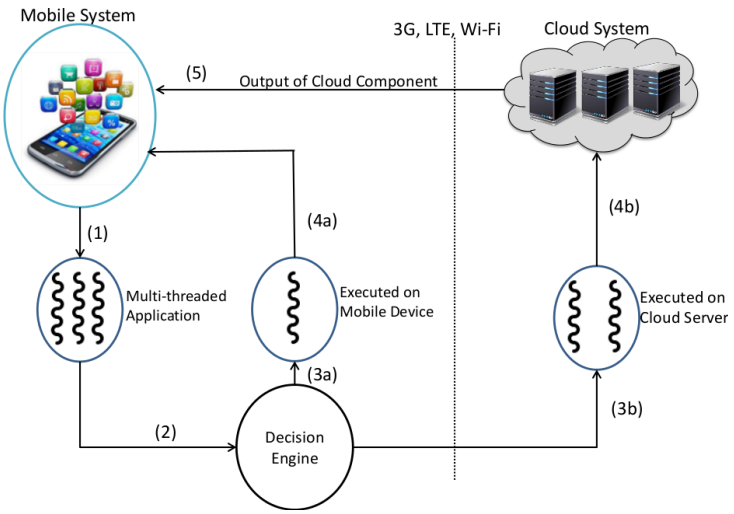


Figure 6.1: Execution of Mobile Application using cloud server. One component of the application is executed on the mobile device, while the other component is executed on the cloud server. The offloading decision engine is typically executed on a separate server.

Fig. 6.1 shows the architecture of an MCC system. An offloading decision engine partitions an application into two parts: one that executes on the mobile device, while the other is migrated to the cloud servers for execution. Communication between the mobile device and the cloud server uses the wireless interface on the mobile, which can be 3G, LTE or Wi-Fi enabled. We assume that the application source code resides on both the mobile device and the cloud server. Thus during execution only the program states need to be migrated to the cloud.

We assume that mobile applications have multiple threads. We model concurrent mobile applications using its call graph, which is a Directed Acyclic Graph (DAG) representing task invocations within the application. Each vertex in the DAG represents a task of the application, and each

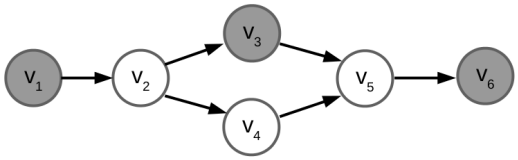


Figure 6.2: A general Directed Acyclic Graph (DAG) representing a mobile application. Methods shaded gray are native, i.e. they must be executed on the mobile device.

edge represents a dependency between two tasks. The set of tasks in the application is denoted by the vertex set $\mathbb{V}$, while the set of dependencies is represented by the edge set $\mathbb{E}$. Executing a task v_i locally on the mobile device incurs e_i^{loc} energy and t_i^{loc} time cost respectively. The application needs to be completed within a time deadline $\mathcal{D}$ and an energy budget $\mathcal{B}$. Some tasks, called native tasks, depend on mobile sensors and must always be executed on the mobile device.

Fig. 6.2 shows an application model where the application has three threads of execution. Two new threads are spawned at v_2. The threads join at v_7 and v_9 respectively. Moreover, three of the methods, v_1, v_4 and v_9 are native, i.e. they must be executed on the mobile device. This DAG model is general in nature, and can be used to model any mobile application.

The second component of an MCC offloading framework is the wireless network. Executing two tasks having dependency between them on different platforms (mobile or cloud) incurs a migration cost. Thus, if there exists an edge (v_i, v_j) to denote a dependency between tasks t_i and t_j, then this incurs a migration cost. This is represented by e_{ij}^{mig} and t_{ij}^{mig} to represent migration energy and time respectively. We assume that these costs do not vary once execution of an application begins. This is a standard assumption used by all MCC offloading frameworks.

The third component of an MCC offloading framework is the cloud system. The cloud system has higher computing resources than the mobile device. We represent the ratio of the computing speed of the cloud to that of the mobile processor by F. Thus, the time cost of executing a task t_i on the cloud system is equal to t_i^{loc}/F. Moreover, execution on the cloud system incurs no computation energy cost on the mobile device.

$\mathbb{V}$	Vertex set of the call graph
$\mathbb{E}$	Edge set of the call graph
v_i	A method in the call graph
(v_i, v_j)	A call invocation of the task v_j by v_i
t_i^{loc}	Local time execution cost of each method v_i
e_i^{loc}	Local energy execution cost of each method v_i
e_{ij}^{mig}	Energy migration cost of the call invocation (v_i, v_j)
t_{ij}^{mig}	Time migration cost of the call invocation (v_i, v_j)
F	Speedup of the cloud compared to the mobile device
$\mathcal{D}$	Time deadline given to application
$\mathcal{B}$	Energy budget given to application

Table 6.1: Symbols introduced in Section 6.2

6.3 Task Partitioning and Offloading: Formal Model

In this section, we formulate the offloading decision problem of a Mobile Cloud Computing (MCC) system for a mobile application. The task partitioning and offloading problem is NP-Complete for general concurrent applications [116]. Thus, we develop an integer-linear programming (ILP) problem to model this problem.

6.3.1 Problem Formulation

Let x_i be a binary decision variable such that:

$$x_i = \begin{cases} 1 & \text{if task } v_i \text{ is executed locally} \\ 0 & \text{if task } v_i \text{ is executed on the cloud} \end{cases}$$

Since there is a single decision variable to denote the location of execution of each method, every method in the call graph has to be executed (by choosing either $x_i = 0$ or $x_i = 1$).

Let the start time and execution duration of a task v_i be st_i and l_i respectively. Then, the completion time of a task is $st_i + l_i$. We know that all tasks must be completed by the given deadline $\mathcal{D}$. The time required for completing a task locally and on the cloud are t_i^{loc} and $t_i^{loc}/\mathcal{F}$ respectively.

x_i	Decision variable denoting execution location of v_i
st_i	Start time of executing the task v_i
l_i	Time taken to execute the task v_i
σ_{ij}	Decision variable denoting execution precedence
sm_{ij}	Start time of migration of the edge (v_i, v_j)
λ	Scaling factor used in optimization function

Table 6.2: Variables introduced in Section 6.3

Let σ_{ij} be a binary variable for all pair of tasks t_i and t_j such that:

$$\sigma_{ij} = \begin{cases} 1 & \text{if } v_i \text{ finishes execution before starting } v_j \\ 0 & \text{otherwise} \end{cases}$$

The variable σ_{ij} allows us to schedule the execution of tasks that have no dependencies between them in parallel.

Precedence constraint: We know that for all edges (v_i, v_j) in the graph, the task v_j has to be executed only after v_i has completed. This precedence constraint is represented using the variable σ_{ij}.

$$\forall (v_i, v_j) \in \mathbb{E},\ \sigma_{ij} = 1 \tag{6.1}$$

The nature of the above precedence constraint is such that if task v_i is executed after task v_j, then the opposite cannot be true. To enforce this property of precedence, we ensure that if for any such pair of tasks, if $\sigma_{ij} = 1$, then $\sigma_{ji} = 0$.

$$\forall v_i, v_j \in \mathbb{V},\ \sigma_{ij} + \sigma_{ji} \leq 1 \tag{6.2}$$

Concurrency constraint: First, we consider the case of a single processor on the mobile device. Thus, if the tasks v_i and v_j are scheduled by the offloading framework concurrently, i.e. $\sigma_{ij} = \sigma_{ji} = 0$, then at least one of the tasks must be executed on the cloud. In other words if $\sigma_{ij} = \sigma_{ji} = 0$, then at least one among x_i and x_j must be equal to 0. On the other hand, if both the tasks are executed locally, i.e. $x_i = x_j = 1$, then the two tasks must have some order between them.

$$\forall v_i, v_j \in \mathbb{V}, x_i + x_j \leq 1 + \sigma_{ij} + \sigma_{ji} \tag{6.3}$$

We have the following possible cases:

1. Tasks v_i and v_j have some order between them, i.e. $\sigma_{ij}+\sigma_{ji}=1$. Then, both the tasks v_i and v_j may be executed either on the cloud or on the mobile device, and so x_i and x_j remain unconstrained.

2. Tasks v_i and v_j do not have any order between them, i.e. they may or may not be executed concurrently. If they are scheduled for concurrent execution, then at least one among v_i and v_j must be executed on the cloud. In this case it is possible to have $x_i=0, x_j=0$; $x_i=0, x_j=1$ or $x_i=1, x_j=0$. On the other hand, they may also be scheduled so that execution of one method commences only after the other finishes. In this case, the two methods may be executed at any point, i.e. both x_i and x_j can have any value, since $\sigma_{ij}+\sigma_{ji}$ is set to 1.

Extending this for n processors, we note that if $(n+1)$ threads are scheduled for parallel execution, then at least one of them must be scheduled for execution on the cloud. To do so, we now pick all combinations of $(n+1)$ methods from the DAG. The constraint can then be mathematically represented as:

$$\forall v_i, ..., v_{i_{n+1}} \in \mathbb{V}^{n+1}, \sum_{k=1}^{n+1} x_{i_k} \leq 1 + \sum_{(k,l)\in\mathbb{V}^2} \sigma_{i_k i_l} \tag{6.4}$$

For each combination of $(n+1)$ methods, we ensure that if the number of tasks being executed concurrently are higher than the number of processors on the mobile device, then one or more of the tasks are scheduled for execution on the cloud. In that case, LHS of Equation 6.4 has a value equal to $n+1$. Thus, the amount of concurrency too has to reduce suitably so that the RHS increases in value. It is possible that executing the tasks sequentially gives a lower objective value. Then, the LHS of Equation 6.4 has a lower value.

We note that Equation 6.3 is a special case of Equation 6.4 corresponding to the case of a single mobile processor. This is because, by setting $n=1$ in Equation 6.4, we get:

$$\forall v_{i_1}, v_{i_2} \in \mathbb{V}\times\mathbb{V}, x_{i_1}+x_{i_2} \leq 1+\sigma_{i_1 i_2}+\sigma_{i_2 i_1} \tag{6.5}$$

Setting i_1 as i and i_2 as j in Equation 6.5, we get Equation 6.3.

Execution Time constraint: Executing a method v_i takes t_i^{loc} time if done locally on the mobile device, and t_i^{loc}/F on the cloud. Before commencing execution, output from all tasks v_j that immediately precede v_i, i.e. all

possible v_j such that $(j,i) \in \mathbb{E}$, have to be migrated to the location where v_i is executed. The time required for this migration must be considered along with the actual execution time. Migrating a task requires the time needed to bring all the data from its preceding tasks.

$$\forall v_i \in \mathbb{V},\ l_i = x_i t_i^{loc} + (1 - x_i) t_i^{loc}/F + \sum_{(j,i)\in\mathbb{E}} |x_i - x_j| t_{ij}^{mig}, \tag{6.6}$$

where t_{ij}^{mig} refers to the migration time between the edges (v_i, v_j). The migration time depends only on the data transfer d_{ij}, which is fixed for a particular edge. Since this formulation assumes constant bandwidth, the time cost of migration t_{ij}^{mig} is a constant.

The first two terms of the above constraint refers to the computation time locally and on the cloud respectively, whereas the last term refers to the time required to migrate the data dependency. If v_2 is executed on the cloud, then $x_2 = 1$ and the constraint gives computation time as t_2^{loc}/F. Depending on where v_1 was executed, migration cost might also have to be added to the computation cost of v_2 to get the total execution length of v_2.

Deadline constraint: The final task $v_{|\mathbb{V}|}$ has to complete execution before the given deadline.

$$st_{|\mathbb{V}|} + l_{|\mathbb{V}|} \leq \mathcal{D} \tag{6.7}$$

Energy budget constraint: The total energy consumption must not exceed the energy budget.

$$\sum_{i\in V} x_i e_i^{loc} + \sum_{(i,j)\in\mathbb{E}} |x_i - x_j| e_{ij}^{mig} \leq \mathcal{B} \tag{6.8}$$

Start time constraint: If method v_j is scheduled to execute after v_i (denoted by σ_{ij}), then the start time v_j is not less than the ending time of task v_i. Otherwise, we do not have any constraint on the start time of v_j, st_j. In that case, we reduce the right hand side of the constraint to a negative value to make st_j unconstrained. To do so, we use the largest time value in this formulation, which is the time deadline $\mathcal{D}$.

$$\forall v_i, v_j \in \mathbb{V}, st_j \geq st_i + l_i + (\sigma_{ij} - 1)\mathcal{D} \tag{6.9}$$

Finally, there can be two different objectives: minimizing energy consumption and minimizing execution time. The first objective, minimizing energy consumption, is:

$$\text{Min} \sum_{i\in V} x_i e_i^{loc} + \sum_{(i,j)\in\mathbb{E}} |x_i - x_j| e_{ij}^{mig} \tag{6.10}$$

This optimization function includes both migration energy and cost of local execution.

Similarly, the second objective, minimizing execution time can be written as:

$$\text{Min } st_{|\mathbb{V}|} + l_{|\mathbb{V}|} \tag{6.11}$$

Since the ending time already includes the time cost of migration, we do not need to explicitly add this to the optimization function of time.

Any one objective among energy or time can be chosen by an offloading framework for optimization. However, it is also possible to optimize both of them together by applying a suitable scaling factor (λ). The optimization function is then represented as:

$$\text{Min } \lambda(st_{|\mathbb{V}|} + l_{|\mathbb{V}|}) + (1-\lambda)(\sum_{i \in V} x_i e_i^{loc} + \sum_{(i,j) \in \mathbb{E}} |x_i - x_j| e_{ij}^{mig}) \tag{6.12}$$

6.3.2 Limitations of the Formulation

Our formulation has a few limitations. Firstly, it assumes that network transmissions succeed eventually. Wireless networks are inherently lossy and have a probability of failure. We assume that retransmissions at lower layers of network stack hide much of the transmission failures. Hence, considering the probability of failure in this formulation is not expected to affect the results of our study.

Secondly, we assume that the energy and time costs of each task is fixed on the mobile device. Thus, we ignore the effect of user input on the energy and time costs. Since offloading is mostly used for computation-intensive tasks, user input does not significantly affect execution costs.

6.4 Simulation Results

In this section, we study the sensitivity of the offloading solutions to various parameters through separate simulation experiments. These parameters include both changes in the properties of the applications, and of the overall offloading system. These experiments demonstrate the impact of parameters on the performance of the offloading system.

6.4.1 Simulation Settings

The simulation parameters and their values are shown in Table 6.3. Unless explicitly mentioned, these are the parameter values used for the experiments. The execution time for each method was chosen randomly with uniform distribution between 100 ms and 500 ms. The limits were chosen based on the range of values obtained from the trace log files of real Android applications [94]. Each experiment was repeated 20 times to ensure that any bias in the values of a particular instance do not affect the overall result.

The energy consumption value for each method was chosen randomly between 1 J and 20 J following uniform distribution. Most offloading frameworks utilize an energy model to determine at run-time the energy gain. If the system is heterogeneous and utilizes frequency scaling, then there is no direct correlation between execution time and energy consumption [18, 28]. Thus, taking random values of both execution time and energy consumption for each method is a reasonable assumption.

Parameter	**Range of Values**
Local execution time of each method (t_i^{loc})	100-500 ms
Local energy consumption of each method (e_i^{loc})	1-20 J [18, 28]
Data transferred for migration (d_{ij})	50 - 500 KB [121]
Energy for migration (e_{ij}^{mig})	$0.007d_{ij} + 0.005t_{ij}^{mig} + 5.9$ J [30]
Bandwidth	1 Mbps [121]
Round-trip Time or propagation delay (RTT)	70 ms [121]
Speed of cloud compared to mobile device (F)	10 [31]
Number of threads spawned from a particular method	0-2
Number of methods in each graph	20
Proportion of native methods in application call graph	30%
Number of experiments performed on each graph	20
Number of processors in mobile device	1

Table 6.3: Parameters used in simulation. These parameter values are used for all experiments, unless otherwise stated.

The size of data to be migrated during offloading is also required. To obtain the size of heap objects that have to be migrated, we refer to the work by Yang, Kwon, Cho, Yi, Kwon, Youn, and Paek [121]. The data transfer size varies between 50 KB and 500 KB.

The energy consumption of the network interface is calculated based on the energy model described by Balasubramanian, Balasubramanian, and

Venkataramani for a Wi-Fi interface [2]. In this energy model, the energy cost of data transfer is obtained as $0.007 \times d_{ij} + 0.005 t_{ij}^{mig} + 5.9J$, where d_{ij} is the number of kilobytes transferred and t_{ij}^{mig} is the total time required for migration. This cost includes the energy required to activate the wireless card, and connecting the device to the access point.

6.4.2 Performance Evaluation

We study the gains achieved by the use of MCC systems in terms of either energy consumption or execution time. We measure the gain in energy consumption by taking the ratio of energy consumption utilizing mobile cloud to that of energy consumption using local execution of the application:

$$\text{Gain in energy consumption} = \frac{\text{Energy consumption using cloud system}}{\text{Energy consumption without using cloud system}}$$

Similarly, the gain in execution time is given by:

$$\text{Gain in execution time} = \frac{\text{Execution time using cloud system}}{\text{Execution time without using cloud system}}$$

We solve the model derived for concurrent applications in Section 6.3 in these experiments. Based on the performance results, we can infer that the formulation used to model MCC systems works correctly.

The experiments are performed multiple times on different inputs to avoid any statistical error. We generate 10 graphs where the connectivity of the nodes, representing the function methods, is chosen randomly. Each graph has 20 nodes. The values of input parameters for each node, such energy consumed, and data transfer size for migration, are also varied randomly within the range described in Table 6.3. The average gains in execution time and energy consumption are then calculated based on the results derived from the 10 experiments on the 10 application call graphs generated.

Fig. 6.3 and 6.4 show the gains observed in execution time and energy consumption using our formulation. We note that the deviations observed from the mean are relatively small (within a range of 0.2). This effectively confirms that the conclusions drawn from these results remain valid irrespective of the graph layout.

We observe that as the scaling factor (λ) used during optimization (Equation 6.12) increases, there is a small increase in gain (around 5%) observed in

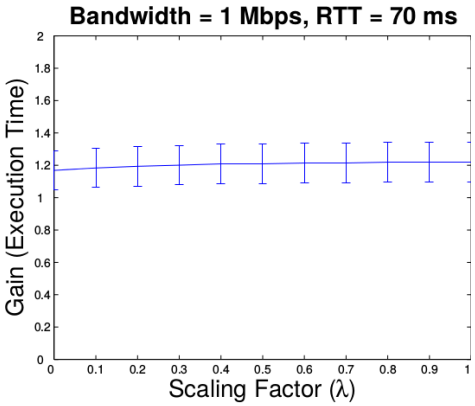


Figure 6.3: Comparison of gain observed in execution time for ten different random graphs with increase in scaling factor (λ) along with the observed deviation from the mean.

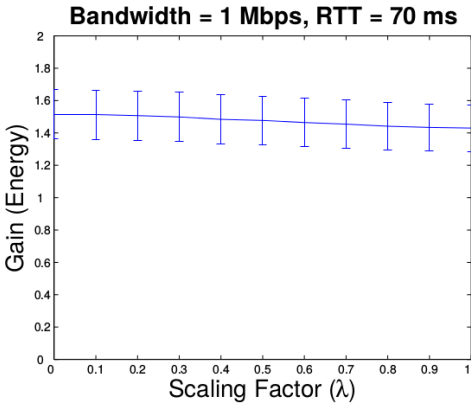


Figure 6.4: Comparison of gain observed in energy consumption for ten different random graphs with increase in scaling factor (λ) along with the observed deviation from the mean.

execution time. However, this comes at the cost of an increase in energy consumption. Thus, we conclude that execution time and energy consumption are conflicting objectives in some cases. An attempt to reduce the execution time might increase the energy consumption, and vice-versa.

The observation that execution time and energy consumption are conflicting objectives is explained by noting that a method having low execution time might consume a lot of energy. Thus, offloading such a method might end up increasing the execution time but reducing energy consumption. The opposite case, i.e. increasing the energy consumption but reducing the execution time due to offloading is also possible.

6.4.3 Impact of Application Variables

We investigate the effects of variabilities in different programs on the performance of MCC systems. Variabilities in a program could be due to difference in the number of native function calls, or the degree of parallelism in the code. Both the factors can be reflected in the graph representation of the program we have shown earlier. We study the effect of both of these factors on execution time and energy consumption.

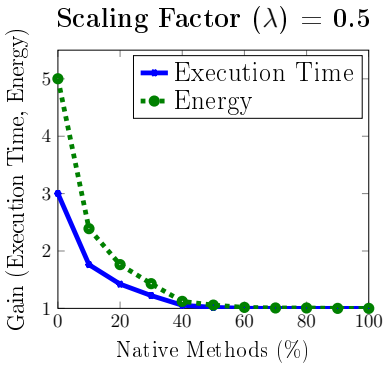


Figure 6.5: Effect of increasing the number of native method in the application on execution time and energy consumption. Scaling factor in optimization is set to 0.5, i.e. equal priorities are given to time and energy optimization.

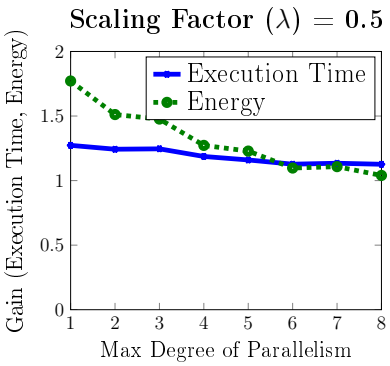


Figure 6.6: Effect of increasing the maximum number of threads that can be spawned by a particular method. Scaling factor in optimization is set to 0.5, i.e. equal priority is assigned to time and energy optimization.

