Trinity College Dublin
School of Computer Science and Statistics

Resource Management for Data Analytics in Edge Computing Networks

by
Apostolos Galanopoulos

Dissertation

Submitted in partial fullfilment of the requirements
for the degree of Doctor of Philosophy

September 2021

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Summary

This is a dissertation submitted for acquiring the degree of Doctor of Philosophy, from School of Computer Science and Statistics, Trinity College Dublin. It contains research carried out in the past 4 years, in the broad area of edge computing networks and optimization of machine learning and data analytic services that have emerged through the deployment of next generation wireless networks. Those services are unique, and thus, new problems arise that cannot be tackled by typical network resource orchestration solutions. In particular, the services do not involve deterministic tasks that can be executed anywhere in the network, i.e the devices themselves or an edge/cloud server, with the same level of accuracy. These are often machine learning tasks, like e.g. object recognition, and the deployment of related software on the various network location imposes intricate trade-offs between execution accuracy/delay, power consumption and even computing capabilities, which are typically diminishing as devices move towards the edges of the network.

The purpose of our research is to study and highlight those trade-offs and propose optimization based solutions to strike a balance between *application performance*, *end user satisfaction*, and *resource availability*. We mainly consider 2 network architectures. In the first one, small, energy constrained IoT devices need to collaboratively execute their data analytic tasks in order to economize network, computing and energy resources, while also ensuring a fast and accurate task execution. We start with a simple model where IoT nodes individually decide the portion of tasks they outsource to their neighboring nodes, using a Lagrange decomposition method, towards optimizing a multi-objective performance metric. We then build upon that model to provide an incentive mechanism for the collaboration of devices using an auction algorithm that enables devices to obscure sensitive information and discourage its misreporting in order to exploit the resources of others. We evaluate the solution using simulations as well as a small Raspberry Pi

testbed used also for obtaining power consumption and task execution accuracy measurements from a custom built face recognition application. Finally, we propose a framework for dynamic scheduling of transmissions and executions of such face recognition tasks using a combination of the approximate dual subgradient method and the Frank-Wolfe algorithm. The challenge here is that the problem we wish to solve is not linear, and the (primary) scheduling variables obtained in each iteration of the solution algorithm are not implementable to our system as, e.g. when a certain link is active, all interfering links should be inactive. Using Frank-Wolfe primal updates we overcome this issue and obtain a series of implementable schedules that, on average, approximate the system's optimal scheduling solution.

We then study centralized networks where a number of end users can leverage a powerful edge computing cloudlet for enhanced data analytics performance. Our initial setup has small devices execute an object recognition task and then using a predictor to estimate the prediction gain if the task is further outsourced to the cloudlet. The gain is weighted against the respective user and cloudlet resource consumption and each user decides whether to outsource the task or not. We use Convolutional Neural Networks (CNN) and K Nearest Neighbor (KNN) algorithm to evaluate the service and a linear regressor and random forest to implement the predictor. Finally, our iterative algorithm makes outsourcing decisions in each time slot, using only information available up to that time slot. Following that work, we study a mobile application that captures images, encodes them, and transmits them to a GPU enabled edge server that carries out object recognition using a state-of-the-art CNN. The results are transmitted back to the phone and displayed on its screen. We initially study the accuracy/latency trade-off of the system as we modify 2 of the service's critical parameters, namely the image encoding rate and Neural Network (NN) input layer size, and propose network related tweaks that effectively render object recognition, rather than image transmission to be the bottleneck of the system with respect to latency. We then proceed to collect measurements for the accuracy and latency performance of this system and propose an online algorithm for configuring the encoding rate and NN size on the fly. We make use of the Multi-Armed Bandit (MAB) framework and combine it with Gaussian Processes to dynamically estimate the performance of the system under different configurations and approach the optimal one. We further consider multiple users sharing the server's GPU and wireless air time of the system so that they all maximize their object recognition accuracy, while satisfying individual frame rate constraints.

Acknowledgments

This dissertation would never be possible without the aid of my supervisor, prof. George Iosifidis. He was very eager to have me as his student, and highly motivated to guide me throughout the years towards improving my knowledge and skills. His persistence and attention to detail has been invaluable; his encouragement and overall attitude during difficult times, hugely appreciated. He offered me unique experiences like the opportunity to assist in his teaching duties, and his urging to pursue an internship at a big tech company like IBM. For all that and many more, I will always consider him as a mentor and a friend.