Effect of Native Methods: To study the impact of native methods, we gradually increase the probability of a method being native in the random graph. For each value of probability, we note the average gains in execution time and energy consumption.

Fig. 6.5 shows the effect of percentage of native methods on performance. We note that the increase in the percentage of native methods reduces the gain in both energy consumption and execution time. Moreover, when all the methods are native, the gains observed in both execution time and energy consumed are equal to 1.

These observations can be explained by noting that increasing the number of native methods forces more local execution of the application. This reduces the advantages of using the cloud. In the extreme case, when all the methods are native methods, then there is no gain in either energy consumption or execution time. This is expected, since the application executes locally and cannot take advantage of offloading.

We also observe that when the number of native methods is low, the reduction in performance with an increase in the number of native methods is non-linear. Thus, a small increase in the number of native methods leads to a very high drop in performance in terms of both execution time and

energy consumption. This observation could be important for application developers trying to leverage the benefits of MCC.

This non-linear decrease in performance can be explained by observing that when the number of native methods is low, it is possible for the method that is spawning the threads itself to be migrated to the cloud. This avoids separate migration of multiple threads and therefore, reduces the cost of migration. Thus, very low number of native methods gives very high gains in both execution time and energy consumption.

Effect of Number of Threads spawned: To study the effect of number of threads spawned, we increase the maximum outgoing degree of each vertex. We have varied the maximum degree of each vertex from 1 to 8. We report the average gains for both execution time and energy consumption.

Fig. 6.6 shows the effect of increasing the number of threads spawned at each method of the application graph on performance. We observe that increasing the number of threads has almost no effect on execution time. However, the energy consumption involved increases with an increase in the number of threads.

To explain these observations, we note that increasing the number of threads increases both time and energy due to migration. However, the time spent on migration is mitigated by better utilization of parallelism. This effect does not apply to energy consumption, and so increases with an increase in the number of threads.

6.4.4 Detailed Study of Model Parameters

In order to understand the effect of individual environment parameters, we select a single representative graph using the layout shown in Fig. 6.2. This DAG is general in nature, and does not make any additional assumptions. It contains multiple threads with each of the threads spawned from the same method, but joins at different methods. Moreover, one of the threads also contains a native method. This ensures that all the different threads have conflicting requirements and thus, the decision problem becomes harder to solve.

Effect of Scaling Factor (λ): To study how the scaling factor affects the performance gains in this graph, we plot the energy and execution time for different values of the scaling factor. This result indicates how to balance the the two objectives, energy consumption and execution time, in the optimization objective function. Fig. 6.7 shows how varying the scaling factor affect both

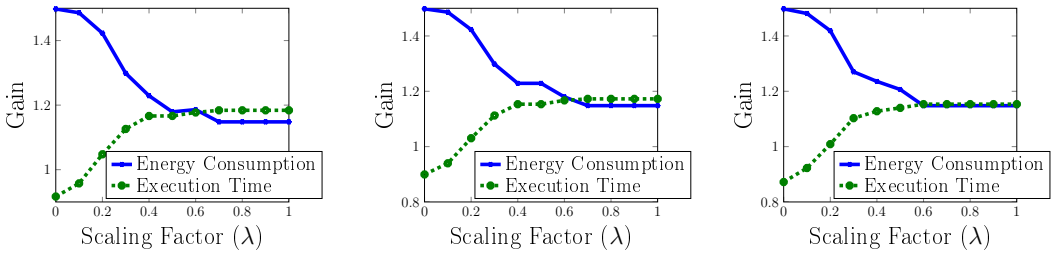


Figure 6.7: Comparison of gain in energy consumption and execution time for different scaling factor values (λ). Round-trip Time (RTT) is used to measure cloud response time.

energy consumption and execution time.

We observe that as the scaling factor increases, the total gain in execution time also increases. However, this comes at the cost of lower gains in energy consumption. When the scaling factor λ is set to 1, i.e. the optimization function considers only execution time, there is a speedup of 40% in execution time. However, there is no improvement in energy consumption. The opposite situation is observed when the scaling factor λ is set to 0. In this case the optimization function considers only energy, and so there is an improvement in energy consumption. This improvement in energy consumption comes at the cost of increased execution time as compared to local execution by around 10%.

This observation once again shows that in this graph too, execution time and energy consumption are conflicting objectives. Aggressively optimizing the execution time increases the energy consumption, and vice-versa. We have already explained the reasons behind this observation in Section 6.4.2.

We also observe that the gains in energy consumption and execution time are similar in all the three sub-figures. This means that the round-trip delay time does not affect the performance at this bandwidth. This observation can be explained by noting that at a bandwidth of 1 Mbps, most of the time is spent in transmission. Thus, the propagation delay is comparatively smaller, and hence does not affect performance.

Moreover, we also note that at a scaling factor of around 0.6, the gains in energy consumption and execution time are almost equal. This shows that irrespective of the cloud response time, a scaling factor equal to 0.6 balances both energy consumption and execution time. We explain this by noting that the conflicting requirements of time and energy are balanced when the scaling factor is equal to 0.6.

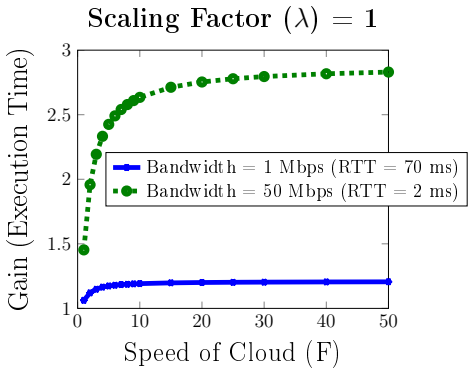


Figure 6.8: Comparison of execution speed-up with increase in speed of the cloud system. Optimization function here considers only execution time i.e. scaling factor $\lambda = 1$

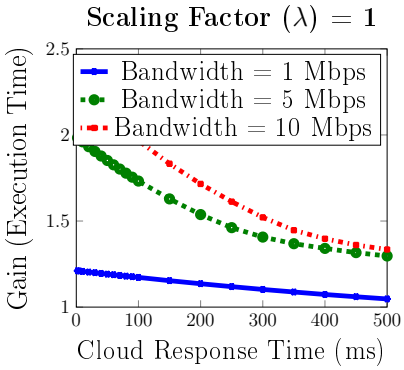


Figure 6.9: Effect of round-trip delay (RTT) on execution time at different bandwidths. RTT is taken as an approximate measure of cloud response time.

Effect of Speed of Cloud (F): We vary the speed of cloud ($\mathcal{F}$) as compared to the mobile device from 1 to 50. For each value of $\mathcal{F}$, we find the gain observed in execution time. We have studied the execution time for two cases – for very high and moderate bandwidths. Since speed of cloud does not have any effect on energy consumption, we have not included it in this study. Thus, the scaling factor (λ) has been set to 1 to ensure that only execution time is optimized by the system.

Fig. 6.8 shows the result of increasing the speed of cloud on execution time. We first note that due to utilization of parallelism, even a cloud system with very low speed gives an improvement of around 50% in execution time. At a low bandwidth, any improvement in the speed of cloud has very little effect on the total execution time. However, at high bandwidths, i.e. when migration time is low, the gain in execution time saturates at a much higher value of $F = 50$.

These observations can be explained by the fact that migration consumes more time than actual execution in case of moderate bandwidth. This explanation is further confirmed by the fact that at high bandwidth, much higher improvement in execution time is observed when the speed of cloud is increased. Further investigations on the effect of network bandwidth have

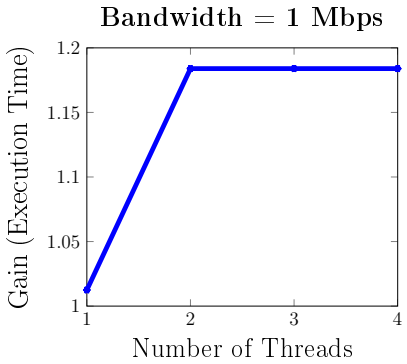


Figure 6.10: Execution speedup measured with respect to increasing parallelism. We increased the number of threads that can run in parallel to measure the speedup in execution.

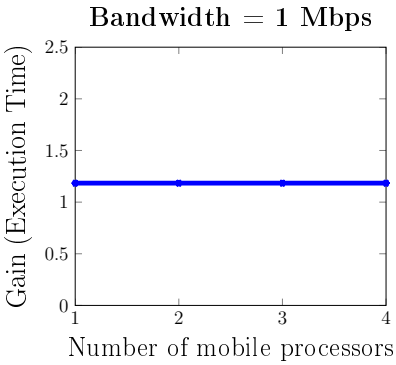


Figure 6.11: Execution speedup measured with respect to increasing number of processors on the mobile device. λ denotes scaling factor used in optimization function.

been discussed later in this section.

Effect of Cloud Response Time: We now study the effect that propagation and transmission delays have on the total execution time (Fig. 6.9). We study how varying the cloud response time at three different bandwidths (1 Mbps, 5 Mbps and 10 Mbps) affect the execution time. Since the energy consumption remains same irrespective of the transmission and propagation time, we do not consider it here.

Fig. 6.9 shows the effect of increase in the propagation delay on execution time at the three different bandwidths. We observe that, at a low bandwidth of 1 Mbps, any increase in propagation delay has no effect on the execution time. However, this does not hold true at high bandwidths. At bandwidth of 10 Mbps, for instance, an increase in the RTT from 2 ms to 100 ms reduces the execution time by 20%.

This result can be explained by the fact that at high bandwidths, the propagation delay is higher than the actual transmission time during migration. However, at low bandwidths, the transmission time is much higher, and so most of the time is taken up by transmission. Thus, increasing the value of response time has no effect on execution time at low bandwidths, but has an adverse effect at high bandwidths.

Effect of Parallelism on Execution Time: We now investigate the

gain on speedup with an increase in parallelism. Some offloading frameworks such as MAUI [31] do not exploit any parallelism in order to have a simpler mathematical formulation. Our objective is to determine if utilizing parallelism leads to any significant improvement in overall execution time.

In our first experiment, we increase the parallelism that can be utilized by the overall (mobile and cloud) system. For example, if the total number of threads is equal to 1, this implies that at a particular point of time, a total of 1 thread is executed (either locally at the mobile device or on the cloud). For more than 1 threads, since the mobile device has a single processor, the rest of the threads must execute on the cloud, if any parallelism is utilized. Once again, we do not consider the energy consumption. This is because according to our energy model, the energy consumption does not depend on the amount of parallelism used.

Fig. 6.10 shows how increasing the number of threads affects the execution time. Increasing the number of threads from 1 to 2 leads to an improvement of 45% in execution time. However, increasing the number of threads from 2 to 3 only leads to an improvement of 2% in execution time. Further increase in the number of threads leads to no improvement.

These observations can be explained by noting that our example graph has three parallel threads. Hence increasing the number of threads to greater than three has no effect on performance. Moreover, the third thread has a native method which must be executed locally on the mobile device. Thus, offloading this thread may or may not lead to any improvement in execution time. Hence the average gain observed when increasing the number of threads from 2 to 3 is small. However, since two of the threads do not contain any native methods, increasing the number of threads from 1 to 2 leads to a large improvement in execution time.

An alternative way of exploiting parallelism is to increase the number of processors in the mobile device itself. Once again, we study the increase in execution time when the number of mobile processors is increased. The result of our simulation is shown in Fig. 6.11. Our simulation result shows that this has no effect on the execution time. We explain this by noting that executing a thread on the cloud is usually faster as compared to local execution. Thus, even if a mobile processor is idle, the offloading framework chooses to offload methods of a thread instead of scheduling it on the idle processor for local execution. Hence, increasing the number of processors on the mobile device shows no improvement in execution time.

6.5 Conclusion

Mobile cloud computing presents a solution to augment resource constrained mobile devices, where computationally intensive tasks can be partially offloaded to the cloud servers. Execution offloading to cloud helps in conserving computation energy on the mobile device, but consumes network energy to communicate with the cloud. Hence offloading decision must carefully select tasks to offload to eventually save energy on the mobile device. The task of offloading becomes more challenging due to the practical operating environment where there are multiple variable parameters. The effects of these parameters must be considered while making the offloading decisions.

In this work, we studied the impact of various parameters present in MCC systems on energy consumption and execution time of mobile applications. We presented a formal model of the offloading decision problem that incorporates various parameters that appear in real MCC execution environments. We utilize this model to study the impact of these parameters on the performance optimization objectives, like energy saved, and reduction in application execution time.

Chapter 7

Computation Offloading from Mobile Devices: Can Edge Devices Perform Better Than the Cloud?

7.1 Introduction

In the previous chapter, we looked at various application and network parameters that affect the performance of offloading. An interesting question in computation offloading is to decide which machines to use as server. Most offloading framework prototypes developed so far use an in-house desktop or server machine. Currently, offloading frameworks have two distinct choices of machines that can be used as server. The first choice involves offloading to commercially available cloud servers and is known as mobile cloud computing. Prototype implementations of offloading utilize this technique. The second choice, known as mobile edge computing, involves offloading to other user edge devices such as tablets, laptops or network routers.

Cloud servers and user edge devices have two major differences:

- Processors of cloud servers have faster processors. For example, running the benchmark CoreMark [37] shows that a Google Cloud Platform [47] processor is around 6 times faster than a mobile device.
- Latency of cloud servers is higher than edge devices. By performing a

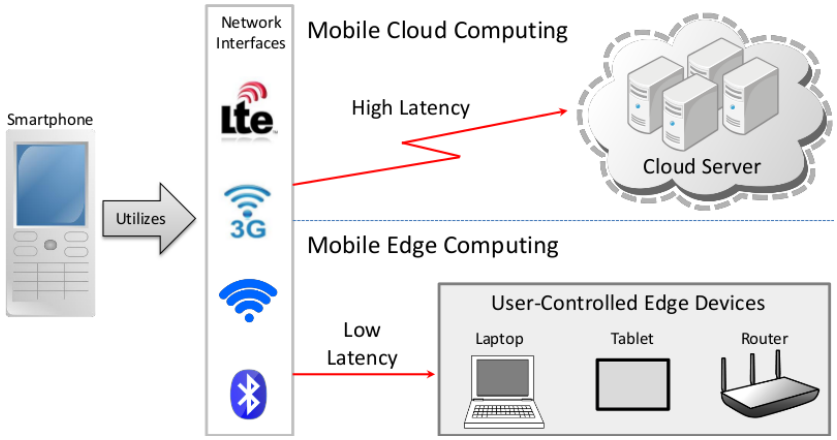


Figure 7.1: Working of a computation offloading framework. The offloading framework may use either low-powered user edge devices, such as routers, laptops and tablets, or commercially available cloud servers. These two cases are known as mobile edge and mobile cloud computing respectively.

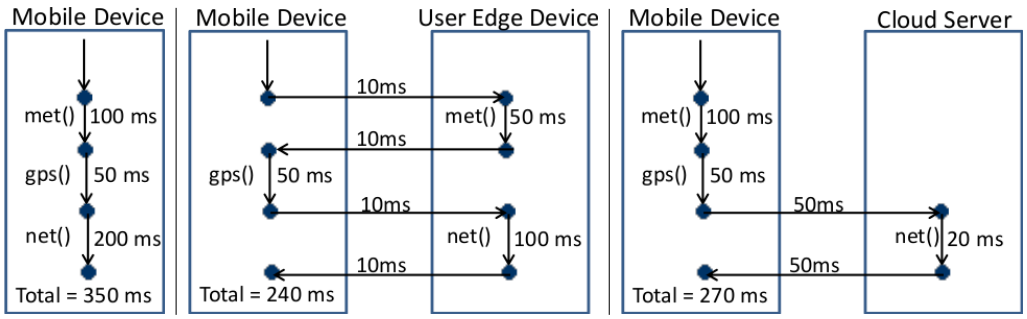
series of ping probes from our mobile device over Wi-Fi, we found that Google Cloud Platform has an average latency of 87 ms, compared to just 14 ms for a device within the same network.

One of the most important factors of user satisfaction is lower application finish time [54]. An offloading framework partitions the mobile application in a way that reduces its finish time. While execution on a cloud is much faster than on a mobile device, migrating data over the network between the mobile device and the server consumes additional time. Thus, a partitioning algorithm has to balance the trade-off between more execution of tasks on the server and ensuring less time spent on migrating data. The speed of the server and the network latency, therefore, has a major impact on the way the offloading framework partitions the application.

Fig. 7.2 shows how the partition generated for a simple program changes due to the type of server used. The application consists of three methods – met(), gps() and net() respectively. The method gps() depends on the GPS, and thus must be executed on the mobile device. When the application is executed entirely on the mobile device, execution of the application takes 350 ms. When the offloading framework can offload to an edge device, both the methods met() and net() are offloaded. This takes a total of 240 ms. However, when it has access to a cloud server, the higher latency ensures

	Latency	Processor Power
Edge Device	10 ms	2 times
Cloud Server	50 ms	10 times

(a) Parameters used for the example. Latency refers to the latency of server from mobile device. Processor power refers to speed of processor compared to mobile device.



(b) (c) (d)

Figure 7.2: An example of how server device affects application partitioning. Parameters used for this example are shown in Fig. 7.2a. Fig. 7.2b shows execution of the application entirely on mobile device. Fig. 7.2c and Fig. 7.2d show execution by offloading to an edge device and cloud server respectively.

that executing met() on the mobile device is faster. Thus, only net() is offloaded. Using a cloud server, therefore, gives an application finish time of 270 ms. In this way, the application partition changes depending on the type of server used.

A key question here is if offloading to user edge devices can provide faster application finish time compared to commercial cloud servers. Edge devices have weaker processors than cloud servers, but also have lower latency. Whether the lower latency of edge devices can compensate for the weaker processors compared to cloud servers needs to be investigated. Such an investigation needs to study the performance of offloading using different cloud servers under realistic workloads.

In this work, we compare the performance of offloading using cloud server and to user edge devices. We study the impact of using different servers on application finish time and application partition. We first develop a system model that can be used to study both cases of offloading. We collect

traces of SPECjvm2008 [108] programs using aspect-oriented programming. Aspect-oriented programming allows us to modify the bytecode of an existing application at run-time to log details of methods executed. We then perform trace-driven simulation to determine the application finish time using cloud server and edge devices of SPECjvm2008 benchmark programs. We find from our trace-driven simulation that edge devices for general-purpose computing, such as laptops, can perform better than cloud servers. Smaller edge devices, like tablets and routers, can also reduce application finish time, but gives slower performance than cloud servers. Our work, therefore, shows that offloading to edge devices is an attractive option for smartphone users.

The rest of this chapter is organized as follows. Section 7.2 describes related work. We develop our formal model of offloading in Section 7.3. Section 7.4 describes our techniques to generate traces of the workload and measure the parameters required for simulation. We discuss our simulation results in Section 7.5, and some of the limitations of our approach in Section 7.6. We conclude in Section 7.7.

7.2 Related Work

In this section, we first start with discussion of related offloading frameworks. We then explain studies that target managing latency of cloud servers. Finally, we discuss related works on offloading to edge devices.

The first computation offloading frameworks from mobile devices, MAUI [31], CloneCloud [29] and Odessa [96], used a single desktop or server machine as remote server. These systems usually used a software-based middleware to vary the network latency to simulate the latency of cloud servers. Other offloading frameworks, such as ThinkAir used a custom-made server with many different virtual machine (VM) configurations [70]. Barbera, Kosta, Mei, and Stefa [4] performed a trace-based study of energy gains using a commercial cloud service Amazon EC2. Another study used PlanetLab servers to study the effect of latency on interactive smartphone applications such as games [117]. These studies first identified high latency of cloud servers as a major problem in computation offloading.

A second category of studies on offloading suggested installation of computing resources with ready access to energy in the vicinity of mobile devices [101]. Such computing resources are called cloudlets. Since cloudlets are closer to mobile devices, they have much lower latency. However, utilizing cloudlets

require installation of additional computing infrastructure. This has slowed their adoption.

Another group of studies aim to reduce the latency of existing cloud data centers. For example, QJUMP suggests using separate queues for latency-sensitive applications that utilize the cloud server [52]. Silo provides guarantees of network latency by utilizing network calculus [60]. Finally, a recent proposal suggests inferring the latency requirements of an application by studying its request patterns [82],

Finally, a few recent studies have suggested utilization of edge devices in the context of Internet of Things (IoT). This is known as fog computing [11]. For example, mobile fog suggested utilizing of distributed network devices such as routers closer to the mobile devices [57]. Garcia Lopez, Montresor, Epema, Datta, Higashino, Iamnitchi, Barcellos, Felber, and Riviere [42] proposed a more general type of offloading, where application is offloaded to different user devices. This is known as mobile edge computing. Our study builds on these works by studying the feasibility of utilizing edge devices using trace-driven simulation.

7.3 Task Partitioning Model

In this section, we first explain the way computation offloading works. We then utilize this technique to develop its formal mathematical model.

7.3.1 Preliminaries

A computation offloading system consists of several processors p_k both in the mobile device and server. We represent the set of processors as $\mathbb{P} = \{p_1, p_2, ..., p_m\}$. A subset of these processors $\mathbb{M} \subset \mathbb{P}$ belong to the mobile device.