There are also other people I collaborated with, and would like to thank. Prof. Doug Leith, the head of our lab, was always there to aid me and provide me with everything I needed for conducting research and generally feel comfortable in the lab. The discussions we had on our research topics were always deep and interesting, and the feedback he provided for our published work was always to the point and extremely valuable. The same is true for Dr. Salonidis, who aided me in my research during my first few years as a PhD researcher. Victor Valls spent many hours with me, working on both mathematical equations and lines of code. Finally, Jose Ayala-Romero joined the lab about 1.5 years ago, and within just a few months we had several publications together, thanks to his attention, knowledge and great ideas.

Then there are all those amazing people that have worked or are still working in our lab: Kariem, Hamid, Jeongho, Pavlos, Mayank, Cillian, Fahri, Daron, Sulthana, Sham, Beiran, Erjona. Some of you I know quite well. The rest I would like to have more time to get to know (I have the pandemic to thank for that). All of you however created a very pleasant atmosphere for me in the lab, which I enjoyed even during times I was under a huge load of work. Many thanks to the CONNECT Center team for the interesting discussions we had now and then, and especially to Prof. Marco Ruffini for technically having me as a PhD after George's departure

from the university earlier this year. Also, the entire administrative and academic staff of the school of Computer Science and Statistics was amazing, always there to respond to any arising issues and arranging and taking part to numerous seminars on several interesting research topics that augmented my curiosity and understanding of science.

Last but not least, I want to thank my fiance Dimitra both for the emotional support when things were getting difficult and I would lose courage, and for the times she celebrated my achievements with me. None of those would be possible if she wasn't always by my side. Finally, a big thank you to my family and friends back in Greece, for being highly supportive of me in my decision to study in Ireland. You are always on my mind, and whenever I get the chance to visit I always get excited and happy to see you.

List of Publications

1. A. Galanopoulos, G. Iosifidis, and T. Salonidis, "Optimizing Data Analytics in Energy Constrained IoT Networks", *IEEE International Symposium on Modeling and Optimization in Mobile, AdHoc and Wireless Networks (WiOpt)*, 2018.

2. A. Galanopoulos, G. Iosifidis, and T. Salonidis, "Poster: Cooperative Analytics for the Internet of Things", *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2019.

3. A. Galanopoulos, T. Salonidis, and G. Iosifidis, "Cooperative Edge Computing of Data Analytics for the Internet of Things", *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1166-1179, Dec. 2020.

4. A. Galanopoulos, V. Valls, D. J. Leith, and G. Iosifidis, "Dynamic Scheduling for IoT Analytics at the Edge", *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2020.

5. A. Galanopoulos, A. G. Tasiopoulos, G. Iosifidis, T. Salonidis, and D. Leith, "Improving IoT Analytics through Selective Edge Execution", *IEEE International Conference on Communications (ICC)*, 2020.

6. A. Galanopoulos, V. Valls, G. Iosifidis, and D. Leith, "Measurement-driven Analysis of an Edge-Assisted Object Recognition System", *IEEE International Conference on Communications (ICC)*, 2020.

7. A. Galanopoulos, J. Ayala-Romero, G. Iosifidis, and D. J. Leith, "Bayesian Online Learning for MEC Object Recognition Systems", *IEEE Global Communications Conference (GLOBECOM)*, 2020.

8. A. Galanopoulos, J. Ayala-Romero, D. J. Leith, and G. Iosifidis, "AutoML for Video Analytics with Edge Computing", *IEEE International Conference on Computer Communications (INFOCOM)*, 2021.

List of Abbreviations

MEC	Multi-access Edge Computing
IoT	Internet of Things
NUM	Network Utility Maximization
RPi	Raspberry Pi
DPP	Drift Plus Penalty
ADSM	Approximate Dual Subgradient Method
ML	Machine Learning
KNN	K Nearest Neighbors
NN	Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DL	Deep Learning
AP	Access Point
NIC	Network Interface Controller
MAB	Multi-Armed Bandits
GP	Gaussian Process
CC	Cumulative Confidence
UCB	Upper Confidence Bound

Contents

Abstract

The emergence of Multi-access Edge Computing (MEC) aspires to divert the aggregation of data and their computation away from the cloud, and closer to the end users. At the same time IoT and mobile devices create a plethora of data, that in many cases are ephemeral and can be consumed at the network's edge. Hence, they should not burden the core network with data transfers, and the cloud with computations. We study the case of executing data analytic services on such networks. Data analytics can be very demanding in terms of computation and energy requirements, which can limit the effectiveness of executing on small edge devices. On the other hand, they are also delay sensitive, and solely relying on the cloud for their execution is inadequate. More importantly however, they are usually non-deterministic tasks, and the selection of execution algorithm or Machine Learning (ML) model can further impact their overall performance.