A mobile application is represented using a Directed Acyclic Graph (DAG) $G = (\mathbb{V}, \mathbb{E})$. A vertex v_j in the vertex set $\mathbb{V}$ represents a method or a task in the application. On a processor p_k a task v_j takes t_j^k time to execute. The value of t_j^k depends on the task v_j and the power of the processor p_k. An application begins and ends on the mobile device. Moreover, the execution of a set of tasks $\mathbb{U}$ may be tied to some hardware such as camera or GPS present only on the mobile device. Dependency from a task v_i to v_j is represented as an edge (v_i, v_j). The set of dependencies is the edge set $\mathbb{E}$ of the DAG. For

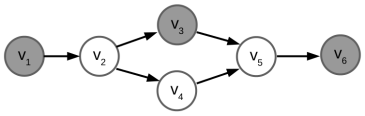


Figure 7.3: An example of a call graph representing a mobile application. Methods shaded gray must be executed on the mobile device.

$\mathbb{V}$	Task set of application
$\mathbb{E}$	Dependency set of application
$\mathbb{U}$	Set of tasks that must be executed on mobile device
$\mathbb{M}$	Set of processors in mobile device
$\mathbb{P}$	Set of processors in mobile device and server system
v_j	A method in the call graph
p_k	A processor in the system (mobile device or servers)
t_j^k	Execution time of a task v_j on processor p_k
m	Number of processors in the system
n	Number of tasks in the mobile application
(v_i, v_j)	Data dependency from task v_i to v_j
r_{ij}^{hk}	Migration time of dependency (v_i, v_j) from p_h to p_k
x_j^k	Decision variable denoting if v_j is executed on processor p_k
T_j	Finish time of the task v_j
S_j	Start time of the task v_j
R_{ij}	Migration time of the edge (v_i, v_j)
σ_{ij}	Decision variable denoting if v_i is executed before v_j

Table 7.1: Symbols and variables introduced in Section 7.3

each dependency (v_i, v_j), execution of task v_j can begin only when v_i finishes. Moreover, if v_i and v_j execute on different machines, then program states must be migrated. Let r_{ij}^{hk} denote the time to migrate data from processor p_h to p_k for (v_i, v_j). The value of r_{ij}^{hk} depend on the location of the processors p_h and p_k as well as on the amount of data that needs to be migrated for (v_i, v_j). We assume that execution time t_j^k and migration time r_{ij}^{hk} are known a priori by profiling the application. We note that prior profiling to get execution and migration time is common in offloading systems.

7.3.2 Mathematical Model

The offloading framework needs to decide the processor p_k on which each task v_j executes. To denote this, let x_j^k be a binary decision variable such that:

$$x_j^k = \begin{cases} 1 & \text{if task } v_j \text{ is executed on processor } p_k \\ 0 & \text{if task } v_j \text{ is not executed on processor } p_k \end{cases}$$

Let the start time and execution duration of a task v_j be S_j and l_j respectively. Also, let the completion time of v_j be T_j. Our aim is to reduce the finish time of mobile application. This is equal to the finish time of the last task v_n of the application. Thus, our objective is to minimize the finish time T_n of the last task v_n:

$$\text{Min } T_n \tag{7.1}$$

The finish time T_j of a task v_j is equal to the sum of its start time S_j and its execution time t_j^k on the processor p_k. Mathematically,

$$\forall v_j \in \mathbb{V}, T_j = S_j + \sum_{j=1}^{m} x_j^k t_j^k \tag{7.2}$$

A task v_j can start executing only if its predecessor tasks v_i become available. A predecessor task v_i becomes available, when v_i finishes execution and its data is migrated to the processor where v_j is executed. Thus, starting time S_j of v_j is not less than the sum of finish time T_i of v_i and migration time R_{ij}.

$$\forall (v_i, v_j) \in \mathbb{E}, S_j \geq T_i + R_{ij} \tag{7.3}$$

The migration time R_{ij} of an edge (v_i, v_j) is the time needed to fetch output of v_i to execute v_j. If an edge (v_i, v_j) is migrated from processor p_h to p_k, this has a cost of r_{ij}^{hk}. Here we assume that if p_h and p_k are same processors, then $r_{ij}^{hk} = 0$. We represent the migration cost mathematically as:

$$\forall (v_i, v_j) \in \mathbb{E}, R_{ij} = \sum_{(v_i, v_j) \in \mathbb{E}} \sum_{h=1}^{m} \sum_{k=1}^{m} x_{ij}^h x_{ij}^k r_{ij}^{hk} \tag{7.4}$$

The constraints that we have defined so far do not limit the amount of parallelism. However, the number of processors available is limited. Thus, the offloading framework also needs to decide the sequence of execution of

tasks. To denote this, let σ_{ij} be a binary variable for all pair of tasks t_i and t_j such that:

$$\sigma_{ij} = \begin{cases} 1 & \text{if } v_i \text{ finishes execution before } v_j \text{ begins execution} \\ 0 & \text{otherwise} \end{cases}$$

Thus, we can now rewrite Equation 7.3 as:

$$\forall v_i, v_j \in \mathbb{V}, S_j \geq \sigma_{ij}(T_i + R_{ij}) \tag{7.5}$$

For all edges (v_i, v_j) in the graph, the task v_j has to be executed only after v_i has completed. This precedence constraint is represented using the variable σ_{ij}.

$$\forall (v_i, v_j) \in \mathbb{E},\ \sigma_{ij} = 1 \tag{7.6}$$

Moreover, for any pair of tasks v_i or v_j, either v_i must be executed before v_j or vice-versa. Mathematically, we represent this as:

$$\forall v_i, v_j \in \mathbb{V},\ \sigma_{ij} + \sigma_{ji} \leq 1 \tag{7.7}$$

If the tasks v_i and v_j are scheduled by the offloading framework concurrently, i.e. $\sigma_{ij} = \sigma_{ji} = 0$, then they must execute on different processors. Thus, in this case, only one of the values among x_i^k and x_j^k can be equal to 1. Mathematically, we represent this constraint as:

$$\forall v_i, v_j \in \mathbb{V}, \forall k = 1...m, x_i^k + x_j^k \leq 1 + \sigma_{ij} + \sigma_{ji} \tag{7.8}$$

Finally, the tasks $v_j \in \mathbb{U}$ can only be executed on the mobile device. In other words, they can be executed on any one of the processors $p_k \in \mathbb{M}$. Mathematically, we represent this as:

$$\forall v_j \in \mathbb{U}, \sum_{p_k \in \mathbb{M}} x_j^k = 1 \tag{7.9}$$

All other tasks $v_j \in \mathbb{V} - \mathbb{U}$, must be executed on any one processor from any device, i.e.

$$\forall v_j \in \mathbb{V} - \mathbb{U}, \sum_{p_k \in \mathbb{P}} x_j^k = 1 \tag{7.10}$$

Equations 6.11 to 7.10 provide a formulation of an offloading system with multiple processors on different devices. An optimization solver on solving this simulation system gives the values of x_i^k to denote the processors on which each task is executed, and σ_{ij} to denote the execution sequence of each task. We now utilize this formal model to develop our simulation system.

7.4 Methodology

In this section, we first describe our method of collecting traces of applications. We then explain our technique of measuring different parameters required for simulation.

7.4.1 Generation of Call Graph

We use aspect-oriented programming to generate an annotated call graph. Aspect-oriented programming (AOP) is a technique of adding additional code to an existing program, without directly modifying its source code. The additional code is called aspect. AOP can even work in cases where source code is not available by modifying intermediate code of the application.

We utilize AspectJ, which is a common framework for aspect-oriented programming in Java [68]. AspectJ can add additional code at run-time to modify the behavior of an existing program. We treat the entry and exit points of each method as possible migration points. For our purpose, we log details of each method at their possible migration points.

To form a call graph from a program that is useful for our purpose, we collect the following data for each method:

- Method name, including its formal parameters and return types
- Thread identifier
- Execution time
- Amount of data that needs to be migrated

We obtain the method name by accessing the stack trace of the current thread. Similarly, Java provides a method within Thread class to access the thread identifier of the current thread. To obtain the execution time, we utilize Java's ThreadMXBean[1] interface. Finally, to obtain the amount of data, we serialize the objects of each argument and return types, write it to a memory buffer and then calculate its length. We then use a java agent at run-time to obtain these data from the benchmark programs. We use the time command on the benchmark programs to find out the overhead of utilizing aspect-oriented programming. Our measurements showed an overhead of less than 10% on the execution time of benchmark programs.

[1] http://docs.oracle.com/javase/7/docs/api

7.4.2 Estimation of Simulation Parameters

A realistic estimate of performance using both cloud and edge devices requires measurement of their processor performance and network latency. To compare the performance of processors with different instruction set architectures, we use CoreMark benchmarks [37]. CoreMark is a set of common benchmark programs, containing matrix multiplication, linked-list manipulation and Cyclic Redundancy Check. One of its major advantages is that it is widely available for execution on different platforms, including desktops, servers and mobile devices. It is a widely used technique of comparing processor performance across platforms. The values obtained using these benchmarks is taken as a representative of the overall processor performance. Table 7.2 shows a list of platforms on which we perform our experiments.

Device	Model	Processor	CoreMark value
Smartphone	Samsung Galaxy S3 [100]	Quad-core 1.4 GHz Cortex-A9	1786
Tablet	Google Nexus 10 [86]	Dual-core 1.7 GHz Cortex-A15	3850
Router	ASUS onHub [89]	Qualcomm IPQ8064	3500 [2]
Laptop	Sony VAIO Notebook[107]	AMD Dual-Core E-350 (1.6 GHz)	4960
Cloud	Google Cloud Platform[47]	n1-standard-8	10906

Table 7.2: A list of Coremark values per core in different hardware devices. Coremark values per core are taken as representative of the processing speed of a single core.

We measure the network latency of user devices and cloud servers by using ping probes. We use the ping utility to send 100 ping probes and then take their average latency values. This is a standard technique widely used in measuring the latency values. Our experiments showed an average latency of 14 ms for user-controlled edge devices and 87 ms for our cloud server.

7.5 Trace-driven Simulation

We selected a set of SPECJVM08 benchmarks that are relevant to mobile devices. The benchmarks we selected include common workload such as encryption, data compression, Fast Fourier Transform and audio decoder. We generate the traces of these benchmark programs using the technique described in Section 7.3. We then implemented the mathematical model

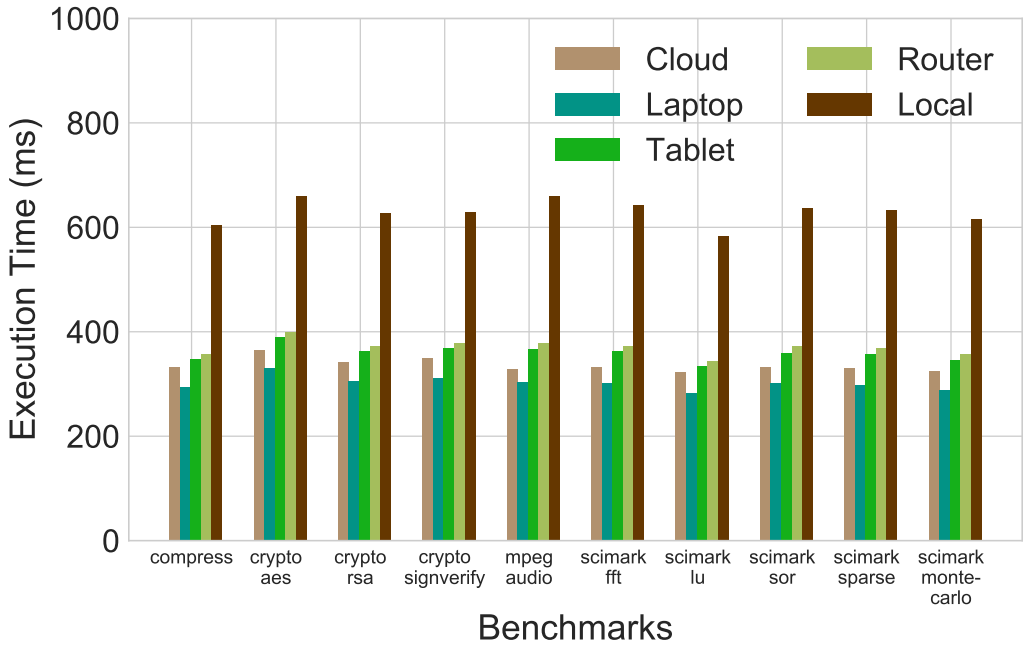


Figure 7.4: A comparison of finish time of different benchmark applications by offloading to cloud server, laptop, tablet, router and without utilizing offloading.

described in Section 7.3 as an Integer Linear Programming (ILP) problem in Matlab.

Fig. 7.4 shows the application finish time of benchmark applications using both edge devices and cloud server. We note that all server systems (edge and cloud) improve application finish time. Moreover, the best performance for each benchmark is obtained using a laptop, followed by cloud server, tablet and router respectively. A cloud server reduces the average application finish time by 46% over local execution, compared to 52% for laptop, 43% for tablet and 41% for router.

Our trace-driven simulation, therefore, shows that user-edge devices can perform better than a cloud server if they have sufficiently powerful processors. The slower processors of user-edge devices is compensated by the lower latency to access these devices. Even a smaller edge-device like router improves the performance of the benchmark applications significantly.

[2]The coremark value of this device is an estimate based on the values of similar

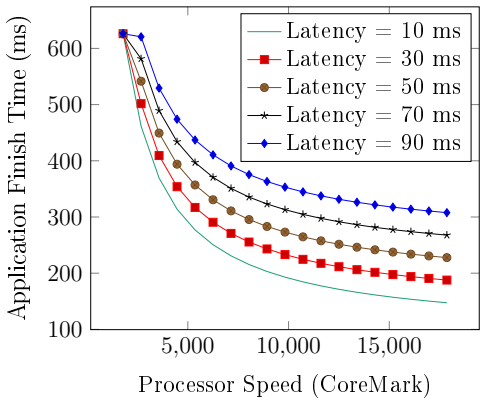


Figure 7.5: Impact of processor speed and latency on application finish time. Processor speed is measured using the CoreMark benchmark value.

To further study the effect of processor performance and network latency, we vary the latency and processor power of the server in our simulation. We then study the average application time across the ten benchmark programs.

Fig. 7.5 shows the impact of processor speed and network latency on application finish time. We note that at a latency of around $90ms$, improving the processor speed does not have much impact on the finish time. At lower latencies, the impact of more powerful processors is much greater. In each case, we observe that the gains of increasing the processor speed diminish after reaching a value of around 8000 Coremarks.

These observations show that the most significant factor limiting performance of mobile cloud systems is high network latency. Since cloud systems are accessed over the Internet, the cloud server provider has only a limited role in reducing latency. On the other hand, user-controlled edge devices, despite having slower processors, have much lower latency. Moreover, due to improvement in processor technology, the processor speeds of such devices is continuously improving. Thus, offloading to user-controlled edge devices is likely to become more attractive for smartphone users in the near future.

processors.

7.6 Discussion

Our trace-driven simulation makes two assumptions. We do not discuss the consider the execution of other processes on edge devices. Execution of other processes lead to time-sharing of processors and increase the response time of smartphone requests. However, our simulation results show that even slower edge devices significantly reduce application finish time. Secondly, we use CoreMark benchmark as a measure of the processor performance. Although CoreMark is a widely used processor benchmark, a proper study of processor speed requires running a variety of workloads. Our trace-driven simulation experiments assume that the processor performance remains approximately similar for different applications.

7.7 Conclusion

In this chapter, we compare the performance of offloading from smartphone to a cloud server and user-controlled edge devices such as laptops, tablets and routers. We first formulate a mathematical model to represent the offloading problem. We then utilize aspect-oriented programming to obtain traces of benchmark Java programs. We perform trace-driven simulation to determine whether offloading to edge devices can reduce application execution time. Our simulation shows that offloading to larger edge devices such as laptops can provide better performance than a cloud server. Smaller edge devices such as tablets or routers provides slower performance than a cloud server, but can also significantly speed up application execution. Thus, offloading to such devices is a promising technique of augmenting the processor resources of smartphones.

As future work, we would like to study the impact of offloading to user-controlled edge devices on energy consumption. Smartphones can utilize bluetooth to connect to user devices, which consumes much lower energy. We would like to explore the impact of utilizing bluetooth for offloading smartphone applications.

Chapter 8

Service Level Guarantee for Mobile Application Offloading in Presence of Wireless Channel Errors

8.1 Introduction

Mobile computing platforms, from smart sensors to smartphones, are increasingly used in personal and enterprise environments. However, these devices have limited compute capacity. This limitation can be mitigated by offloading parts of a mobile application to execute on cloud servers, thereby reducing application finish time. A number of proposals [31, 29] for mobile cloud computing have received prominence in literature. Among other factors, offloading decisions depend on network conditions. Since network conditions in mobile environment vary widely, offloading decisions based on profiled network parameters can lead to sub-optimal solutions.

Channel error rates are one of the hardest to model among network parameters. Channel error is dependent on unpredictable external interference, and mobility characteristics, like walking or driving. Measurement based studies have shown channel error rates up to 30% under different conditions [53, 50]. Therefore, offloading solutions depend on the MAC layer retransmission mechanism to handle channel errors. Since the number of retransmissions can depend on transient channel error states, this can undermine the benefit of

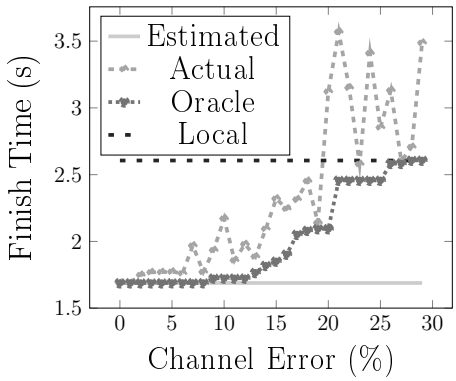


Figure 8.1: Execution time comparison under varying channel errors.

offloading in saving energy and/or finish time.

We illustrate with an example. We take a task graph with 100 tasks, where a task, representing a method in the application execution, can be offloaded to the remote cloud server for faster execution. Given each task's workload profile, and network parameters, an optimization solver (as in MAUI [31]) computes the *estimated* time to finish execution. Fig. 8.1 shows a comparison of application finish time using four schemes: *Estimated* is the result of using an optimization solver with application and network profile as input, *local* computes without any offloading, *actual* is the result of offloading in practice due to channel errors, and an hypothetical solution (called oracle in the figure) assumes complete knowledge of channel errors. Compared to *local* execution, where no task is offloaded, an offloading scheme performs better. However, in practice, the channel error conditions can break the assumption about network parameters. In presence of varying channel errors, the *actual* result of offloading may not be as computed by an optimization solver. An *oracle* solution, with complete knowledge of channel errors, can indeed perform better.

In this work, we pose the question, *even in presence of unpredictable channel errors, can we ensure service level guarantee to complete the application execution faster than that of local execution on the device?* We show that, given a failure rate bound, the question can be modeled as a chance constrained optimization problem [38]. We propose an error-aware run-time adaptive heuristic that decides at each task offload point, the locally optimal choice considering stochastic channel errors. We provide guarantee to minimize the expected application finish time. Our scheme ensures that an application

completes execution faster than local execution, in presence of retransmissions due to channel errors. We validate our solution using simulations and on traces of benchmark applications.

We present relevant prior work in Section 8.2. Section 8.3 and Section 8.4 present the analytical model and the proposed heuristic. Evaluation is presented in Section 8.5 followed by conclusion in Section 8.6.

8.2 Related Work

There are two different categories of work in the context of offloading over wireless channels. One group of work assumes that the Medium Access Control (MAC) layer handles channel errors successfully. The first offloading frameworks, MAUI [31] and CloneCloud [29], used this approach. They estimated the channel bandwidth before solving the offloading decision problem. Another offloading framework, ThinkAir [70] looks at history of migration and assumes that the channel conditions remain similar to the past observations. Some other works try to reduce the amount of data migration. [121] proposes compiler-level optimizations to decide which data is actually used by the cloud server. These offloading frameworks do not consider the cost of transmission failure.

Finally, a few studies have considered the effect of channel errors. [73] shows how intelligent checkpointing of applications to ensure consistency on the mobile device and the cloud server can save energy of offloaded applications. COSMOS [106] senses the response time to determine the quality of connection, and uses this observation for the offloading decision. However, they do not consider retransmission of lost packets. In [127], the authors consider retransmission, but the decision about the number of retransmissions is not made at run-time. In Foreseer [120], the initial partition obtained by running an optimization solver is modified at run-time based on the channel bandwidth. In contrast to the work above, our proposal models the number of retransmissions due to channel errors and presents an adaptive offloading algorithm design.