We study such trade-offs that arise in edge computing networks. We formulate and solve problems that decide the efficient allocation of communication and computation resources, towards optimizing key performance metrics for the underlying services. Moreover we provide the necessary incentives for the cooperation of nodes. Next, we propose a scheduling algorithm that makes no assumptions about the generally unknown system parameters, and thus is able to adapt to the varying system dynamics. We then study mobile edge computing systems, where a central edge node (cloudlet) serves multiple users based on resource availability. Finally, we delve into the specifics of a typical data analytic task, i.e. real time object recognition, and study the impact of application and network specific parameters to highlight various system trade-offs. We then proceed to optimally tune the service in an online fashion, paving the way towards automatic deployment of machine learning services over edge computing networks.

Chapter 1

Introduction

1.1 Motivation and Contributions

The Internet of Things (IoT) enables a new breed of applications by connecting a massive number of typically resource-constrained nodes at the network edge [1]. Among those, of particular interest are the *data analytic services* that collect and analyze data for making inferences or actuations, and assisting user decisions often in real time. Prominent examples include IoT cameras that perform face recognition [2], mobile health wearables [3], and natural language processing applications [4] among many others; see overview of use-cases in [5]. Indeed, the domain of data analytics has been identified as one of the main driving forces for the next generation of IoT systems, and it is expected that a plethora of such services will be deployed at an increasing pace in the next few years [6].

These services are as challenging to implement as they are important. On one hand, they create a huge amount of data that is very costly to transport to distant cloud servers where the analytics can be effectively executed. Large portions of this collected data is ephemeral with negligible payload and should ideally be filtered at the edge [7,8]. On the other hand, these services require heavy-duty computing, and most often prior information that might require preprocessing and significant memory space (e.g., a large training dataset is needed for face recognition apps). Hence, implementing them at the IoT nodes, which are usually energy-constrained with limited computation capabilities, might induce significant performance degradation, e.g., reduced accuracy, higher error rate, and so on (see Fig. 1.1). One possible solution to that problem is to enable the collaboration of IoT devices and exploit their heterogeneity to improve the

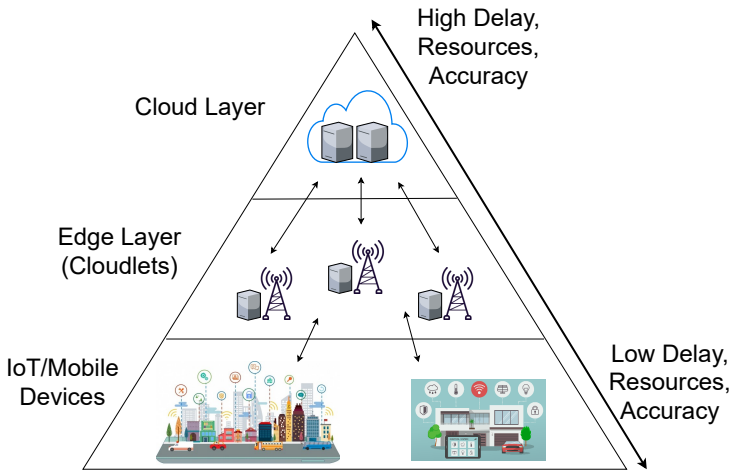


Figure 1.1: Layered structured of devices, edge/cloud servers and the relative trade-offs.

performance of analytics, while adhering to their individual resource consumption constraints. For example, an IoT camera that performs face recognition, could transmit some of its captured pictures to another camera in range, in order to expedite the task execution or improve its accuracy (if the other camera has, say, a larger training dataset). This approach comes with several advantages: it is scalable, reduces data transfer costs and task completion times, and satisfies privacy criteria since the data is not transferred to the cloud.

Moreover, some parameters of the data analytics tasks like their arrival rates are usually unknown, since they generally depend on application parameters. At the same time we need to provide scheduling decisions that are aligned with the system's hard interference constraints, and can adapt to the unknown system parameters. For example, for two interfering links that have been allocated with some data rates, we have to decide the schedules in each time slot so that the links will never be active at the same time and the allocated rates are satisfied on average. Many scheduling algorithms for wireless networks in the literature require a-priori knowledge of the set of valid schedules [9–12], which is rarely the case for such dynamic systems.