$\mathbb{V}$	Vertex set of the graph
$\mathbb{E}$	Edge set of the graph
v_j	A task in the application execution
v_m	Last task in the graph
(v_i, v_j)	A dependency from the task v_i to v_j
$\mathcal{M}_0$	Mobile device
$\mathcal{M}_1$	Cloud server
t_j^l	Execution time of task v_j on machine $\mathcal{M}_l$
r	Time to migrate a single frame
U_m	Time deadline given to application
ϵ	Failure bound given to application
w_{ij}	Number of frames needed to migrate (v_i, v_j)
x_j	Variable indicating execution of v_j on $\mathcal{M}_0$ or $\mathcal{M}_1$
z_{ij}	Maximum number of retransmission attempts of (v_i, v_j)
Y_{ij}	Number of retransmission attempts of frames of (v_i, v_j)
R_{ij}	Total time to migrate (v_i, v_j)
T_j	Finish time of v_j
α_k	Failure bound on k^{th} migration
α_k^s	Failure bound on sending packet of k^{th} migration
α_k^r	Failure bound on receiving packet of k^{th} migration

Table 8.1: Symbols introduced in Section 8.3

8.3 Models and Problem Formulation

We represent execution of a mobile application as a directed acyclic graph (DAG) $G = (\mathbb{V}, \mathbb{E})$, where the vertex set $\mathbb{V}$ represents the set of m methods or tasks, and the edge set $\mathbb{E}$ represents the dependencies among tasks. A task can be executed either locally on the mobile device, $\mathcal{M}_0$, or on the remote cloud server, $\mathcal{M}_1$. However, the first and last task, v_1 and v_m respectively, must execute on the mobile device. If a task v_j is executed on a platform, $\mathcal{M}_0$ or $\mathcal{M}_1$, different from that of any of its predecessor tasks, v_i's, where $(v_i, v_j) \in \mathbb{E}$, then the task output states of v_i must be transferred over the network to v_j's execution platform. Since the data transfer size will vary across dependencies, therefore, the number of data fragments or frames at the MAC layer will also vary.

The wireless channel is modeled as a stochastic process [95], where the

probability of successful transmission of a frame is denoted by p. The value of p depends on the time varying nature of the channel. However, we assume that for a single data packet (i.e. for all the corresponding frames) the channel state remains unchanged. Due to channel errors, if a frame is lost, it is retransmitted. Let Y_{ij} be the total transmission attempts for all the frames of a packet transferring data from v_i to v_j. If the time to transmit a frame is r, then the time, R_{ij}, for the packet transmission will therefore be,

$$R_{ij} = rY_{ij}$$

Since Y_{ij} depends on the channel conditions, both Y_{ij} and R_{ij} are stochastic parameters.

The total time to execute an application depends on where each task is executed (i.e. execution time) and the time for the network transfer (i.e. migration time). Note that time to execute an application is same as the finish time, T_m, of the last task, v_m. T_m depends on the time for network transfers (R_{ij}'s), and is therefore also a stochastic parameter. Let U_j denote the time taken to finish v_j if v_j and all tasks preceding it are executed locally. Our objective is to minimize the expected finish time, T_m, under a constraint that T_m exceeds the local execution time, U_m only with a fixed probability ϵ. The constraint guarantees a service level agreement (SLA) that the application finish time will exceed local execution time (U_m) with maximum probability ϵ while offloading to cloud in unpredictable channel conditions. We express this as a chance constrained optimization problem:

$$\begin{aligned} &\textbf{Min } E[T_m] \\ &\textbf{subject to: } \mathbb{P}(T_m > U_m) \leq \epsilon \end{aligned} \tag{8.1}$$

We now explain the nature of this optimization problem. Since there are some tasks in the DAG that must be executed on $\mathcal{M}_0$, there may be multiple send and receive migrations to the cloud server. We consider these migrations in pairs. A send migration offloads the data needed by an offloaded method from $\mathcal{M}_0$ to $\mathcal{M}_1$, while a receive migration sends data back from $\mathcal{M}_1$ to $\mathcal{M}_0$. Corresponding to every send migration of a method, we can therefore uniquely associate a receive migration of another method before the next send migration is initiated. We leverage on this pairwise send-receive association to build the foundation of our theory. In our work, for the sake of simplicity, we use a migration to denote a send-receive association pair. Then, we define the event of failure of a single migration as "execution time greater than local

execution time". We denote the failure for k^{th} migration attempt as F_k, i.e. F_k is true if $(T_j > U_j)$ where v_j is executed on mobile device. Since the condition of the channel may change between migrations, it is possible that after a single migration is completed, the channel condition degrades to allow no further migrations. Thus, failure of a single migration may lead to failure of the entire execution. We therefore, rewrite the chance constraint as:

$$\mathbb{P}(\bigcup_k F_k) \leq \epsilon, \tag{8.2}$$

where k varies over the number of migrations during the application's execution from start to finish. Using inclusion-exclusion principle [110], we rewrite this as:

$$\sum_k \mathbb{P}(F_k) \leq \epsilon \tag{8.3}$$

A conservative way to satisfy Eqn 8.3 is by imposing a failure bound α_k on each migration:

$$\mathbb{P}(F_k) \leq \alpha_k, \quad \forall k \text{ such that: } \sum_k \alpha_k \leq \epsilon \tag{8.4}$$

As before, a single migration consists of two different probabilistic events: sending a packet to cloud and receiving it back to mobile device. Then, the total time available for migration to satisfy deadline may be divided up into three components: sending a packet, executing tasks on cloud and receiving a packet. Since only sending and receiving are probabilistic events, we define F_k^s and F_k^r as failure while sending and receiving respectively. Here, F_k^s and F_k^r are defined as events denoting failure to send and receive a packet within an assigned time (to be detailed in the following) that guarantees SLA satisfaction. As in Eqn 8.4, we bound the probability of failure while sending and receiving by α_k^s and α_k^r respectively:

$$\mathbb{P}(F_k^s) \leq \alpha_k^s \text{ and } \mathbb{P}(F_k^r) \leq \alpha_k^r \text{ such that: } \alpha_k^s + \alpha_k^r = \alpha_k \tag{8.5}$$

We need α_k^s and α_k^r that minimizes the overall application finish time. We now establish a bound on the number of transmission attempts for each individual send or receive migration. Let z_{ij} be the maximum number of transmission attempts for a send or receive migration between v_i and v_j. The values of α_k^s and α_k^r determine the value of z_{ij}. We assume a single

packet of (v_i, v_j) data contains w_{ij} frames. Thus, if migration (either send or receive) is performed, the actual number of transmission attempts Y_{ij} must satisfy:

$$w_{ij} \leq Y_{ij} \leq z_{ij} \tag{8.6}$$

We need to find values of z_{ij} that minimize the overall execution time while satisfying to satisfy SLA. Increasing z_{ij} reduces the failure rate. However, this also increases the expected application finish time.

8.4 Solution Approach

In this section, we design a heuristic that minimizes application finish time. We denote z^s_{ij} and z^r_{ij} as the maximum number of transmission attempts for send and receive migrations respectively. This requires allowing a maximum z^s_{ij} and z^r_{ij} transmission attempts while sending and receiving packets from cloud server.

We explain our methodology on z^s_{ij}. The computation of z^r_{ij} is similar. Sending a (v_i, v_j) packet succeeds only if all of its w_{ij} frames are successfully transmitted. Let Q_{ij} be a random variable denoting the number of frames successfully transmitted in a total of z^s_{ij} transmission attempts. Then, failure to send a dependency to the cloud server (F^s_k) occurs when less than w_{ij} frames are transmitted successfully in z^s_{ij} transmission attempts. We, therefore, rewrite Eqn 8.5 as follows:

$$\mathbb{P}(Q_{ij} < w_{ij}) \leq \alpha^s_k \tag{8.7}$$

As discussed before in our channel model, the probability p of successful transmission remains same while sending frames of a single packet. Thus, we can treat Q_{ij} as a binomial random variable with the parameters z^s_{ij} and p, i.e. $Q_{ij} \sim Binomial(z^s_{ij}, p)$. There is no closed form formula to find the probability of success of at least w_{ij} trials in z^s_{ij} attempts [49]. We, therefore, find an approximate value of z^s_{ij} using Hoeffding's inequality [56]. Hoeffding's inequality states that for a random variable, $Q_{ij} \sim Binomial(z^s_{ij}, p)$, the deviation from the mean t (where $t < 0$) is bounded by:

$$\mathbb{P}(Q_{ij} - E[Q_{ij}] \leq t) \leq \exp\{-2t^2/z^s_{ij}\} \tag{8.8}$$

Algorithm 4 Our channel error offloading algorithm.

procedure EXECUTE-APPLICATION($\mathbb{V}$, $\mathbb{E}$, U_m, ϵ, r)
  x[1] $\leftarrow 0$
  $k \leftarrow 1$
  Execute first task v_1 on mobile device
  for all $v_j \in \mathbb{V}$ ready for execution **do**
    Get the probability of successful transmission p
    $\alpha_k = \epsilon/2^k$
    CALCULATE-BUDGET($\mathbb{V}$, $\mathbb{E}$, U_m, p, α_k, r)
    Y $\leftarrow 0$
    for all $(v_i, v_j) \in \mathbb{E}$ **do**
      Calculate number of frames w_{ij} for migration
      $z_{ij} = \lceil \frac{p(w_{ij}-1)(4+\sqrt{2})-\ln(\alpha_k/2)}{2p^2} \rceil$
      if x[i] = 0 & mobBudget[j] > cldBudget[j] $+z_{ij}r$ **then**
        migTime $\leftarrow T_i$ - cldBudget[j]
        x[j] $\leftarrow 1$
        f $\leftarrow 1$
        while $f \leq w_{ij}$ & x[j] = 1 **do**
          maxAttempts $\leftarrow$ migTime / rw_{ij}
          Attempt migration of f^{th} frame maxAttempt times
          if migration of frame failed **then**
            x[j] $\leftarrow 0$
          Store number of frame transmission attempts in Y_{ij}
          f $\leftarrow$ f + 1
      else if x[i] = 0 & mobBudget[j] $\leq$ cldBudget[j]$+z_{ij}r$ **then**
        x[j] $\leftarrow 0$
      else if x[i] = 1 & mobBudget[j] $+z_{ij}r >$ cldBudget[j] **then**
        Attempt transmission of frames till successful migration
        Store number of frame transmission attempts in Y_{ij}
        x[j] $\leftarrow 0$
        $k = k + 1$
      else if x[i] = 1 & mobBudget[j] $\leq$ cldBudget[j] $+z_{ij}r$ **then**
        x[j] $\leftarrow 1$
    $Y \leftarrow \max(Y, Y_{ij})$
    $h \leftarrow$ x[j]
    Execute v_j on $\mathcal{M}_h$
    $T_j \leftarrow T_i + rY + t_j^h$
procedure CALCULATE-BUDGET($\mathbb{V}$, $\mathbb{E}$, U_m, p, α_k, r)
  cldBudget[m] $\leftarrow \infty$
  mobBudget[m] $\leftarrow U_m - t_m^0$
  $C \leftarrow \{v_m\}$
  for all $v_j \in C$ **do**
    for all $(v_i, v_j) \in \mathbb{E}$ **do**
      Let w_{ij} be number of frames to migrate (v_i, v_j)
      $z_{ij} \leftarrow \lceil \frac{p(w_{ij}-1)(4+\sqrt{2})-\ln(\alpha_k/2)}{2p^2} \rceil$
      mobTime[j] $\leftarrow \max($mobBudget[j] - t_i^0,cldBudget[j] - t_i^1 - z_{ij}r)
      cldTime[j] $\leftarrow \max($cldBudget[j] - t_i^1, mobBudget[j] - t_i^0 - z_{ij}r)
      mobBudget[i] $\leftarrow \min($mobBudget[i], mobTime[j])
      cldBudget[i] $\leftarrow \min($cldBudget[i], cldTime[j])
    $C \leftarrow C \cup v_i$

We rewrite Eqn 8.7 as shown below to match Eqn 8.8.

$$\begin{aligned} \mathbb{P}(Q_{ij} < w_{ij}) &\leq \alpha_k^s \\ \implies \mathbb{P}(Q_{ij} - z_{ij}^s p < w_{ij} - z_{ij}^s p) &\leq \alpha_k^s \\ \implies \mathbb{P}(Q_{ij} - E[Q_{ij}] < w_{ij} - z_{ij}^s p) &\leq \alpha_k^s \\ \implies \mathbb{P}(Q_{ij} - E[Q_{ij}] \leq w_{ij} - z_{ij}^s p - 1) &\leq \alpha_k^s \\ \implies \exp\{\frac{-2(w_{ij} - z_{ij}^s p - 1)}{z_{ij}^s}\} &\leq \alpha_k^s \end{aligned} \tag{8.9}$$

Taking logarithm of both sides of Eqn 8.9, and solving for z_{ij}^s gives us the solution:

$$z_{ij}^s \geq \frac{4w_{ij}p - 4p - \ln(\alpha_k^s) + \sqrt{(4w_{ij}p - 4p - \ln(\alpha_k^s))^2 + 8p^2(w_{ij}-1)^2}}{4p^2} \tag{8.10}$$

z_{ij}^s represents the minimum number of send attempts needed to satisfy the SLA. Since increasing the number of transmission attempts also satisfy the SLA, we can utilize the inequality $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ in the above expression for z_{ij}^s to get a higher bound on z_{ij}^s as:

$$z_{ij}^s \geq \frac{8w_{ij}p - 8p - 2\ln(\alpha_k^s) + 2\sqrt{2}p(w_{ij}-1)}{4p^2} \tag{8.11}$$

Eqn 8.11 expresses the SLA constraint for sending (Eqn 8.5) in terms of number of transmission attempts z_{ij}^s. z_{ij}^s being an integer, we write z_{ij}^s as:

$$\begin{aligned} z_{ij}^s &= \lceil \frac{8w_{ij}p - 8p - 2\ln(\alpha_k^s) + 2\sqrt{2}p(w_{ij}-1)}{4p^2} \rceil \\ &= \lceil \frac{p(w_{ij}-1)(4+\sqrt{2}) - \ln(\alpha_k^s)}{2p^2} \rceil \end{aligned} \tag{8.12}$$

As discussed earlier, the k^{th} migration also involves receiving a packet of $(v_{i'}, v_{j'})$ from cloud server to mobile device. Solving the SLA constraint involves finding both z_{ij}^s and $z_{i'j'}^r$. Using the same method that we used for z_{ij}^s, we find the number of transmissions $z_{i'j'}^r$ to receive a packet:

$$z_{i'j'}^r = \lceil \frac{p(w_{i'j'}-1)(4+\sqrt{2}) - \ln(\alpha_k^r)}{2p^2} \rceil \tag{8.13}$$

So far, we have the values of z^s_{ij} and $z^r_{i'j'}$ in terms of weight parameters α^s_k and α^r_k respectively. We need to find values of α^s_k and α^r_k that minimize total time to send and receive packets, i.e. network time. We note that the time to send and receive a packet is equal to $z^s_{ij} \times r$ and $z^r_{i'j'} \times r$ respectively. Thus, total network time is given by $z^s_{ij} \times r + z^r_{i'j'} \times r$. We differentiate this with respect to α^s_k and set the derivative to 0 to obtain $\alpha^s_k = \alpha^r_k = \alpha_k/2$. Therefore, we replace α^s_k and α^r_k in the expressions of z^s_{ij} and $z^r_{i'j'}$ respectively by $\alpha_k/2$:

$$z^s_{ij} = \lceil \frac{p(w_{ij} - 1)(4 + \sqrt{2}) - \ln(\alpha_k/2)}{2p^2} \rceil \tag{8.14}$$

$$z^r_{i'j'} = \lceil \frac{p(w_{i'j'} - 1)(4 + \sqrt{2}) - \ln(\alpha_k/2)}{2p^2} \rceil \tag{8.15}$$

The above gives us the values of z^s_{ij} and $z^r_{i'j'}$ needed to satisfy SLA in terms of α_k for the different migrated edges.

We now need to assign values of α_k for each migration. The values of α_k must be assigned in a way that satisfies Eqn 8.5. Moreover, the total number of possible migrations are not known. A conservative strategy is to choose higher values of α_k for the early migrations, since saving time at the beginning increases the time available for later migrations. Thus, we choose α_k as a geometric distribution, with a ratio of 1/2 as shown below:

$$\alpha_k = \frac{\epsilon}{2^k} \tag{8.16}$$

Our heuristic now follows directly from this calculation. It takes as input the set of tasks $\mathbb{V}$, the set of tasks $\mathbb{E}$, the time deadline U_m, failure bound ϵ and time to transmit a single frame r. It then executes each task either on mobile device or cloud server. Whenever a task v_j is ready for execution on the mobile device ($\mathcal{M}_0$) or the cloud server ($\mathcal{M}_1$), we check whether executing it on the same machine or migrating it saves time. The time required for migration is obtained by sensing the channel condition at each step to find the probability p of successful transmission and using it to calculate the number of transmission attempts z^s_{ij} and $z^r_{i'j'}$. For simplicity, since z^s_{ij} and $z^r_{i'j'}$ have the same expressions, we refer to it as z_{ij} in our heuristic. If migration is faster, then a packet of (v_i, v_j) is migrated. While migrating, sending of a packet from mobile device to cloud server can be aborted before transmitting all frames if the number of failures is high. However, this is not possible for receiving a packet from cloud server to mobile device, since execution must

finish on mobile device. The exact algorithm is shown in detail in Algorithm 4.

We now analyze the time complexity of our method. The Procedure CALCULATE-BUDGET iterates over all dependencies in the application. Thus, it has a time complexity of $O(|\mathbb{E}|)$. Procedure EXECUTE-APPLICATION iterates over each task in the graph. For each task, it calls CALCULATE-BUDGET once. Thus, the total complexity of computing the overall budget is $O(|\mathbb{V}||\mathbb{E}|)$ It also has an inner loop that iterates over each dependency of a single task. Assuming the number of frames to be transmitted as constant, this has a time complexity of $O(|\mathbb{E}|)$. Therefore, total time complexity of using our algorithm is equal to $O(|\mathbb{V}||\mathbb{E}|)$. Assuming a constant number of parallel tasks, and since $|\mathbb{V}| = m$, the time complexity is equal to $O(m^2)$.

8.5 Evaluation

In this section, we evaluate the performance of our algorithm using simulation on both randomly generated graphs and benchmark programs.

8.5.1 Settings

We implement our heuristic at different channel error rates and failure bounds. To better understand the performance of our algorithm, we implement an Integer Linear Programming (ILP) based solution which assumes that there is no channel error. We also implement another ILP-based solution called oracle which knows in advance the cases in which transmission attempts fail. We have assumed in our simulation that the channel error rate varies around the mean with uniform distribution. The simulation parameters are given in Table 8.2.

8.5.2 Simulation Results

To study the performance of our heuristic, we first run the ILP-based solution, oracle and our heuristic on a set of 10000 randomly generated graphs. We then compare the failure rate, mean finish time and energy consumption of our heuristic with the ILP-based solution and the oracle.

Failure rate We compare the failure rates of the three implementations to check whether our algorithm satisfies the failure bound. Fig. 8.2 shows

Parameter	Range of Values
Migration time of each packet (r)	50 ms
Server speed compared to mobile device	5 times
Processor power	1 J/s
Network power	0.5 J/s
Number of random graphs	10000
Failure bound	1%
Channel error rate	30%

Table 8.2: Parameters used for each simulation experiment. Unless mentioned otherwise, these parameters are used in the experiments.

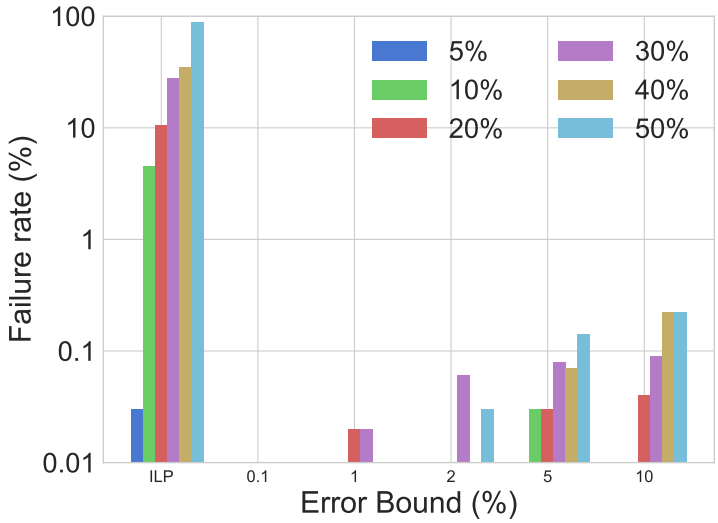


Figure 8.2: Comparison of failure rate at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). Failure represents a finish time higher than local execution.

the failure rates under different channel conditions compared to ILP based solution. We omit the oracle implementation since it knows in advance the cases of transmission failure and therefore, can never fail. We also do not show channel error rate of 2%, since the number of failures at 2% is too small. At channel error rates of 5%, 10% and 30%, the ILP gives a failure rate of 0.03%, 4.5% and 28.1% respectively. The failure rates for our solution are bounded within 10% even at 30% channel error, giving a service level guarantee of 90%. The number of failures in our scheme never exceeds the

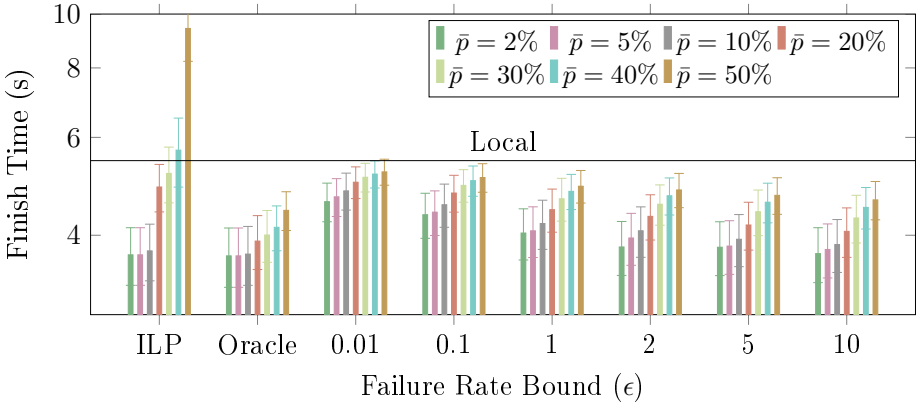


Figure 8.3: Comparison of finish time at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). Oracle solution represents best possible finish time for a given level of channel error.

defined failure bound ϵ.

These observations confirm that since ILP runs a priori, its solution might lead to worse than expected results while executing the application. Although our heuristic does not guarantee an optimal solution, it can sense the channel condition and decide accordingly whether to offload. This reduces the number of failures compared to an ILP. Moreover, when the number of errors in the wireless channel increases, our heuristic reduces the chances of failure by offloading tasks to the cloud. We confirm this observation by noting in Table 8.3 that the number of tasks executed on cloud server decreases with a decrease in failure bound (ϵ).

We also note that in a few cases the number of failures decreases with an increase in channel error. However, this decrease in failure at a higher channel error rate is less than 0.2%, which may be explained by the uncertain nature of the wireless network.

Finish Time We compare the finish times of our heuristic with the ILP-based and oracle solutions. Fig. 8.3 shows the mean finish time of the application samples under varying channel error rates. The heuristic has a better average performance than global optimization solver for channel error rate greater than 10%. When the channel error exceeds 20%, our heuristic takes less time than the ILP solution in all cases, with a failure rate of 10% giving a gain of 18%. Below 20% error, our heuristic provides a solution within 5% of the ILP solution for all values of ϵ. At error rate of 50%, the

ϵ \ $\bar{p}$	5	10	20	30	40	50
0.1	22	19	14	10	8	7
1	31	28	22	17	14	11
2	34	32	25	20	16	13
5	39	36	30	24	20	16
10	41	39	34	28	23	19

Table 8.3: Percentage of tasks executed on cloud server at different channel error rates ($\bar{p}$) and failure bounds (ϵ).

ILP takes twice the finish time of our heuristic.

We explain these observations by noting that an ILP obtains the best possible solution when there is no channel error. Thus at lower levels of channel error, it performs better, because channel error does not lower finish time significantly. When the number of channel errors increases, our heuristic performs better since it is able to adapt to the channel condition.

Energy Consumption We now investigate the effect of our heuristic on energy consumption of the battery in the mobile device. Since a mobile device runs on battery, reducing usage of battery energy is important for mobile users. We assume that execution on mobile device consumes power of 1 J/s, while network transmission takes 0.5 J/s. Fig. 8.4 compares the energy consumption of our heuristic with the ILP based solution. We note that energy consumption follows the same trend as finish time. This is because, the power consumption of processor system is greater than the network card. Thus, reducing the number of tasks that are executed on mobile device also reduces its energy consumption.

8.5.3 Trace-driven Results

To further confirm that our results are practical, we generate graphs from execution traces of SPECjvm08 benchmarks. We utilized AspectJ framework to generate traces of six SPEC benchmark programs: compress, scimark.monte-carlo, crypto.aes, mpegaudio, scimark.fft.small and cypto.rsa. These benchmarks were chosen based on the workloads that are most commonly run on mobile devices. Of these benchmarks, the programs compress, scimark.monte-carlo and crypto.aes are compute-intensive. The other programs mpegaudio, crypto.aes

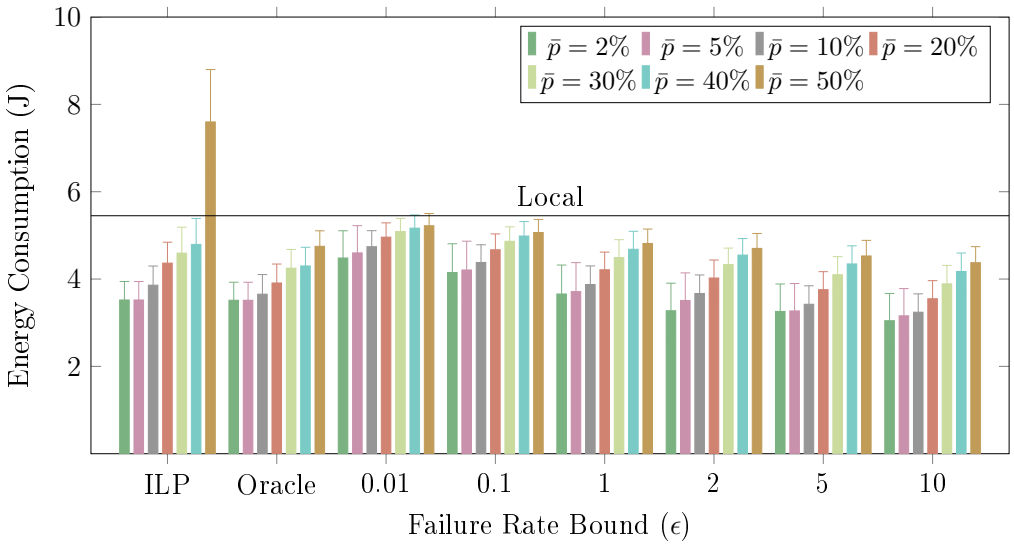


Figure 8.4: Comparison of energy consumption on mobile device at different levels of channel error ($\bar{p}$) using ILP and our heuristic at different failure bounds (ϵ). We have obtained the energy consumption by assuming that processor power = 1 J/s and network power = 0.5 J/s.

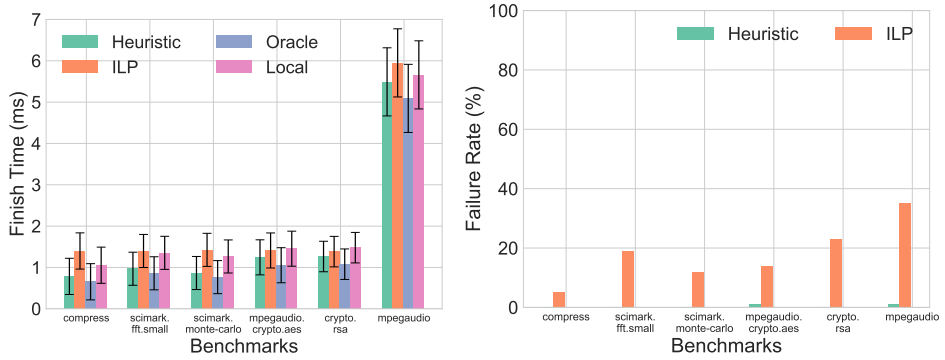


Figure 8.5: Comparison of finish time and failure rate of six different SPECjvm2008 benchmarks using execution on mobile device (local), oracle solution, our heuristic and ILP. Each benchmark has been executed 100 times.

and crypto.rsa are input-intensive as they read data from a file.

Fig. 8.5 show the finish time and failure rate on each of these benchmark programs. We note that in each case, the finish time is lower than the ILP, but higher than the oracle solution. This confirms our finding that our heuristic

gives a better finish time in the presence of channel errors. Moreover, for the input-intensive applications, the ILP solution has a higher finish time than local execution. From the failure plot, we also note that the failure rate is lower than 1% for each of the benchmark programs. This is much lower than the ILP solution, where the failure rates are all higher than 10%.

These observations confirm that our adaptive heuristic works on realistic workloads. Moreover, input-intensive applications require higher number of migrations, and thus lead to more failures using an ILP-based solution. Our heuristic can reduce failure while executing input-intensive applications by reducing the number of tasks executed on cloud server when the channel error probability is high.

8.6 Conclusion

Offloading of mobile applications to cloud servers can augment the limited compute capacity of their processors. However, the quality of offloading based execution depends on the network parameters, like channel error conditions. Unbounded retransmissions to handle channel errors can lead to service degradation as it may end up taking longer than local execution time to complete the application. In this work, we propose an adaptive algorithm that tracks the channel error, defines a stochastic model to capture channel conditions, and uses it to adjust the number of retransmissions to deliver a better service level guarantee in completing an application compared to optimization solutions. The mean finish time of an application is also comparable to typical solutions. We show the efficacy of our technique on both traces and randomly generated application profiles.

Our study has a few limitations. First, we assume that the channel error during a single migration remains same. This may not hold true in a rapidly varying channel. However, we have shown through simulation that a rapidly varying channel affects finish time only when the amount of channel variation is high. Secondly, our algorithm does not guarantee the minimum possible expected finish time. We provide a heuristic that reduces the application finish time compared to local execution under different channel conditions.

Chapter 9

Scheduling with Task Duplication for Application Offloading

9.1 Introduction

As explained in Chapters 5-8, offloading frameworks model execution of a mobile application as a task graph. A task graph consists of a set of vertices representing the tasks in the application, and a set of edges representing dependencies between tasks. Each task and dependency is annotated with one or more cost representing time or energy. The offloading framework selects tasks for remote execution at application startup in order to reduce time and/or energy. Thus, the algorithm used by the offloading framework to select tasks needs to be fast and has to generate a good schedule to ensure quick startup and time and/or energy savings.

In this chapter, we propose utilizing scheduling using task duplication for execution on mobile device and cloud server. Existing application offloading frameworks partition the task graph into two distinct components for execution on mobile device and cloud server respectively. In this work, we show that allowing a limited number of tasks to execute on both mobile device and cloud server reduces the finish time of application, or makespan. Moreover, unlike graph partitioning, scheduling using duplication can be done in polynomial time. Thus, our technique of task duplication leads us to an algorithm that runs in polynomial time and reduces makespan compared to existing scheduling techniques.

We illustrate the benefit of task duplication with an example. Fig. 6.2

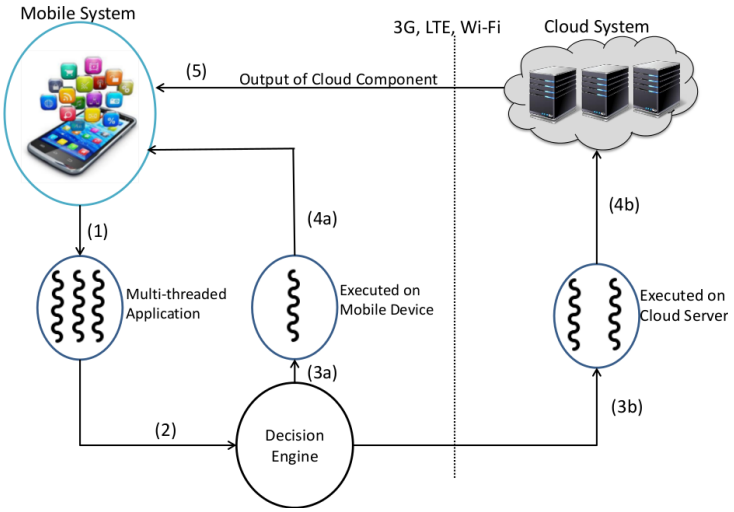


Figure 9.1: Workflow of an offloading framework. Execution of a mobile application, represented as a task graph, is profiled to determine the compute workload of each task. The code partitioning algorithm uses the profile as input to schedule a task locally or on remote server.

shows a task graph, where some tasks marked in gray must execute locally, while others can be scheduled on the device or remote server. Time to execute v_1, v_3, v_6 locally is $10ms$ each, while v_2, v_4, v_5 is $20ms$ each. Assuming that the remote server is 5 times faster than the device, time to execute v_2, v_4, v_5 on cloud is $4ms$. The communication latency due to data transfer is $10ms$ for each edge. With this setting, complete local execution without offloading takes $80ms$, where v_3 and v_4 can be executed in parallel on a multi-core mobile processor. Formulating the problem as an ILP, a solver schedules v_1, v_3, v_5, and v_6 locally, and v_2 and v_4 remotely, giving a makespan of $78ms$. Now, if duplicate execution is allowed, then v_2 can be executed both locally and remotely, thereby saving the time to transfer data for the dependent tasks v_3 and v_4, where v_3 is scheduled locally and v_4 on cloud. This leads to a makespan of $70ms$, showing the benefit of task duplication.

The rest of the chapter is organized as follows. Section 9.2 develops a formulation of the task scheduling problem. Section 9.3 presents the polynomial task scheduling algorithm, ATOM. Sections 9.4 and 9.5 present the evaluation of ATOM using simulation and real-world application traces respectively. Related work is presented in Section 9.6. We conclude in Section 9.7.

Offloading Framework	Optimization Objective	Constraint Parameter	Application Type	Solution Technique	Type of Solution	Scheduling Time Complexity
MAUI [31]	Energy	Time	Sequential	ILP	Optimal	Exponential $[O(2^n)]$
CloneCloud [29]	Energy	Time	Concurrent	ILP	Optimal	Exponential $[O(2^n)]$
ThinkAir [70]	Energy, Time		Concurrent	Heuristic	No performance bound	Polynomial $[O(n)]$
Hermes [65]	Time	Energy	Subset of concurrent	Algorithm	Near-optimal	Polynomial $[O(n^4m^2)]$
Tango [48]	Time		Concurrent	Heuristic using duplicate execution	No performance bound	Constant
ATOM (Our Work)	Time		Concurrent	Dynamic Programming Algorithm	Optimal	Polynomial $[O(m^2n^2)]$

Table 9.1: Comparison of different offloading approaches. n and m represent the number of tasks in the task graph and number of servers in the offloading system respectively.

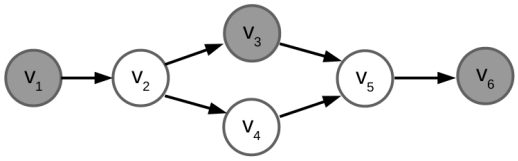
9.2 Problem Formulation

A mobile cloud computing (MCC) system comprises of a mobile device (denoted by $\mathcal{M}_0$) and multiple cloud servers (denoted by $\mathcal{M}_k$, where $1 \leq k \leq m$). We assume that each of these machines have unbounded number of processors. Moreover, processors on each machine are homogeneous.

We represent execution of a mobile application as a directed acyclic graph (DAG) $G = (\mathbb{V}, \mathbb{E})$, where the vertex set $\mathbb{V}$ represents the set of n methods or tasks, and the edge set $\mathbb{E}$ represents the dependencies among tasks. A task v_j may be executed on one or more of the available machines $\mathcal{M}_k (0 \leq k \leq m)$. However, the first task v_1 and the last task v_n must be executed locally on mobile device $\mathcal{M}_0$. Execution of some other tasks may also be tied to the mobile device, as they may depend on some hardware such as camera, GPS, etc. Execution of v_j on $\mathcal{M}_k$ takes t_j^k time. If for a dependency (v_i, v_j), v_j is executed on a different machine $\mathcal{M}_k$ than v_i's machine $\mathcal{M}_h$, then data associated with (v_i, v_j) must be migrated to $\mathcal{M}_k$ before v_j can begin execution. Migrating this data takes r_{ij}^{hk} time. However, migrating from a different processor within the same machine is assumed to take negligible time, i.e. $r_{ij}^{kk} = 0 \ \forall k = 0, ...m, \forall (v_i, v_j) \in \mathbb{E}$. We assume that both execution times t_j^k and migration times r_{ij}^{hk} are obtained by prior profiling of the application.

We define makespan as the time T_n^0 to finish execution of the last task v_n on $\mathcal{M}_0$. We now define the execution finish time of each task v_j. Let T_j^k be the execution finish time of v_j on $\mathcal{M}_k$. Let S_j^k denote the time when execution of v_j on $\mathcal{M}_k$ starts. Then, the finish time of v_j is the sum of start time S_j^k and execution time t_j^k:

$$\forall v_j \in \mathbb{V}, \forall k = 0, \ldots, m, \quad T_j^k = S_j^k + t_j^k \tag{9.1}$$



(a) A task graph representing a mobile application. Tasks marked in gray must be executed locally on the mobile device, while the remaining tasks can be scheduled locally or on remote servers.

	v_1	v_2	v_3	v_4	v_5	v_6
Mobile device (t_j^0)	10	20	10	20	20	10
Cloud server (t_j^1)	NA	4	NA	4	4	NA

(b) Execution times of each task.

Figure 9.2: A task graph along with its parameters. We assume a single cloud server, with a communication time of $10ms$ between the mobile device and cloud server for each edge.

$\mathbb{V}$	Vertex set of the graph
$\mathbb{E}$	Edge set of the graph
v_j	A task in the application execution graph
v_1	First task in the application execution graph
v_n	Last task in the application execution graph
m	Number of servers in the offloading system
n	Number of tasks in the task graph
(v_i, v_j)	A dependency from the task v_i to v_j
$\mathcal{M}_0$	Mobile device
$\mathcal{M}_k$	A machine with multiple processors
t_j^k	Execution time of task v_j on machine $\mathcal{M}_k$
r_{ij}^{hk}	Time to migrate data of (v_i, v_j) from $\mathcal{M}_h$ to $\mathcal{M}_k$
x_j^k	Decision variable indicating execution of v_j on $\mathcal{M}_k$
T_j^k	Finish time of v_j on $\mathcal{M}_k$
S_j^k	Start time of v_j on $\mathcal{M}_k$
D_{ij}^k	Data arrival time of (v_i, v_j) on $\mathcal{M}_k$
T_n^0	Finish time of last task on mobile device, i.e. makespan

Table 9.2: Symbols introduced in Section 9.2

To find the start time S_j^k of v_j on $\mathcal{M}_k$, we note that v_j can start when all its predecessor v_i's are available. Let D_{ij}^k denote the time when data associated with (v_i, v_j) becomes available on $\mathcal{M}_k$. Since each machine $\mathcal{M}_k$ has multiple processors, a task can be executed as soon as its data is available. Thus, the earliest start time is equal to the highest value of data arrival time:

$$\forall v_j \in \mathbb{V}, \forall k = 0, \ldots, m, \quad S_j^k = \max_{(v_i, v_j)} D_{ij}^k \tag{9.2}$$

For the first task v_1, there are no predecessors. Moreover, it can be executed only on the mobile device $\mathcal{M}_0$. Thus, for the first task, we say that start time on $\mathcal{M}_0$ as 0, and all other machines $\mathcal{M}_1, ...$ as ∞:

$$\begin{gathered} S_1^0 = 0, \\ \forall k = 1, ...m, \quad S_1^k = \infty \end{gathered} \tag{9.3}$$

The data arrival time of (v_i, v_j) is the sum of finish time T_h^k of v_i on any $\mathcal{M}_h$ and migration time r_{ij}^{hk}. However, since v_i can execute on many $\mathcal{M}_h$'s, and we are looking for the lowest possible data arrival time, we have:

$$\forall (v_i, v_j) \in \mathbb{E}, \forall k = 0, \ldots, m, \quad D_{ij}^k = \min_{h=0,\ldots,m} (T_i^h + r_{ij}^{hk}) \tag{9.4}$$

Eqns 9.1 to 9.4 give us a recurrence relation that computes the minimum makespan. However, we note that a particular task v_j is only executed on one or more $\mathcal{M}_k$'s. Let x_j^k be a decision variable denoting whether v_j is executed on $\mathcal{M}_k$, i.e.

$$x_j^k = \begin{cases} 1, \text{ if } v_j \text{ is executed on } \mathcal{M}_k, \text{ and} \\ 0, \text{ if } v_j \text{ is not executed on } \mathcal{M}_k. \end{cases}$$

Then, we rewrite Eqn 9.1 in terms of x_j^k as:

$$T_j^k = \begin{cases} S_j^k + t_j^k, & \text{if } x_j^k = 1, \\ \infty, & \text{if } x_j^k = 0. \end{cases}$$

We need to design an algorithm to choose values of x_j^k's that minimizes makespan T_n^0. We utilize the recurrence relation to design a dynamic programming algorithm.

9.3 Our Proposed Algorithm

Our algorithm starts by assuming that each v_j is executed on machines $\mathcal{M}_k$'s. Thus, for each v_j, the output of its predecessor v_i is available on each $\mathcal{M}_k$. Before execution of v_j on $\mathcal{M}_k$ begins, we need to determine which $\mathcal{M}_h$ can send the data associated with (v_i, v_j) the fastest. We store the fastest time when data of (v_i, v_j) arrives at $\mathcal{M}_k$ in D_{ij}^k and store the corresponding value of h in a lookup table. When all the predecessors v_i's have arrived at $\mathcal{M}_k$, execution of v_j can start. This value of time, equal to the maximum value of D_{ij}^k across all v_i's, is stored in S_j^k. The time taken to finish execution of v_j, T_j^k is the sum of start time S_j^k and execution time t_j^k. By calculating recursively the finish times of each task, we obtain the finish time of the last task, or makespan T_0^n. Once the makespan is obtained, we use the lookup table to determine the machines M_h from each output of each predecessor v_i's was used. This lets us get the execution machines of each task. The exact algorithm is shown in detail in Algorithm 5. Table 9.3 shows the working of the algorithm on our example task graph shown in Fig. 6.2.