Such scheduling interventions are of critical importance, especially for applications like video analytics, that often require real-time performance. Mobile applications, e.g. augmented reality [13], take advantage of modern high speed wireless networks to utilize specialized hardware like GPUs, installed on edge computing servers. On one hand, this approach enables real-time performance, but on the other hand, it creates a whole new genre of resource allocation problems, since many of the parameters that need to be configured are not typical networking parameters,

but involve service specific decisions like the video frame rate and resolution, or the machine learning model to be used by the edge server for, e.g., object recognition. On top of that, the accuracy/delay performance of such services is rarely known a-priori for fixed parameter settings. They can vary with time, since e.g. wireless channels do not offer stable performance, or even with device software and hardware as for example, a GPU's processing speed can affect Deep Learning (DL) processing delays. Clearly, optimizing the performance of such unique services by continuously learning their initially unknown behavior is of immense importance.

The purpose of our research is to study the complex nature of edge computing systems and provide solutions for the orchestration of communication and computation resources using optimization techniques. We model the systems by deriving formulas that describe their major performance metrics and solve optimization problems that yield the desired operation for each of the devices towards optimizing their performance. We study the trade-offs that arise, and how the system parameters can be tweaked towards obtaining the desired performance. We highlight their characteristics, and validate our solutions using extensive simulations and experiments. In specific, the contributions of this thesis are summarized as follows:

- **Problem formulations.** The particularity of data analytic services gives birth to new interesting problems that extend classic Network Utility Maximization (NUM). In specific, the execution accuracy and delay of such services often need to be jointly optimized, or at least optimize one while respecting a constraint for the other. Moreover, we have power and computing resource constraints that create multiple trade-off situations when we want to decide the services' execution location or other service related parameters. Our research aims to study such problems and highlight those trade-offs between key service and networking metrics.

- **Optimization based algorithms.** In order to solve those problems, we employ several optimization techniques like Lagrange decomposition, and propose efficient algorithms for making key system decisions in a way that is optimal and fair across the different devices/users of such systems. We use frameworks such as the Approximate Dual Subgradient Method (ADSM) and Multi-Armed Bandits (MAB) to counter realistic scenarios where parts of the system's parameters follow unknown, even non-i.i.d. distributions, and provide theoretical proofs on the optimality of the proposed algorithms. Our goal is to develop solutions that

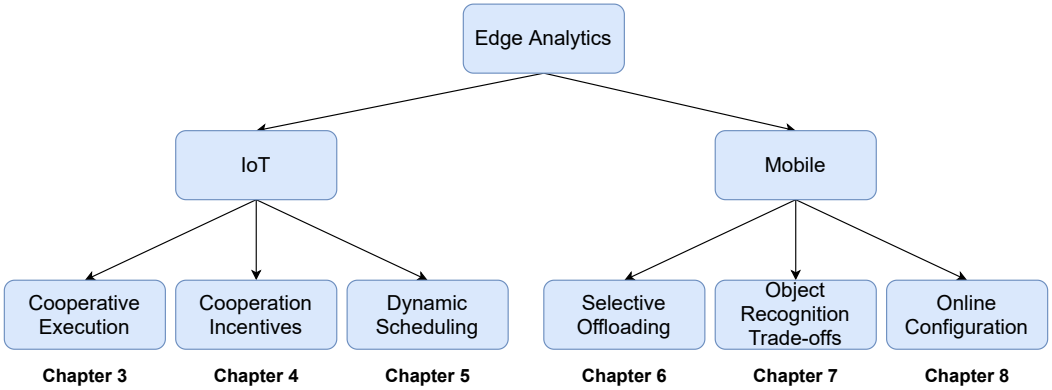


Figure 1.2: Categorization of thesis' chapters.

can configure such complex systems on the fly, and with minimal a-priori information, which is often unavailable in practice.

- **Experimentally validated solutions.** We evaluate the efficiency of the proposed algorithms using simulations, but more importantly, using testbeds consisting of state-of-the-art equipment like GPU-enabled servers, Raspberry Pi devices and mobile phones. This way we further validate the results of our research in a more tangible way, and even create datasets from real systems that can help other researchers study similar problems and expand scientific knowledge.