Current Task v_j	Predecessor Task v_i	Data Arrival Time D_{ij}^0	Start Time S_j^0	Location of Predecessor $Lookup_{ij}^0$	Finish Time T_j^0	Data Arrival Time D_{ij}^1	Start Time of Current Task S_j^1	Location of Predecessor $Lookup_{ij}^1$	Finish Time T_j^1
v_2	v_1	10	10	Mobile	30	20	20	Mobile	24
v_3	v_3	$\min(30, 22+10) = 30$	30	Mobile	40				
v_4	v_2	$\min(30, 22+10) = 30$	30	Mobile	50	$\min(30+10, 24) = 24$	24	Cloud	28
v_5	v_3	40	40	Mobile	60	$40+10 = 50$	50	Mobile	54
	v_4	$\min(50, 28+10) = 38$		Cloud		28		Cloud	
v_6	v_5	$\min(60, 52+10) = 60$	60	Mobile	70				

Table 9.3: Table to minimize makespan of task graph shown in Fig. 9.2 used by Algorithm 5

To analyze the time complexity of ATOM, we first analyze Procedure CALCULATE-MAKESPAN. We note that the loop on Line 4 runs $n-1$ times, once for each task in the DAG. The inner loop (on Line 5) runs once for each predecessor task, i.e. the number of incoming edges in the DAG. Let the number of such incoming edges to a task v_i be d_i. The loops on Lines 6 and 7 iterate a total of m^2 times. Within the innermost loop on Line 7, each step requires constant ($O(1)$) time. Thus, total number of steps to run the procedure is given by:

$$\mathcal{T}_1(n) = \sum_{i=1}^{n-1} d_i = O(m^2|\mathbb{E}|).$$

Algorithm 5 Algorithm ATOM to compute makespan and obtain execution schedule of an application execution graph. It accepts a DAG $G = (\mathbb{V}, \mathbb{E})$ representing a mobile application as input. It returns the makespan T_n^0 and decision variable x_j^k indicating whether v_j should be executed on $\mathcal{M}_k$.

1: **procedure** CALCULATE-MAKESPAN
2: $T_1^0 \leftarrow t_1^0$
3: $T_1^1 \leftarrow \infty$
4: **for** $j = 2$ to n **do**
5: **for all** predecessors v_i of v_j **do**
6: **for all** $k = 0$ to m **do**
7: **for all** $h = 0$ to m **do**
8: **if** $T_i^k < T_i^h + r_{ij}^{hk}$ **then**
9: $D_{ij}^k \leftarrow T_i^k$
10: $Lookup_{ij}^k \leftarrow k$
11: **else**
12: $D_{ij}^k \leftarrow T_i^k + r_{ij}^{hk}$
13: $Lookup_{ij}^k \leftarrow h$
14: **for all** $k = 0$ to m **do**
15: $S_j^k \leftarrow \max_{(i,j)\in\mathbb{E}}\{D_{ij}^k\}$
16: $T_j^k \leftarrow S_j^k + t_j^k$
17: **if** v_j is tied to mobile AND $k \neq 0$ **then**
18: $T_j^k \leftarrow \infty$
return T

Similarly, Procedure GET-SCHEDULE also has an outer loop running $n-1$ times, and an inner loop for each predecessor. Each inner loop requires m number of times. Thus, time complexity of Procedure GET-SCHEDULE, $\mathcal{T}_2(n)$ is also $O(m|\mathbb{E}|)$. Therefore, time complexity of our proposed algorithm is given by:

$$\mathcal{T}(n) = \mathcal{T}_1(n) + \mathcal{T}_2(n) = O(m^2|\mathbb{E}|) + O(m|\mathbb{E}|) = O(m^2|\mathbb{E}|).$$

Since the number of dependencies is of the order of $O(n^2)$, this gives us a time complexity of $O(m^2n^2)$, where m and n are the number of servers and number of tasks respectively.

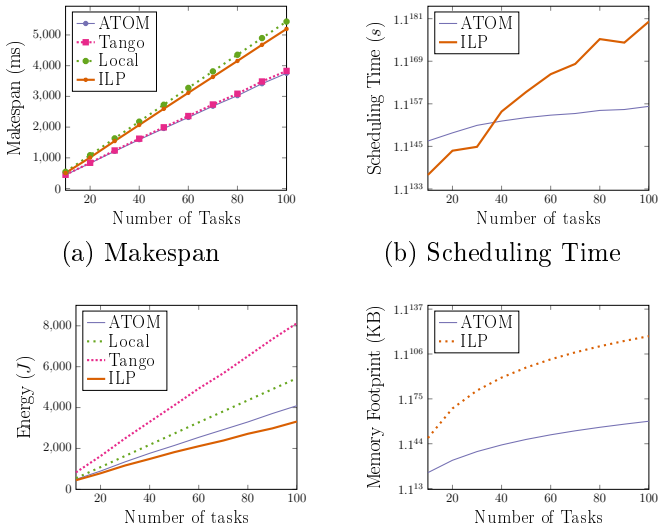
We now explain how task duplication reduces makespan in our algorithm. First, we note that a task is duplicated only when new threads are spawned. When a new thread is spawned, one thread may be faster on the mobile

device, while the other thread is faster on cloud server. In this case, executing one or more tasks preceding the spawning of the thread on both mobile device and cloud server may be faster. For example, in Fig. 9.2, a new thread is spawned at v_2. Thus, v_2 has two outgoing edges connecting v_3 and v_4. If we execute v_2 only on $\mathcal{M}_0$ (mobile), then migrating (v_2, v_4) and then executing v_4 on $\mathcal{M}_1$ is slower than executing only v_4 on $\mathcal{M}_0$. Thus, v_4 also executes on mobile device. If we execute v_2 only on $\mathcal{M}_1$ (cloud server), then migrating (v_2, v_3) back to $\mathcal{M}_0$ slows down execution of v_3. On the other hand, if we execute v_2 on both $\mathcal{M}_0$ and $\mathcal{M}_1$, this allows execution of v_3 to start much faster, and also does not require migration of (v_2, v_3).

Another major advantage of allowing task duplication is that it results in a polynomial algorithm. This is because allowing the same task to execute on multiple machines $\mathcal{M}_k$ allows us to divide the entire scheduling of task graphs into smaller scheduling problems. For example, in Fig. 9.2, it is possible to separately schedule the tasks v_1, v_2, v_3, v_5, v_6 in one step, and v_1, v_2, v_4, v_5, v_6 separately in another step. If any v_j is scheduled on two different machines $\mathcal{M}_h$ and $\mathcal{M}_k$, then it can be executed on both. Since scheduling a linear sequence of tasks is polynomial, using task duplication reduces the problem to a series of polynomial problems. Thus, the overall scheduling problem also becomes polynomial when tasks duplication is allowed.

9.4 Simulation-based Evaluation

In this section, we compare ATOM with schedules generated by Integer Linear Programming (ILP), Tango [48] and local execution. We implemented the ILP (discussed in Section 9.2), Tango and ATOM on an Intel Xeon (CPU: E5-2630) 6-core processor system in Java (openJDK 1.7) programming language. We generated call graphs of different sizes ranging from 10 to 100, with each size of call graph having 100 random samples each. We study different performance parameters like makespan, scheduling time, energy consumption and memory footprint of scheduling algorithms. We use Java ThreadMXBean interface to measure scheduling time, the energy model discussed in Section 9.2 to measure energy consumption and Java Instrumentation to measure memory footprint [112].



(a) Makespan (b) Scheduling Time

(c) Energy Consumption (d) Memory footprint

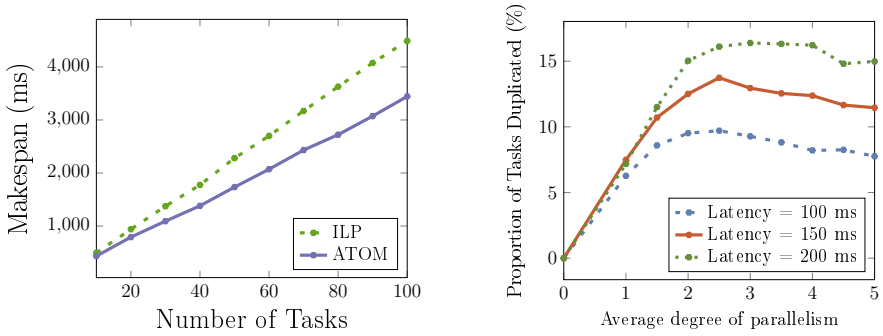
Figure 9.3: Comparison of makespan, scheduling time, energy consumption and memory footprint of ATOM, ILP, Tango and local execution.

9.4.1 Performance Comparison

Makespan We compare the makespans of different algorithms. Fig. 9.3(a) shows the makespan for different number of tasks in the application graph. We note that ATOM provides the smallest makespan, followed by Tango, ILP and local execution. This is because ATOM always provides the optimal makespan, and thus its maskespan must be the smallest across different methods for any given application.

Scheduling Time Fig. 9.3(b) shows the scheduling time of ATOM and ILP for different number of tasks from 10 to 100. We omit Tango and local execution here since these techniques do not need to run any scheduling algorithm during startup. We note that for smaller applications with less than 40 tasks, an ILP is faster. For larger applications, the scheduling time of an ILP increases rapidly. This is because solving an ILP takes exponential time, whereas ATOM is a polynomial algorithm.

Energy Consumption Fig. 9.3(c) shows the energy consumption using the four different methods. We note that an ILP consumes the least amount of energy, followed by ATOM, local execution and Tango. This is because ATOM uses task duplication to save time. However, this consumes additional



(a) Makespan for different number of tasks (b) Amount of task duplication for different degrees of parallelism

Figure 9.4: Comparison of makespan of ATOM with an ILP formulation to obtain the effect of task duplication. The ILP formulation does not use task duplication.

energy on mobile device, since this requires both local execution and migration. Thus, Tango consumes the highest energy, since it duplicates all tasks on both mobile device and server.

Memory Footprint Fig. 9.3(d) shows the memory footprint of ATOM and ILP. To account for the large differences in memory footprint, we plot it on a logarithmic scale. We once again note that running ATOM consumes much smaller memory than an ILP. This is because an ILP formulation requires storing a large matrix as input. The space complexity of ATOM is linear with additional memory only being used to store the start times, finish times and execution platforms of each task.

9.4.2 Effect of Task Duplication

We now study the amount of task duplication performed by ATOM, and its effect on the makespan. We note that unlike general DAGs, tree-structured graphs do not require any task duplication to minimize makespan. Thus, we generate random general DAGs for these experiments.

Makespan To understand the effect of task duplication on makespan, we utilize the formulation described in Section 9.2. In the formulation described in Section 9.2, we add an additional constraint to ensure that no task redundancy

is used:

$$\forall v_j \in \mathbb{V}, \sum_{k=0}^{m} x_j^k = 1 \tag{9.5}$$

We then compare the makespan given by the ILP with ATOM.

Fig. 9.4(a) shows the difference in makespan using the ILP and ATOM for different number of tasks. We note that ATOM has a lower makespan than the ILP in each case. Moreover, the difference in makespan increases with an increase in the number of tasks. Thus, for 10 tasks, ATOM has 12% lower makespan than the ILP. For 100 tasks, this increases to 25%.

This observation is explained by noting that task duplication reduces makespan. The increase in time saving with an increase in size of DAG also shows that task duplication saves more time for larger DAGs. This is because larger graphs have more scope for exploitation of parallelism, which can be better exploited with lower costs using task duplication.

Amount of task duplication Fig. 9.4(b) shows the amount of task duplication performed by ATOM for different amounts of available parallelism. We note that when out-degree is equal to 1, the application is completely sequential. Thus, no task duplication is used. The amount of task duplication reaches a peak of around 10% when the maximum out-degree is 5. Further increase in out-degree of tasks slightly reduces the amount of task duplication.

This observation confirms that task duplication reduces makespan by reducing communication cost of parallel execution. When there is more concurrency in the application, more parallelism can be utilized by utilizing more duplication. Thus, when more threads are spawned, the task amount of duplication increases.

9.5 Trace-Based Evaluation

We perform trace-driven simulation on benchmark programs, and compare its performance with other algorithms. To obtain traces from any available Java program, we utilize aspect-oriented programming using AspectJ framework [68]. AspectJ allows programmers to add additional code at the call points of each method through bytecode-level modifications. We use AspectJ to obtain the traces of each method call. We also serialized arguments of each method and printed the size of arguments. This gives us the amount of data required to migrate at any particular call point. Finally, we calculated the time taken

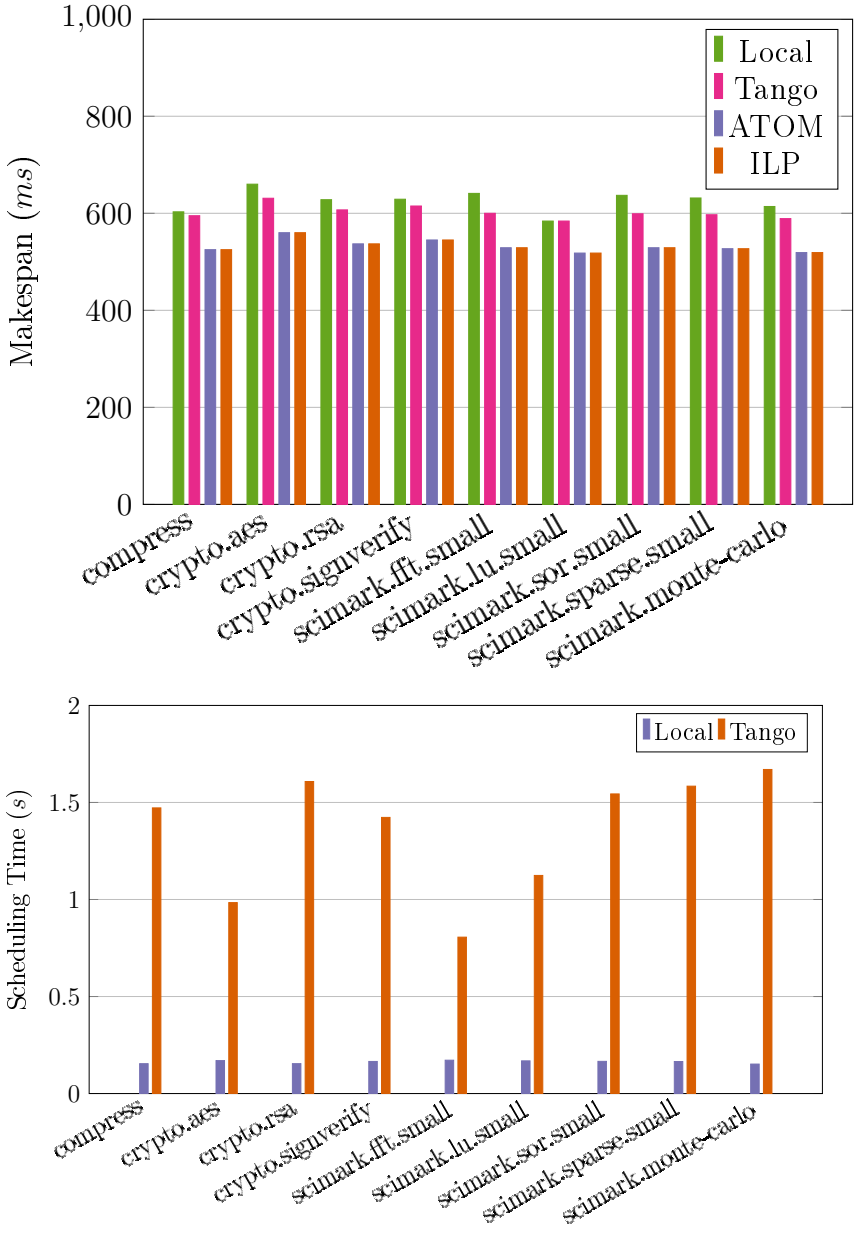


Figure 9.5: Comparison of energy consumption of SPEC benchmarks using ILP, ATOM, Tango and local execution.

to execute each method using Java's ThreadMXBean interface [112]. We use these data to annotate the call graph. We identify the methods that require

access to user input or output device (such as println method) as native.

We perform our experiments on nine selected SPEC JVM benchmarks [108]. The nine benchmarks are selected because they mirror mobile workloads. Thus, we use traces of SPEC JVM benchmarks to get results that are representative of those on real workloads.

9.5.1 Makespan

To understand the effect on execution time, we obtain the makespan or application finish time using ILP, ATOM, Tango and local execution. We use a constant bandwidth of 1 Mbps to run our traces, and a round-trip time (RTT) of $50ms$. Fig. 9.5(a) shows the effect of the four methods on makespan. We note that ATOM reduces the makespan by 15% compared to local execution, and 10% compared to Tango. Moreover, ATOM and ILP gives us almost the same makespan in each case.

9.5.2 Scheduling Time

We now compare the scheduling time of ILP and ATOM in Fig. 9.5(b). We note that the average scheduling time is less than $0.2s$ for ATOM. This is much lower than an ILP, which requires an average of over $1s$ of scheduling time. ATOM reduces the average scheduling time of applications by around 8 times.

9.6 Related Work

To systematically study the varieties of job scheduling problems, they are classified based on machine architecture (α), task model (β) and optimization objective (γ) [51]. This classification scheme is referred to as $\alpha|\beta|\gamma$ model, based on the three parameters of classification. We explain related scheduling algorithms in the context of this scheme.

In our machine architecture, communication costs differ based on the execution platform of each processor. This is known as a cluster machine model, and is denoted by $P(a, b)$. Here a denotes the number of clusters, and b represents the number of processors in each cluster. Thus, in our case, $a = 2$ and $b = \infty$. Precedence constraints between tasks are denoted by *prec*, and task duplication is denoted by *dup*. The objective is to reduce the

makespan or schedule length of the last task on mobile device T_N^m. Thus, this problem is denoted by $P(2,\infty)|prec, dup|$makespan. Most previous studies have proposed scheduling algorithms for machine models that are either completely homogeneous or heterogeneous.

Existing Mobile Cloud Computing frameworks fall into two categories based on their scheduling techniques. MAUI [31] and CloneCloud [29] utilize an Integer-Linear Programming (ILP) solver to optimally schedule tasks in exponential time. The alternative approach, used by ThinkAir [70], utilizes heuristic to schedule tasks. This has a low time complexity, but does not guarantee minimization of time or energy. Hermes [65] presents an approximation scheme to minimize makespan within a given energy budget. Tango [48] uses duplicate execution of all possible tasks on mobile device and server to speed up applications. Our algorithm ATOM combines the advantages of Tango and ILP by guaranteeing minimum makespan while having low time complexity.

9.7 Conclusion

Mobile devices continue to be limited by their compute power. In this setting, offloading parts of the application to resource rich remote servers can enable wide class of applications. Typically offloading algorithms were designed as optimization problems solved as Integer Linear Programs, or using heuristics, thereby lacking performance guarantees, and may scale poorly. We show that allowing duplicate execution of a few selected tasks leads to a polynomial time scheduling algorithm that minimizes the total completion time of an application. Our algorithm ATOM (Algorithm for Time Optimization on Mobiles) determines a schedule to execute tasks of a concurrent application with duplication such that makespan is minimized. Our simulation and trace-driven experiments show that ATOM significantly reduces makespan and energy consumption while executing in polynomial time.

Chapter 10

Conclusion

With the increase in the number of connected devices, it has become increasingly important for emerging wireless applications to provide some performance guarantee and have high scalability. A large number of such applications depend on heuristics that may work well in limited circumstances, but may not scale well or perform well in actual deployments. Thus, it is important for the algorithms used in such applications to provide some type of performance guarantees.

In this thesis, we proposed and analyzed algorithms to optimize two distinct applications in wireless networks. In the first part of this thesis, we specifically looked at the application of low-cost spectrum monitoring using crowdsourced spectrum sensors. The first application deals with crowdsourced spectrum monitoring to detect the source of illegal transmitters. We made four distinct contributions. We first propose using a variant of Maximum Relevance Minimum Redundance (MRMR) to select the most relevant spectrum sensors. Although this technique does not give any performance bound with the optimal, we show using experiments that it performs well in practice. We also show a technique of combining the local decision of the individual sensors to get the optimal global decision.

We next propose an algorithm called Auxiliary Greedy Algorithm (AGA) that selects the sensors that should be run to maximize performance within a given budget. Although finding the optimal is NP-Hard, we provide a performance bound of our algorithm with the optimal. This algorithm utilizes the idea that a monotonous submodular objective set function can be approximately maximized using a greedy algorithm.

Our next contribution is to propose selecting sensors using an algorithm

called Online Greedy Algorithm (NGA). In contrast to the above techniques, NGA selects the sensors for probing, and utilizes the results of the probe to probe the subsequent sensors. We show that NGA can be much more accurate than AGA, albeit at a cost of higher latency.

Our fourth contribution on spectrum monitoring is to improve the efficiency of individual sensors by utilizing FPGA-based spectrum sensors. FPGA-based sensors substantially reduce the energy and latency cost of running the sensors, leading to a reduction in overall running of a spectrum monitoring system.

In the second part of this thesis, we deal with computation offloading from mobile devices to servers. Our first contribution is to thoroughly evaluate the performance of such offloading, in diverse network conditions and different server conditions and applications. We show that in many cases, smaller devices closer to smartphones can provide lower latency than more powerful servers hosted in data centers.

Our second contribution is to propose an algorithm that gives a probabilistic guarantee of finishing the execution of any given application within the budgeted time. We show that our algorithm, apart from guarantees, also provides on average faster execution time than a static optimization solver.

Our third contribution is to propose a dynamic programming based algorithm to minimize the execution time of a given application. Conventional techniques use an ILP solver that can take a long time to execute. In contrast, our algorithm minimizes the execution time in polynomial time. We show that our algorithm reduces application execution time, while also running in an order of magnitude than other conventional schedulers.

10.1 Future Work

Our current work on spectrum monitoring focuses on cases where the transmitters are static in nature. However, recently there have been cases of illegal transmitters such as GPS spoofers installed in moving vehicles. Efficient localization and identification of moving transmitters remains a challenge. We plan to extend our work to such moving transmitters to protect the spectrum from such illegal users.

With the emergence of paradigms such as fog computing, there is a need to consider additional Quality of Experience parameters such as cost, amount of data transmitted, and reliability in case of failures. Currently our work

focuses on optimizing application finish time and energy consumption. For future work, we plan to provide similar scalable performances guarantees for these other Quality of Experience parameters.