1.2 Synopsis

The structure of the thesis is such that it presents 1-2 published papers per chapter. This decision was taken because some papers are straightforward extensions of previous works, e.g. conference paper extended to journal paper. The main chapters are organized in a hierarchy displayed in Fig. 1.2. IoT and collaborative networks are studied in Chapters 3-6, starting from simpler static models, to more practical and realistic scenarios where system parameters are unknown. Then our attention is focused on mobile, edge-assisted applications, retaining the realistic assumptions made previously, and examining custom built edge systems before proposing online methods for their optimal configuration.

In Chapter 2 we discuss the most recent literature on edge computing and data analytics for IoT and mobile networks. Following the structure of the remainder of the thesis, we firstly discuss in detail works studying edge computing problems where low computing power edge

devices execute (collaboratively) data analytic tasks, based on input data generated by them. Next, we expand in scenarios where more powerful edge computing servers are used to outsource the edge devices' load, in order to handle more delay/accuracy sensitive applications. We further discuss additional works in other chapters when they are necessary and more relevant to the specifics of the chapter.

The model of Chapter 3 assumes an architecture where devices are willing to collaboratively[1] execute data analytic tasks in order to improve their execution delay and accuracy. We design a static optimization framework where the nodes decide where their data analytic tasks are going to be executed, in order to jointly optimize their average execution delay and accuracy, while respecting power consumption constraints. We assume that several system parameters like link capacities and task arrival rates are known a-priori, and propose a distributed dual ascent solution to the formulated convex problem, so that the nodes can make the outsourcing decisions by exchanging local information. This is enabled by introducing an auxiliary variable used to control the amount of tasks each node admitts for computation and essentially decoupling the problem's constraints. The results indicate that the nodes can achieve better performance when collaborating than when they locally compute the tasks, depending on the network load.

In Chapter 4 we drop the assumption that devices are willing to cooperate. We propose a framework that provides the necessary incentives for the collaborative execution of data analytic tasks between the IoT nodes. In detail, we introduce a cooperative auction-based algorithm which optimizes the aggregate execution accuracy and delay, without requiring information about the nodes' accuracy/delay priorities, nor their resource availability. This is possible because nodes bid for accessing the resources of others and request compensation for admitting their tasks, and thus, do not need to reveal private information regarding the aforementioned priorities and resource availability. The algorithm yields the optimal task - node assignment, as well as the necessary reimbursements for ensuring the devices' full-scale cooperation. Our framework is then extended to multi-stage analytics, where different parts of the task execution can take place to different nodes, further improving the system's flexibility and performance.

Next, in Chapter 5, we further drop the static modeling used before, by assuming unknown and time varying system parameters. We study the problem of making fast scheduling decisions on IoT networks that request the execution of data analytics. The objective is to decide which

[1]E.g. the devices belong to the same entity and thus the entire system can benefit from cooperation.

of the nodes, that execute the analytics in different rates and precision levels, will process each data stream. We propose an online dual subgradient method to deal with unknown system parameters. We observe that simple dual method solutions are not trivially implementable to our system since they violate interference constraints. Contrary to other similar works, [14, 15], we propose an algorithm based on the Frank-Wolfe algorithm [16] that updates the primal variables in a way that always produces feasible schedules. Furthermore, it has low complexity per iteration, and also adapts to system changes.

In Chapter 6, we shift our focus on mobile, edge-assisted analytics by proposing and evaluating a rigorous task outsourcing framework. The edge devices execute their tasks locally and further outsource them to edge servers (a.k.a. cloudlets) only when they predict a significant performance improvement. We consider the practical scenario where system parameters such as the requests and cloudlet computing capacities are unknown and vary stochastically over time. We propose an online optimization algorithm that is oblivious to this information and provably maximizes the analytics performance. Our approach relies on Nedic's exemplar work on the approximate dual subgradient method [17], combined with a primal-averaging scheme, and works under minimal assumptions about the system stochasticity. We fully implement the proposed algorithm in a wireless testbed and evaluate its performance using a state-of-the-art image recognition application. Our analysis demonstrates the capability of our algorithm to intelligently leverage the cloudlets, adapting to the service requirements and task load variations.

After observing the capabilities of edge systems in providing fast and accurate analytics, we develop a mobile, edge-assisted object recognition system with the aim of studying the system level trade-offs between end-to-end latency and object recognition accuracy. We focus on developing techniques that optimize the transmission delay of the system and demonstrate the effect of image encoding rate and neural network size on these two performance metrics. We explore optimal trade-offs between these metrics by measuring the performance of our real time object recognition application. Our measurements reveal hitherto unknown parameter effects and sharp trade-offs, hence paving the road for optimizing this key service. Finally, we formulate two optimization problems using our measurement-based models and following a Pareto analysis we find that careful tuning of the system operation yields at least 33% better performance for real time conditions, over the standard transmission method.

Finally in Chapter 8, we consider the edge-assisted object recognition system analyzed in

the previous chapter, and provide a solution for its optimal online configuration, as opposed to a static trade-off analysis. In detail, the system configurable parameters are distributed among the different user, network and edge devices, constituting their joint selection an intricate problem. We propose an Automated Machine Learning (AutoML) framework for jointly configuring the service and wireless network parameters, towards maximizing the analytics' accuracy subject to minimum frame rate constraints. Our experiments reveal the volatile and system/data-dependent performance of the service, and motivate the development of a Bayesian online learning algorithm which optimizes on-the-fly the service's performance. Similar online solutions have been proposed, e.g. [18, 19] but here we employ a method that does not assume a fixed model describing the system's performance, but rather learn it over time. We prove that our solution is guaranteed to find a near-optimal configuration using *safe* exploration, i.e., without ever violating the set frame rate thresholds. We further evaluate this AutoML framework in a variety of scenarios, using real datasets.

Chapter 9 presents a summary of conclusions for the thesis as well as research directions for future work. Finally Chapter 10 contains mathematical proofs for lemmas and theorems of Chapters 5 and 6 that are too extensive to appear in the main body of the respective chapters.

Chapter 2

Literature Review

Edge Computing has introduced a paradigm shift regarding the conventional way of executing data analytics. Computation resources have moved closer to the network edges and they can be used to handle the intensive computations in coordination with the standard cloud services [5, 20–23]. In [20] the authors present a framework for collaborative cloud/edge computing, and some of the challenges that arise towards supporting a broader spectrum of applications with such architectures. [21] classifies the major types of data analytics as classification, prediction, clustering and association rule mining, and presents some of the use cases for Big IoT data analytics, e.g. smart transportation/grid/agriculture/traffic lights, and healthcare. Other examples are smart city applications, security surveillance and smart manufacturing [22]. In [23] the authors introduced a system for smart city data analytics. The architecture entails multiple tiers of computing to provide the necessary resources and Quality of Service (QoS) for a series of different applications.

The surveys [24, 25] demonstrate the basic ideas behind the modeling and architecture of edge computing with computation offloading. They reveal the main performance metrics that need to be optimized in such systems to be execution latency, edge device power consumption and other utilities and costs that are related to them, or the executed tasks, e.g. task success rate. [26] proposes a load balancing scheme for the distributed execution of analytics with the objective of minimizing their delay. Regarding cellular networks, the works [27–30] use optimization techniques to allocate radio and computing resources to the network's devices, towards optimizing the network revenue [27], latency [28] and the users' energy consumption [29, 30].

Another critical performance metric regarding data analytics is their execution accuracy.

The work in [31] demonstrates a smart camera IoT device, capable of executing classification algorithms. Depending on the desired level of classification accuracy, the device is able to select the minimum energy operation point. The work in [32] studies the trade-off between data collection latency and execution accuracy. Finally, ApproxIoT [33] proposes a framework for executing data analytics using approximate computations. This way it is able to sacrifice accuracy of analytics for computation efficiency. All the above works indicate that in order to provide higher accuracy data analytics in edge computing networks, it is unavoidable to concede higher latency/power consumption.

2.1 IoT Analytics and Collaborative Networks

Edge IoT analytics are attracting growing interest. This idea extends in many ways previous schemes such as data fusion in sensor networks and mobile crowdsourcing/sensing applications [34–36]. Interestingly, there are already products, deployed or under testing, pointing to the direction of cooperative edge analytics. For instance, Google Nest cameras support video analytics [37], Microsoft offers a suite for edge analytics [38], which is similar to Amazon's Greengrass [39], while IBM developed project owl [40], and the Edgeware platform [41]. These services however do not optimize the collaborative task execution and ignore the issue of incentives. Yet, given the scale of IoT deployments and the nodes' resource constraints, ensuring cooperation is instrumental for the success of these systems.