Bibliography

[1] Buke Ao, Yongcai Wang, Lu Yu, Richard R. Brooks, and S. S. Iyengar. "On Precision Bound of Distributed Fault-Tolerant Sensor Fusion Algorithms". In: *ACM Computing Surveys* 49.1 (May 2016), pp. 1–23. DOI: 10.1145/2898984.

[2] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. "Energy consumption in mobile phones. a measurement study and implications for network applications". In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*. ACM. 2009, pp. 280–293. DOI: 10.1145/1644893.1644927.

[3] Marco V. Barbera, Sokol Kosta, Alessandro Mei, Vasile C. Perta, and Julinda Stefa. "Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. Apr. 2014. DOI: 10.1109/infocom.2014.6848180.

[4] Marco V. Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing". In: *2013 Proceedings IEEE INFOCOM*. Apr. 2013, pp. 1285–1293. DOI: 10.1109/infcom.2013.6566921.

[5] Arani Bhattacharya, Ansuman Banerjee, and Pradipta De. "Parametric Analysis of Mobile Cloud Computing Frameworks using Simulation Modeling". In: *International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*. Springer. 2015, pp. 32–48.

[6] Arani Bhattacharya, Ansuman Banerjee, and Pradipta De. "Scheduling with task duplication for application offloading". In: *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. Jan. 2017, pp. 678–683. DOI: 10.1109/ccnc.2017.7983212.

[7] Arani Bhattacharya, Ansuman Banerjee, and Pradipta De. "Service Level Guarantee for Mobile Application Offloading in Presence of Wireless Channel Errors". In: *2016 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2016, pp. 1–6. DOI: 10.1109/glocom.2016.7842264.

[8] Arani Bhattacharya, Ayon Chakraborty, Samir R. Das, Himanshu Gupta, and Petar M. Djuric. "Spectrum Patrolling with Crowdsourced Spectrum Sensors". In: *IEEE Transactions on Cognitive Communications and Networking* (2019), pp. 1–1. DOI: 10.1109/tccn.2019.2939793.

[9] Arani Bhattacharya, Han Chen, Peter Milder, and Samir R. Das. "Quantifying Energy and Latency Improvements of FPGA-Based Sensors for Low-Cost Spectrum Monitoring". In: *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. Oct. 2018, pp. 1–5. DOI: 10.1109/dyspan.2018.8610459.

[10] Arani Bhattcharya and Pradipta De. "Computation Offloading from Mobile Devices. Can Edge Devices Perform Better Than the Cloud?" In: *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing - ARMS-CC'16*. ACM. 2016, pp. 1–6. DOI: 10.1145/2962564.2962569.

[11] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. Helsinki, Finland, 2012, pp. 13–16. DOI: 10.1145/2342509.2342513.

[12] Alexandros-Apostolos A. Boulogeorgos, Nestor D. Chatzidiamantis, and George K. Karagiannidis. "Energy Detection Spectrum Sensing Under RF Imperfections". In: *IEEE Transactions on Communications* 64.7 (July 2016), pp. 2754–2766. DOI: 10.1109/tcomm.2016.2561294.

[13] Alexandros-Apostolos A. Boulogeorgos, Haythem A. Bany Salameh, and George K. Karagiannidis. "Spectrum Sensing in Full-Duplex Cognitive Radio Networks Under Hardware Imperfections". In: *IEEE Transactions on Vehicular Technology* 66.3 (Mar. 2017), pp. 2072–2084. DOI: 10.1109/tvt.2016.2582790.

[14] Danijela Cabric, Artem Tkachenko, and Robert W. Brodersen. "Experimental study of spectrum sensing based on energy detection and network cooperation". In: *Proceedings of the first international workshop on Technology and policy for accessing spectrum - TAPAS '06*. ACM. 2006, p. 12. DOI: 10.1145/1234388.1234400.

[15] Angela Sara Cacciapuoti, Ian F. Akyildiz, and Luigi Paura. "Correlation-Aware User Selection for Cooperative Spectrum Sensing in Cognitive Radio Ad Hoc Networks". In: *IEEE Journal on Selected Areas in Communications* 30.2 (Feb. 2012), pp. 297–306. DOI: 10.1109/jsac.2012.120208.

[16] Roberto Calvo-Palomino, Domenico Giustiniano, Vincent Lenders, and Aymen Fakhreddine. "Crowdsourcing spectrum data decoding". In: (2017).

[17] Roberto Calvo-Palomino, Domenico Giustiniano, Vincent Lenders, and Aymen Fakhreddine. "Crowdsourcing spectrum data decoding". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. May 2017, pp. 1–9. DOI: 10.1109/infocom.2017.8057078.

[18] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone." In: *USENIX annual technical conference*. 2010, pp. 271–285.

[19] Volkan Cevher, Petros Boufounos, Richard G Baraniuk, Anna C Gilbert, and Martin J Strauss. "Near-optimal Bayesian localization via incoherence and sparsity". In: *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. IEEE Computer Society. 2009, pp. 205–216.

[20] Z. Chair and P.K. Varshney. "Optimal Data Fusion in Multiple Sensor Detection Systems". In: *IEEE Transactions on Aerospace and Electronic Systems* AES-22.1 (Jan. 1986), pp. 98–101. DOI: 10.1109/taes.1986.310699.

[21] G V Chaitanya, P. Rajalakshmi, and U. B. Desai. "Real time hardware implementable spectrum sensor for Cognitive Radio applications". In: *2012 International Conference on Signal Processing and Communications (SPCOM)*. July 2012, pp. 1–5. DOI: 10.1109/spcom.2012.6290024.

[22] Ayon Chakraborty, Arani Bhattacharya, Snigdha Kamal, Samir R. Das, Himanshu Gupta, and Petar M. Djuric. "Spectrum Patrolling with Crowdsourced Spectrum Sensors". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Apr. 2018. DOI: 10.1109/infocom.2018.8486343.

[23] Ayon Chakraborty, Udit Gupta, and Samir R. Das. "Benchmarking resource usage for spectrum sensing on commodity mobile devices". In: *Proceedings of the 3rd Workshop on Hot Topics in Wireless - HotWireless '16*. 2016. DOI: 10.1145/2980115.2980129.

[24] Ayon Chakraborty, Md. Shaifur Rahman, Himanshu Gupta, and Samir R. Das. "SpecSense: Crowdsensing for efficient querying of spectrum occupancy". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. May 2017. DOI: 10.1109/infocom.2017.8057113.

[25] K. Chamberlin and R. Luebbers. "An evaluation of Longley-Rice and GTD propagation models". In: *IEEE Transactions on Antennas and Propagation* 30.6 (Nov. 1982), pp. 1093–1098. DOI: 10.1109/tap.1982.1142958.

[26] Jian Chen, Andrew Gibson, and Junaid Zafar. "Cyclostationary spectrum detection in cognitive radios". In: *IET Seminar on Cognitive Radio and Software Defined Radio: Technologies and Techniques*. IET. 2008. DOI: 10.1049/ic:20080398.

[27] Yuxin Chen, S Hamed Hassani, Amin Karbasi, and Andreas Krause. "Sequential information maximization: When is greedy near-optimal?" In: *Conference on Learning Theory*. 2015, pp. 338–363.

[28] Kwang-Ting Cheng and Yi-Chu Wang. "Using mobile GPU for general-purpose computing–a case study of face recognition on smartphones". In: *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*. 2011, pp. 1–4.

[29] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. "CloneCloud. elastic execution between mobile device and cloud". In: *Proceedings of the sixth conference on Computer systems - EuroSys '11*. Salzburg, Austria, 2011, pp. 301–314. DOI: 10.1145/1966445.1966473.

[30] Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi. "Can execution time describe accurately the energy consumption of mobile apps? an experiment in Android". In: *Proceedings of the 3rd International Workshop on Green and Sustainable Software - GREENS 2014*. ACM. 2014, pp. 31–37. DOI: 10.1145/2593743.2593748.

[31] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. "Maui. making smartphones last longer with code offload". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*. San Francisco, California, USA, 2010, pp. 49–62. DOI: 10.1145/1814433.1814441.

[32] Mallesham Dasari, Muhammad Bershgal Atique, Arani Bhattacharya, and Samir R. Das. "Spectrum Protection from Micro-transmissions Using Distributed Spectrum Patrolling". In: *Passive and Active Measurement*. Cham, 2019, pp. 244–257.

[33] Fadel F. Digham, Mohamed-Slim Alouini, and Marvin K. Simon. "On the Energy Detection of Unknown Signals Over Fading Channels". In: *IEEE Transactions on Communications* 55.1 (Jan. 2007), pp. 21–24. DOI: 10.1109/tcomm.2006.887483.

[34] E. Drakopoulos and C.-C. Lee. "Optimum multisensor fusion of correlated local decisions". In: *IEEE Transactions on Aerospace and Electronic Systems* 27.4 (July 1991), pp. 593–606. DOI: 10.1109/7.85032.

[35] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[36] Aveek Dutta and Mung Chiang. ""See Something, Say Something" Crowdsourced Enforcement of Spectrum Policies". In: *IEEE Transactions on Wireless Communications* 15.1 (Jan. 2016), pp. 67–80. DOI: 10.1109/twc.2015.2466550.

[37] *EEMBC – The Embedded Microprocessor Benchmark Consortium*. http://www.eembc.org/coremark. Online; accessed 26 February 2016. 2015.

[38] E. Erdoğan and G. Iyengar. "Ambiguous chance constrained problems and robust optimization". In: *Mathematical Programming* 107.1-2 (Dec. 2005), pp. 37–61. DOI: 10.1007/s10107-005-0678-0.

[39] *FCC Proposes Levying Huge Fine on New York Police Radio Jammer, ARRL news*. http://bit.ly/2uPsB1P.

[40] *FlightFeeder for Android, FlightAware*. http://flightaware.com/adsb/android/.

[41] Wei Gao, Yong Li, Haoyang Lu, Ting Wang, and Cong Liu. "On Exploiting Dynamic Execution Patterns for Workload Offloading in Mobile Cloud Applications". In: *2014 IEEE 22nd International Conference on Network Protocols*. Oct. 2014, pp. 1–12. DOI: 10.1109/icnp.2014.22.

[42] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. "Edge-centric Computing. Vision and Challenges". In: *ACM SIGCOMM Computer Communication Review* 45.5 (Sept. 2015), pp. 37–42. DOI: 10.1145/2831347.2831354.

[43] A. Ghasemi and E.S. Sousa. "Collaborative spectrum sensing for opportunistic access in fading environments". In: *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.* 2005. DOI: 10.1109/dyspan.2005.1542627.

[44] Ahmet Gokceoglu, Sener Dikmese, Mikko Valkama, and Markku Renfors. "Energy Detection under IQ Imbalance with Single- and Multi-Channel Direct-Conversion Receiver: Analysis and Mitigation". In: *IEEE Journal on Selected Areas in Communications* 32.3 (Mar. 2014), pp. 411–424. DOI: 10.1109/jsac.2014.1403001.

[45] Ahmet Gokceoglu, Sener Dikmese, Mikko Valkama, and Markku Renfors. "Enhanced energy detection for multi-band spectrum sensing under RF imperfections". In: *8th International Conference on Cognitive Radio Oriented Wireless Networks*. July 2013. DOI: 10.1109/crowncom.2013.6636794.

[46] Daniel Golovin and Andreas Krause. "Adaptive submodularity: Theory and applications in active learning and stochastic optimization". In: *Journal of Artificial Intelligence Research* 42 (2011), pp. 427–486.

[47] *Google Cloud Computing, Hosting Platform & APIs*. cloud.google.com. Online; accessed 26 February 2016. 2016.

[48] Mark S. Gordon, David Ke Hong, Peter M. Chen, Jason Flinn, Scott Mahlke, and Zhuoqing Morley Mao. "Accelerating Mobile Applications through Flip-Flop Replication". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '15*. ACM. 2015, pp. 137–150. DOI: 10.1145/2742647.2742649.

[49] R. W. Gosper. "Decision procedure for indefinite hypergeometric summation". In: *Proceedings of the National Academy of Sciences* 75.1 (Jan. 1978), pp. 40–42. DOI: 10.1073/pnas.75.1.40.

[50] Javier Gozalvez, Miguel Sepulcre, and Ramon Bauza. "Impact of the radio channel modelling on the performance of VANET communication protocols". In: *Telecommunication Systems* 50.3 (Dec. 2010), pp. 149–167. DOI: 10.1007/s11235-010-9396-x.

[51] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. "Optimization and approximation in deterministic sequencing and scheduling: A survey". In: *Annals of discrete mathematics* 5 (1979), pp. 287–326.

[52] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. "Queues Don't Matter When You Can JUMP Them!" In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA, May 2015, pp. 1–14.

[53] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. "Predictable 802.11 packet delivery from wireless channel measurements". In: *ACM SIGCOMM Computer Communication Review* 40.4 (Aug. 2010), p. 159. DOI: 10.1145/1851275.1851203.

[54] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction". In: *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Mar. 2016, pp. 64–76. DOI: 10.1109/hpca.2016.7446054.

[55] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. "Measuring energy consumption for short code paths using RAPL". In: *ACM SIGMETRICS Performance Evaluation Review* 40.3 (Jan. 2012), p. 13. DOI: 10.1145/2425248.2425252.

[56] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (Mar. 1963), pp. 13–30. DOI: 10.1080/01621459.1963.10500830.

[57] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. "Mobile fog. a programming model for large-scale applications on the internet of things". In: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing - MCC '13*. Hong Kong, China, 2013, pp. 15–20. DOI: 10.1145/2491266.2491270.

[58] S. Huang, X. Liu, and Z. Ding. "Opportunistic Spectrum Access in Cognitive Radio Networks". In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*. Apr. 2008. DOI: 10.1109/infocom.2008.201.

[59] *Intel Core i7-6700 Processor*. https://ark.intel.com/products/88196/Intel-Core-i7-6700-Processor-8M-Cache-up-to-4-00-GHz-. Accessed on Sep 21, 2018. 2015.

[60] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. "Silo. Predictable Message Latency in the Cloud". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM '15*. London, United Kingdom, 2015, pp. 435–448. DOI: 10.1145/2785956.2787479.

[61] Xiaocong Jin, Jingchao Sun, Rui Zhang, Yanchao Zhang, and Chi Zhang. "SpecGuard: Spectrum misuse detection in dynamic spectrum access systems". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015. DOI: 10.1109/infocom.2015.7218380.

[62] Joblib. *joblib/joblib*. Oct. 2018.

[63] S. Joshi and S. Boyd. "Sensor Selection via Convex Optimization". In: *IEEE Transactions on Signal Processing* 57.2 (Feb. 2009), pp. 451–462. DOI: 10.1109/tsp.2008.2007095.

[64] M. Kam, Q. Zhu, and W.S. Gray. "Optimal data fusion of correlated local decisions in multiple sensor detection systems". In: *IEEE Transactions on Aerospace and Electronic Systems* 28.3 (July 1992), pp. 916–920. DOI: 10.1109/7.256317.

[65] Yi-Hsuan Kao, Bhaskar Krishnamachari, Moo-Ryong Ra, and Fan Bai. "Hermes: Latency optimal task assignment for resource-constrained mobile computing". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. Apr. 2015. DOI: 10.1109/infocom.2015.7218572.

[66] Mojgan Khaledi, Mehrdad Khaledi, Shamik Sarkar, Sneha Kasera, Neal Patwari, Kurt Derr, and Samuel Ramirez. "Simultaneous Power-Based Localization of Transmitters for Crowdsourced Spectrum Monitoring". In: *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking - MobiCom '17*. ACM. 2017, pp. 235–247. DOI: 10.1145/3117811.3117845.

[67] Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. "The budgeted maximum coverage problem". In: *Information Processing Letters* 70.1 (Apr. 1999), pp. 39–45. DOI: 10.1016/s0020-0190(99)00031-9.

[68] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William Griswold. "Getting started with ASPECTJ". In: *Communications of the ACM* 44.10 (Oct. 2001), pp. 59–65. DOI: 10.1145/383845.383858.

[69] Nikolaus Kleber, Abbas Termos, Gonzalo Martinez, John Merritt, Bertrand Hochwald, Jonathan Chisum, Aaron Striegel, and J. Nicholas Laneman. "RadioHound: A pervasive sensing platform for sub-6 GHz dynamic spectrum monitoring". In: *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. Mar. 2017. DOI: 10.1109/dyspan.2017.7920764.

[70] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading". In: *2012 Proceedings IEEE INFOCOM*. Mar. 2012, pp. 945–953. DOI: 10.1109/infcom.2012.6195845.

[71] Andreas Krause and Carlos E Guestrin. "Near-optimal nonmyopic value of information in graphical models". In: *arXiv preprint arXiv:1207.1394* (2012).

[72] Andreas Krause, Ajit Singh, and Carlos Guestrin. "Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies". In: *Journal of Machine Learning Research* 9.Feb (2008), pp. 235–284.

[73] Young-Woo Kwon and Eli Tilevich. "Energy-Efficient and Fault-Tolerant Distributed Mobile Execution". In: *2012 IEEE 32nd International Conference on Distributed Computing Systems*. June 2012, pp. 586–595. DOI: 10.1109/icdcs.2012.75.

[74] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba. a LLVM-based Python JIT compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15*. Austin, Texas, 2015, 7:1–7:6. DOI: 10.1145/2833157.2833162.

[75] Jiwei Li, Kai Bu, Xuan Liu, and Bin Xiao. "Enda. embracing network inconsistency for dynamic application offloading in mobile cloud computing". In: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing - MCC '13*. 2013. DOI: 10.1145/2491266.2491274.

[76] Zhijing Li, Zhujun Xiao, Bolun Wang, Ben Y. Zhao, and Haitao Zheng. "Scaling Deep Learning Models for Spectrum Anomaly Detection". In: *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing - Mobihoc '19*. ACM. 2019, pp. 291–300. DOI: 10.1145/3323679.3326527.

[77] Ying-Dar Lin, Edward T.-H. Chu, Yuan-Cheng Lai, and Ting-Jun Huang. "Time-and-Energy-Aware Computation Offloading in Handheld Devices to Coprocessors and Clouds". In: *IEEE Systems Journal* 9.2 (June 2015), pp. 393–405. DOI: 10.1109/jsyst.2013.2289556.

[78] J. Marcum. "A statistical theory of target detection by pulsed radar". In: *IEEE Transactions on Information Theory* 6.2 (Apr. 1960), pp. 59–267. DOI: 10.1109/tit.1960.1057560.

[79] Richard Martin and Ryan Thomas. "Algorithms and bounds for estimating location, directionality, and environmental parameters of primary spectrum users". In: *IEEE Transactions on Wireless Communications* 8.11 (Nov. 2009), pp. 5692–5701. DOI: 10.1109/twc.2009.090494.

[80] Lime Microsystems. Accessed May 26, 2018.

[81] Peter Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. "Computer Generation of Hardware for Linear Digital Signal Processing Transforms". In: *ACM Transactions on Design Automation of Electronic Systems* 17.2 (Apr. 2012), pp. 1–33. DOI: 10.1145/2159542.2159547.

[82] Jeffrey C. Mogul and Ramana Rao Kompella. "Inferring the Network Latency Requirements of Cloud Tenants". In: *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. Kartause Ittingen, Switzerland, May 2015.

[83] *Monsoon Solutions*.

[84] MyriadRF. *Myriad-RF 1*. Accessed May 31, 2018.

[85] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical Programming* 14.1 (Dec. 1978), pp. 265–294. DOI: 10.1007/bf01588971.

[86] *Nexus – Google*. https://www.google.com/nexus/. Online; accessed 26 February 2016.

[87] Ana Nika, Zhijing Li, Yanzi Zhu, Yibo Zhu, Ben Y. Zhao, Xia Zhou, and Haitao Zheng. "Empirical Validation of Commodity Spectrum Monitoring". In: *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems - SenSys '16*. Stanford, CA, USA, 2016, pp. 96–108. DOI: 10.1145/2994551.2994557.

[88] *NSF Workshop on Spectrum Measurements Infrastructure*. 2016.

[89] *onHub – Google*. https://on.google.com/hub/. Online; accessed 26 February 2016.

[90] Jung-Min Park, Jeffrey H. Reed, A. A. Beex, T. Charles Clancy, Vireshwar Kumar, and Behnam Bahrak. "Security and Enforcement in Spectrum Sharing". In: *Proceedings of the IEEE* 102.3 (Mar. 2014), pp. 270–281. DOI: 10.1109/jproc.2014.2301972.

[91] Neal Patwari. *CRAWDAD The utah/CIR dataset (v. 2007-09-10)*. Downloaded from https://crawdad.org/utah/CIR/20070910/. Sept. 2007. DOI: 10.15783/C7630J.

[92] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.8 (Aug. 2005), pp. 1226–1238. DOI: 10.1109/tpami.2005.159.

[93] Phani Krishna Penumarthi, Alberto Quattrini Li, Jacopo Banfi, Nicola Basilico, Francesco Amigoni, Jason O'Kane, Ioannis Rekleitis, and Srihari Nelakuditi. "Multirobot exploration for building communication maps with prior from communication models". In: *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. Dec. 2017. DOI: 10.1109/mrs.2017.8250936.

[94] *Profiling with Traceview and dmtracedump*. http://developer.android.com/tools/debugging/debugging-tracing.html. [Online; accessed 05-August-2014].

[95] Matthias Pätzold. *Mobile Fading Channels*. John Wiley & Sons, Ltd, Jan. 2002. DOI: 10.1002/0470847808.