Recent works emphasize the importance of executing analytics in edge/cloud infrastructures, e.g. [23, 27]. Most of them perform task offloading to optimize execution delay [20, 42, 43]. Misco [44], and CWC [45] implement frameworks for parallel task execution on phones, catering to MapReduce-like batch processing models rather than tasks modeled as data streams. Several cooperative models have been proposed in the literature for the execution of intensive computations between nearby edge devices. MobiStreams [46], Swing [47] and [48] enable collaborative computations on data streams. Nevertheless, these systems either do not optimize the task distribution [46], or use heuristic policies that do not cater for energy consumption or execution accuracy [47] [48]. DeepCham [49] is an adaptive mobile object recognition framework that uses deep learning techniques. It trains a collaborative learning model using a number of mobile users to produce more accurate recognition results. The authors in [50] propose a framework

that allows mobile devices to extract features from surveillance cameras for video processing applications. Serendipity [51] focuses on distributed computing in mobile networks with intermittent connectivity, and proposes a greedy task allocation scheme to minimize completion time. However, it does not take into account accuracy or energy constraints, and does not provide optimality guarantees.

Energy constraints in IoT systems are very important and several prior works have proposed solutions for reducing their power consumption. D2D Fogging [52] and [53] designed frameworks for Device-to-Device cooperation towards energy efficient task execution. In [54], the authors propose a dynamic algorithm that minimizes energy costs by leveraging green energy sources; and similarly, [55] and [56] propose cooperative solutions that reduce energy consumption. However, those prior works do not consider data analytics nor minimize the task execution delay. On the other hand, [26] decides the task offloading based on delay only, but does not account for the power consumption or the execution accuracy, which perhaps is the most important factor for such services. Finally, the authors in [55] propose a resource allocation scheme for cooperative computation between nodes of a wireless sensor network, towards minimizing the total energy consumption.

2.2 Mobile Analytics with Edge Computing

2.2.1 Task Execution

Most solutions partition compute-intense mobile applications and offload them to cloud [57, 58]. This approach cannot support applications with stringent requirements due to possible large delays in data transfers [59]. Cloudlets on the other hand, achieve lower delay by leveraging edge computing [23, 60] but have limited serving capacity. The execution of mobile analytics, either on local devices or on edge computing nodes has been extensively studied. For example, [61] proposes a solution for deciding the execution location of augmented reality tasks, either on the mobile, or an edge server. Previous works in this area consider simple performance criteria, such as reducing computation loads, and focus on the architecture design, e.g. Misco [44], CWC [45].

The increasing importance of these services has motivated the design of mobile and wireless systems that can execute such tasks. For instance, [62–64] tailor deep neural networks for execution in mobile devices. These works however, focus on low delay execution of analytics and

do not consider improving their execution accuracy. Cachier [65] uses edge servers as caches for image recognition requests sent to the cloud, aiming to minimize latency; while [66] optimizes again delay but through the orchestration of the edge resources. Precog [67] prefetches trained classifiers on devices which use these lightweight models to accelerate image recognition. In a different approach, [68] selects in runtime the ML models, namely the DNN size, in order to balance accuracy and resource consumption.

Prior analytical works in the context of computation offloading focus on different metrics, such as the number of served requests, e.g., see [69] and overview in [25]. Other works either rely on heuristics or static models and complete knowledge of system parameters [18, 68, 70]. Clearly, these assumptions are invalid for many practical systems where the performance metrics and system parameters not only vary with time, but often do not follow an i.i.d. (or Markov) process. This renders the application of max-weight type of policies [71] inefficient. Despite the efforts to improve the execution of analytics at small devices, e.g., by compressing NN models [72] or using residual learning [73], the trade-off between low-accuracy local and high-accuracy cloudlet execution is still important due to the increasing number and complexity of these tasks. This observation has spurred efforts for designing fast multi-tier (cloud to edge) deep neural networks [74]; for dynamic model selection [68, 75]; and for threshold-based task allocation to DNNs [76].

2.2.2 Video Analytics

Video analytics in particular, often employ MEC to improve scalability and latency. For instance, [74] and [76] explore DNN partitioning across user devices, edge servers and the cloud. In [77] the authors propose a framework for distributing deep learning sub-processes to edge, cloudlet and cloud nodes towards increasing the job execution rate of the system. Other object recognition systems like JAGUAR [78], Glimpse [79], and [80] use methods such as object tracking to improve real-time performance. OverLay [81] is a mobile augmented reality system, aided by GPU-enabled edge servers, to minimize the tracking error. Focus [82] uses a low latency / low accuracy first stage CNN for real time analytics, followed by a more robust one for queries that do not satisfy accuracy requirements. VideoStorm [83] schedules GPU resources to maximize an accuracy/latency utility and presents a system for large scale DL analytics that uses an offline profiler to evaluate the system's configurations, and an online scheduler to allocate GPU

resources towards maximizing a joint accuracy/latency utility, while Chameleon [19] decides the NN model, frame rate, and image resolution to maximize accuracy. A convex program is solved in [84] to decide frame sizes, and [85] formulates a similar integer decision problem; while [70] minimizes latency. Another large corpus of works take a computation-offloading perspective and handle computing or network resources without considering video analytics metrics (e.g., accuracy) or NN configurations (e.g. NN size); see [86–88].