[96] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. "Odessa. enabling interactive perception applications on mobile devices". In: *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*. Bethesda, Maryland, USA, 2011, pp. 43–56. DOI: 10.1145/1999995.2000000.

[97] *Raspberry Pi 3 Model-B*. Accessed May 31, 2018.

[98] Hosam Rowaihy, Sharanya Eswaran, Matthew Johnson, Dinesh Verma, Amotz Bar-Noy, Theodore Brown, and Thomas La Porta. "A survey of sensor selection schemes in wireless sensor networks". In: *Unattended Ground, Sea, and Air Sensor Technologies and Applications IX*. Vol. 6562. International Society for Optics and Photonics. Apr. 2007, 65621A. DOI: 10.1117/12.723514.

[99] Rtl-Sdr. *Modellierung von digitalen Systemen mit SystemC*. Accessed August 13, 2018. Jan. 2012. DOI: 10.1524/9783486718959.fm.

[100] *Samsung GALAXY S3*. http://www.samsung.com/global/galaxys3/. Online; accessed 26 February 2016.

[101] Mahadev Satyanarayanan, Victor Bahl, Ramon Caceres, and Nigel Davies. "The Case for VM-based Cloudlets in Mobile Computing". In: *IEEE Pervasive Computing* 8.4 (Oct. 2011), pp. 14–23. DOI: 10.1109/mprv.2009.64.

[102] E Seneta. "On the history of the Strong Law of Large Numbers and Boole's inequality". In: *Historia Mathematica* 19.1 (Feb. 1992), pp. 24–39. DOI: 10.1016/0315-0860(92)90053-e.

[103] Manohar Shamaiah, Siddhartha Banerjee, and Haris Vikalo. "Greedy sensor selection: Leveraging submodularity". In: *49th IEEE Conference on Decision and Control (CDC)*. Dec. 2010, pp. 2572–2577. DOI: 10.1109/cdc.2010.5717225.

[104] Chao Shang, Fan Yang, Dexian Huang, and Wenxiang Lyu. "Data-driven soft sensor development based on deep learning technique". In: *Journal of Process Control* 24.3 (Mar. 2014), pp. 223–233. DOI: 10.1016/j.jprocont.2014.01.012.

[105] *Shanghai wants law on radio spectrum*. https://www.shine.cn/news/metro/1803061282/.

[106] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. "Cosmos. computation offloading as a service for mobile devices". In: *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing - MobiHoc '14*. Philadelphia, Pennsylvania, USA, 2014, pp. 287–296. DOI: 10.1145/2632951.2632958.

[107] *SONY Asia-Pacific*. http://www.sony-asia.com.

[108] *Specjvm2008*. https://www.spec.org/jvm2008/.

[109] S Srinu, Samrat L Sabat, and Siba K Udgata. "FPGA implementation of cooperative spectrum sensing for cognitive radio networks". In: *Cognitive Wireless Systems (UKIWCWS), 2010 Second UK-India-IDRC International Workshop on*. 2010, pp. 1–5.

[110] Wojciech Szpankowski. "Inclusion-Exclusion Principle". In: *Average Case Analysis of Algorithms on Sequences* (2001), pp. 49–72.

[111] Rahul Tandra and Anant Sahai. "SNR Walls for Signal Detection". In: *IEEE Journal of Selected Topics in Signal Processing* 2.1 (Feb. 2008), pp. 4–17. DOI: 10.1109/jstsp.2007.914879.

[112] *ThreadMXBean (Java SE 7)*. http://docs.oracle.com/javase/7/docs/api/.

[113] Jayakrishnan Unnikrishnan and Venugopal V. Veeravalli. "Cooperative Sensing for Primary Detection in Cognitive Radio". In: *IEEE Journal of Selected Topics in Signal Processing* 2.1 (Feb. 2008), pp. 18–27. DOI: 10.1109/jstsp.2007.914880.

[114] H. Urkowitz. "Energy detection of unknown deterministic signals". In: *Proceedings of the IEEE* 55.4 (1967), pp. 523–531. DOI: 10.1109/proc.1967.5573.

[115] Rahul Vaze and Chandra R. Murthy. "Multiple Transmitter Localization and Whitespace Identification Using Randomly Deployed Binary Sensors". In: *IEEE Transactions on Cognitive Communications and Networking* 2.4 (Dec. 2016), pp. 358–369. DOI: 10.1109/tccn.2016.2634000.

[116] Tim Verbelen, Tim Stevens, Filip De Turck, and Bart Dhoedt. "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing". In: *Future Generation Computer Systems* 29.2 (Feb. 2013), pp. 451–459. DOI: 10.1016/j.future.2012.07.003.

[117] Haiyang Wang, Ryan Shea, Xiaoqiang Ma, Feng Wang, and Jiangchuan Liu. "On Design and Performance of Cloud-Based Distributed Interactive Applications". In: *2014 IEEE 22nd International Conference on Network Protocols*. Oct. 2014, pp. 37–46. DOI: 10.1109/icnp.2014.25.

[118] Hanbiao Wang, Kung Yao, Greg Pottie, and Deborah Estrin. "Entropy-based sensor selection heuristic for target localization". In: *Proceedings of the third international symposium on Information processing in sensor networks - IPSN'04*. 2004. DOI: 10.1145/984622.984628.

[119] Weiwu Yan, Di Tang, and Yujun Lin. "A Data-Driven Soft Sensor Modeling Method Based on Deep Learning and its Application". In: *IEEE Transactions on Industrial Electronics* 64.5 (May 2017), pp. 4237–4245. DOI: 10.1109/tie.2016.2622668.

[120] Lei Yang, Jiannong Cao, Shaojie Tang, Di Han, and Neeraj Suri. "Run Time Application Repartitioning in Dynamic Mobile Cloud Environments". In: *IEEE Transactions on Cloud Computing* 4.3 (July 2016), pp. 336–348. DOI: 10.1109/tcc.2014.2358239.

[121] Seungjun Yang, Yongin Kwon, Yeongpil Cho, Hayoon Yi, Donghyun Kwon, Jonghee Youn, and Yunheung Paek. "Fast dynamic execution offloading for efficient mobile cloud computing". In: *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Mar. 2013, pp. 20–28. DOI: 10.1109/percom.2013.6526710.

[122] Ali Yassin, Youssef Nasser, Mariette Awad, Ahmed Al-Dubai, Ran Liu, Chau Yuen, Ronald Raulefs, and Elias Aboutanios. "Recent Advances in Indoor Localization: A Survey on Theoretical Approaches and Applications". In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), pp. 1327–1346. DOI: 10.1109/comst.2016.2632427.

[123] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. "A Survey of Indoor Localization Systems and Technologies". In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2568–2599. DOI: 10.1109/comst.2019.2911558.

[124] Tan Zhang, Ning Leng, and Suman Banerjee. "A vehicle-based measurement framework for enhancing whitespace spectrum databases". In: *Proceedings of the 20th annual international conference on Mobile computing and networking - MobiCom '14*. 2014. DOI: 10.1145/2639108.2639114.

[125] Tan Zhang, Ashish Patro, Ning Leng, and Suman Banerjee. "A Wireless Spectrum Analyzer in Your Pocket". In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications - HotMobile '15*. 2015. DOI: 10.1145/2699343.2699353.

[126] Weiwen Zhang, Yonggang Wen, Kyle Guan, Dan Kilper, Haiyun Luo, and Dapeng Oliver Wu. "Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel". In: *IEEE Transactions on Wireless Communications* 12.9 (Sept. 2013), pp. 4569–4581. DOI: 10.1109/twc.2013.072513.121842.

[127] Weiwen Zhang, Yonggang Wen, and D.O. Wu. "Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel". In: *IEEE Transactions on Wireless Communications* 14.1 (Jan. 2015), pp. 81–93. DOI: 10.1109/TWC.2014.2331051.

Appendix A

Proofs of Theorems in Chapter 3

A.1 Proof of Theorem 1

Let $\mathbf{T}$ be a given subset of sensors. For simplicity and without any loss of generality, let use assume that (i) the JPD for H_0 has a zero mean, and (ii) the variance of the JPDs for both H_0 and H_1 is same (=Σ). T hus, the JPD for H_0 is $N(0, \Sigma)$ and for H_1 is $N(\mathbf{p}, \Sigma)$, where $N(\mu, \sigma)$ is a normal distribution with mean μ and variance σ, p is the vector (one dimension for each sensor in $\mathbf{T}$) of means, and Σ is the covariance matrix. To prove the theorem, we will show the following for a given set of sensors $\mathbf{T}$:

1. $\mathrm{P}_{\mathrm{err}}(\mathbf{T}) = Q(\frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}})$, where $\mathbf{p}^T$ is the *transpose* of the p vector and $Q()$ is the tail function.

2. $\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) = 1 - \mathrm{P}_{\mathrm{err}}(\mathbf{T})$ is montone and submodular.

The theorem follows easily from the above, as it is well known that greedy algorithms for a monotone and submodular objective function yield a 63% approximation [85].

Expression for $\mathrm{P}_{\mathrm{err}}(\mathbf{T})$. We start with computing $\mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_0)$, i.e., the probability that `MAP` picks H_1 (i.e., $P(H_1|\mathbf{x}) > P(H_0|\mathbf{x})$) when the prevailing hypothesis in H_0, based on an observation vector $\mathbf{x}$ from $\mathbf{T}$. We easily get:

$$
\begin{aligned}
P(H_1|x) &= \frac{1}{\sqrt{2\pi\Sigma|}} \exp[-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T\Sigma^{-1}(\mathbf{x}-\mathbf{p})] \\
P(H_0|x) &= \frac{1}{\sqrt{2\pi|\Sigma|}} \exp[-\frac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}]
\end{aligned}
$$

Now, the expression $P(H_1|\mathbf{x}) > P(H_0|\mathbf{x})$ is equivalent to $P(H_1|\mathbf{x})/P(H_0|\mathbf{x}) > 1$ which simplifies to:

$$\mathbf{x}^T\Sigma^{-1}\mathbf{p} > \frac{1}{2}\mathbf{p}^T\Sigma^{-1}\mathbf{p}. \tag{A.1}$$

We are interested in computing the probability of above expression being true, given H_0. In essence, given H_0, we want to compute:

$$\begin{aligned} \mathrm{P}_{\mathrm{err}}(\mathbf{T}|H_0) &= P(\mathbf{x}^T\Sigma^{-1}\mathbf{p} > \frac{1}{2}\mathbf{p}^T\Sigma^{-1}\mathbf{p}) \\ &= P(\frac{\mathbf{x}^T\Sigma^{-1}\mathbf{p}}{\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}}} > \frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}}) \\ &= Q(\frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}}), \end{aligned}$$

where Q is the tail function of the standard Gaussian distribution. The last equation follows, since in H_0, the expression $\frac{\mathbf{x}^T\Sigma^{-1}\mathbf{p}}{\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}}}$ has a mean of 0 and a standard deviation of 1. Similarly, we can show that $\mathrm{P}_{\mathrm{err}}(\mathbb{T}|H_1) = Q(\frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}})$, since variance of the JPDs in H_0 and H_1 is same (Σ). Thus, $\mathrm{P}_{\mathrm{err}}(\mathbf{T}) = Q(\frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}})$, and the localization accuracy $\mathrm{O}_{\mathrm{acc}}(\mathbf{T})$ is:

$$\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) = 1 - \mathrm{P}_{\mathrm{err}}(\mathbf{T}) = 1 - Q(\frac{1}{2}\sqrt{\mathbf{p}^T\Sigma^{-1}\mathbf{p}}) \tag{A.2}$$

$\mathrm{O}_{\mathrm{acc}}$ () is Monotone and Submodular. First, we note that the value of $\mathbf{p}^T\Sigma^{-1}\mathbf{p}$ increases monotonically with "growth" (more dimensions) in $\mathbf{p}$. Now, since $Q(z)$ is a monotonically decreasing function, we get $\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) > \mathrm{O}_{\mathrm{acc}}(\mathbf{T} \cup \{s\})$. For submodularity, we note that $Q(z)$ is continuous, differentiable, with $\frac{d^2Q(z)}{dz^2} > 0, \forall z > 0$. Thus, the rate of reduction of $Q(z)$ reduces with an increase in z. Thus, $\mathrm{O}_{\mathrm{acc}}$ is submodular.

A.2 Counter-Example to Show that $\mathrm{O}_{\mathrm{acc}}$ is not Submodular in General Case

We have the following expression of $\mathrm{O}_{\mathrm{acc}}$:

$$\mathrm{O}_{\mathrm{acc}} = \sum_{i=0}^{m}\prod_{j\neq i}[1 - P(\mathtt{MAP}_{ij}(\mathbf{x}) \neq j|H_i)]P(H_i) \tag{A.3}$$

Note that since $\texttt{MAP}_{ij}$ is the test among the two hypotheses:

$$H_i : N(\mathbf{p}_i, \Sigma) \quad \& \quad H_j : N(\mathbf{p}_j, \Sigma) \tag{A.4}$$

We first shift the means of both hypotheses so that the mean of H_i is set to O. This does not affect the probability of misclassification, since both the means are equally shifted. Then, H_i and H_j have means of O and $\mathbf{p}_j - \mathbf{p}_i$ respectively. Now, using Lemma 2, we have the value of $\mathrm{O}_{\mathrm{acc}}$ as:

$$\mathrm{O}_{\mathrm{acc}} = \sum_{i=0}^{m} P(H_i)\big[\prod_{j \neq i}(1 - Q(\frac{1}{2}\sqrt{(\mathbf{p}_j - \mathbf{p}_i)^T \Sigma^{-1}(\mathbf{p}_j - \mathbf{p}_i)}))\big] \tag{A.5}$$

For convenience, we denote $Q(\frac{1}{2}\sqrt{(\mathbf{p}_j - \mathbf{p}_i)^T \Sigma^{-1}(\mathbf{p}_j - \mathbf{p}_i)})$ by Q_{ij}. We now show that $\mathrm{O}_{\mathrm{acc}}$ is not submodular using a counter-example. Let there be three hypothesis H_0, H_1 and H_2 with prior probabilities $P(H_i)$ each equal to 0.33 and two sensors with the mean vectors $[0, 0], [0.5, 0.5]$ and $[1, 1]$. Also assume that Σ is an identity matrix.

We first observe that when no sensors are selected, we select one among the three hypothesis at random, which will be correct only with an expected probability of 0.33, i.e., $\mathrm{O}_{\mathrm{acc}}(\{\}) = 0.33$. We note that since the two sensors have identical distributions, we can select the first sensor arbitrarily. Now, after selecting one sensor, and using the fact that each hypothesis has equal prior, we get the following value of $\mathrm{O}_{\mathrm{acc}}$:

$$\begin{aligned}\mathrm{O}_{\mathrm{acc}}(\{s_1\}) = 0.33[&(1 - Q_{12})(1 - Q_{01}) + (1 - Q_{12})(1 - Q_{13}) \\ &+ (1 - Q_{01})(1 - Q_{02})] = 0.3954\end{aligned}$$

Thus, the gain $G(s_1, \{\}) = \mathrm{O}_{\mathrm{acc}}(s_k) - \mathrm{O}_{\mathrm{acc}}(\{\}) = 0.0654$. Now, we compute the gain of adding the second sensor. Selecting both sensors, we get the value of $\mathrm{O}_{\mathrm{acc}}$ as:

$$\begin{aligned}\mathrm{O}_{\mathrm{acc}}(\{s_1, s_2\}) = 0.33[&(1 - Q(0.5))(1 - Q(0.5)) + (1 - Q(0.5)) \\ &(1 - Q(1)) + (1 - Q(1))(1 - Q(1))] = 0.58923\end{aligned}$$

Thus, the gain $G(s_2, \{s_1\}) = \mathrm{O}_{\mathrm{acc}}(\{s_1, s_2\}) - \mathrm{O}_{\mathrm{acc}}(\{s_1\}) = 0.19$. We observe that the gain has gone up from 0.0654 to 0.19 on adding the sensor s_1 to our set. Thus, the objective $\mathrm{O}_{\mathrm{acc}}$ is not submodular.

A.3 Proof of Lemma 1

We prove the lemma in three parts.

$\underline{\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) \leq \mathrm{O}_{\mathrm{acc}}(\mathbf{T})}$. This directly follows from an application of Boole's inequality [102] which states that the probability of a union of events is never greater than the sum of the probabilities of individual events. In particular, by Boole's inequality, we have for all i:

$$P(\bigcup_{j \neq i} \mathtt{MAP}_{ij} = j | H_i) \leq \sum_{j \neq i} P(\mathtt{MAP}_{ij} = j | H_i) \tag{A.6}$$

Then, by multiplying each by $P(H_i)$, summing over all i, subtracting each side from 1, and noting that $\sum_i P(H_i) = 1$, we get $\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) \leq \mathrm{O}_{\mathrm{acc}}(\mathbf{T})$ using Eq (3.8) and Eq (3.9).

$\underline{\mathrm{O}_{\mathrm{acc}}(\mathbf{T}) \leq 1 - \frac{1}{k}(1 - \mathrm{O}_{\mathrm{aux}}(\mathbf{T}))}$. To get this, we utilize the fact that the probability of a union of events is more than the probability of each of the individual events. Thus,

$$P(\bigcup_{j \neq i} \mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i) \geq \max_{j \neq i} \{P(\mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i)\} \quad \forall i.$$

We also have the below, as maximum is greater than mean:

$$\max_{j \neq i} \{P(\mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i)\} \geq \frac{1}{m} \sum_{j \neq i} P(\mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i) \quad \forall i,$$

where $0 \leq i \leq m$. Now, using Eq (3.8) and the above two equations, we get:

$$\begin{aligned} \mathrm{O}_{\mathrm{acc}}(\mathbf{T}) &\leq 1 - \frac{1}{m} \sum_{i=0}^{m} \sum_{j \neq i} P(\mathtt{MAP}_{ij}(\mathbf{x}) = j | H_i) P(H_i) \\ &= 1 - \frac{1}{m}(1 - \mathrm{O}_{\mathrm{aux}}(\mathbf{T})). \end{aligned}$$

The lemma now follows from the following fact, whose proof we omit for lack of space.

$$\frac{1 - \mathrm{O}_{\mathrm{aux}}(T')}{1 - \mathrm{O}_{\mathrm{acc}}(T')} \leq \frac{1 - \mathrm{O}_{\mathrm{aux}}(T)}{1 - \mathrm{O}_{\mathrm{acc}}(T)}, \quad \text{for any} \quad T' \supseteq T$$

A.4 Independent Sensor Observations

From Theorem 1's proof and notations therein, note that Eq (3.9) can be written as:

$$\mathrm{O}_{\mathrm{aux}}(\mathbf{T}) = 1 - \sum_i \sum_{j \neq i} Q((\mathbf{p}_j - \mathbf{p}_i)\Sigma^{-1}(\mathbf{p}_j - \mathbf{p}_i)^T)P(H_i), \tag{A.7}$$

where $Q(x)$ denotes the Marcum Q-function [78]. Now, suppose we wish to compute $\mathrm{O}_{\mathrm{aux}}(\mathbf{T} \cup \{s_k\})$ for a sensor s_k whose observations have a mean of p_{ki} for hypothesis H_i and a variance is σ_k^2. Let us denote the argument of $Q()$ in Eq (3.9) by $\mathbf{q}_{ij}(\mathbf{T})$. Then, we have the following recurrence relation:

$$\begin{aligned} \mathrm{O}_{\mathrm{aux}}(\mathbf{T} \cup \{s_k\}) = & \quad 1 - \sum_i \sum_{j \neq i} Q(\mathbf{q}_{ij}(\mathbf{T} \cup \{s_k\}))P(H_i) \\ = & \quad 1 - \sum_i \sum_{j \neq i} Q(\mathbf{q}_{ij}(\mathbf{T}) + \frac{p_{ki} - p_{kj}}{\sigma_k^2}) \end{aligned}$$

We note that computing $\mathbf{q}_{ij}(\mathbf{T})$ directly using Eq (A.7) takes $O(B^2)$ time. However, we can compute $\mathbf{p}_{ij}(\mathbf{T})$ incrementally by using the equation

$$\mathbf{q}_{ij}(\mathbf{T} \cup \{s_k\}) = \mathbf{q}_{ij}(\mathbf{T}) + \frac{p_{ki} - p_{kj}}{\sigma_k^2}$$

in constant time. As computing the Q-function takes constant time, the above reduced the time complexity by a factor of $O(B^2)$.

A.5 Proof of Theorem 2

Proof. Let $\mathbf{T}$ be AGA solution, and $\mathbf{T}$' be any solution. We have:

$$\begin{aligned} \mathrm{O}_{\mathrm{aux}}(\mathbf{T}) &\geq 0.63\mathrm{O}_{\mathrm{aux}}(\mathbf{T}') \\ (1 - \mathrm{O}_{\mathrm{aux}}(\mathbf{T}) &\leq 0.63(1 - \mathrm{O}_{\mathrm{aux}}(\mathbf{T}')) + 0.37 \\ (1 - \mathrm{O}_{\mathrm{acc}}(\mathbf{T}) &\leq 0.63k(1 - \mathrm{O}_{\mathrm{acc}}(\mathbf{T}') + 0.37 \\ P_{err}(\mathbf{T}) &\leq 0.63kP_{err}(\mathbf{T}') + 0.37 \end{aligned}$$

We have used Lemma 1 in the third equation above. Let $\mathbf{T}$' be the solution with optimal $\mathrm{O}_{\mathrm{acc}}$ () (and thus, optimal P_{err}), and the lemma follows. □