Other works explore the accuracy/latency trade-off. JALAD [89] proposes the decoupling of a Deep Neural Network (DNN) between edge and cloud towards minimizing latency with execution accuracy guarantees. MobiQoR [90] studies the trade-off between delay and Quality of Result for edge applications that involve machine learning and analytics like face recognition. The authors show that sacrificing computation result quality can decrease delay as well as energy consumption. LAVEA [70] proposes a system for computation offloading of data analytics to nearby edge nodes. The formulated optimization problem aims in making offloading and bandwidth allocation decisions towards minimizing latency. DeepDecision [18] is a video analytics system that balances accuracy and latency, by properly adjusting the camera sample rate, video encoding rate, and deep learning model. However, both transmission and processing delays are much higher than the ones obtained by our system. DeepMon [91] distributes the execution of a large DNN across multiple mobile GPUs to reduce latency. It focuses on DNN optimizations, instead of the network-centric approach presented in our work.

All the above works, highlight the inherent trade-off between latency and accuracy in edge architectures. However, most of those works assume that the accuracy and delay of the available DL models and wireless links are known a-priori. In practice however, quite often we do not have access to complete datasets, and even then, their validity is limited as they presume a stationary environment. Hence, online learning solutions are better suited for such systems. Most works applying online learning to MEC systems are focused on making offloading decisions, e.g. [92]. Recently, online learning has been applied for configuring MEC systems for real-time operation, without however considering accuracy [93]. In [94] a Lyapunov framework is used to configure the image resolution and frame rate of video analytics applications in an online fashion. The objective is to optimize the accuracy/energy consumption trade-off, under latency constraints. In [95] video quality and computing resources are selected to maximize the (approximate) accuracy, without however considering the frame rate. Chameleon [19] profiles periodically the

configurations and searches greedily for the most resource-prudent, but does not consider latency. VideoEdge [96] solves a similar problem for hierarchical systems, while [97] solves an integeer program to allocate computing resources and decide the image compression and DNN model. In [98], video quality and computing resources are selected to maximize successful queries. These interesting works either do not offer optimality bounds [19, 97] or consider asymptotic-only performance [94], assume known models [94, 97, 98] or convex functions [98]. Finally, online algorithms for *general* edge computing, e.g., [99–101], do not cater for video analytic metrics or the specifics of video pipelines.

Chapter 3

Optimizing Data Analytics in Energy Constrained IoT Networks

3.1 Introduction

Data analytics services usually generate large volumes of data that are difficult to process in small IoT devices. A strong candidate solution for this problem is to leverage resources that are available at the edge, e.g., at nearby IoT devices or even at small edge servers, a concept also known as fog computing [7,8]. For example, an IoT camera that performs face recognition, could transmit some of its captured pictures to another camera in range, in order to expedite the task execution or improve its accuracy (if the other camera has, say, a larger training dataset). This approach comes with several advantages: it is scalable, reduces data transfer costs and task completion times, satisfies privacy criteria since the data is not transferred to the cloud, and so on.

Nevertheless, enabling a group of resource-limited IoT devices to jointly execute their analytic tasks is an intricate goal [20]. Some devices might be able to execute certain functions with higher accuracy, but might not have enough computation resources to support all nearby devices. It is therefore difficult to decide where to execute the tasks, even more since their execution is often delay-sensitive. Clearly, there is here an inherent trade off between accuracy and task execution delay. On top of that, the IoT nodes usually have limited communication capabilities, e.g., small transmission range, and tight energy constraints (average power limitations). Hence, they need to wisely decide how to allocate their power budget among computations, data collection and

transmissions.

Our goal in this chapter is to address the above challenges by *designing a framework for the collaboration of energy-constrained IoT devices which jointly optimize the execution delay and accuracy of their analytic tasks.*

In this chapter, we introduce an optimization framework for devising a data analytic task execution policy in an energy-constrained IoT environment. The framework allows the policy designer to balance execution time and accuracy, based on the system's priorities (or, mission). Our model is detailed: it captures transmission and reception energy costs that might vary across nodes; caters for the inherent heterogeneity in link capacities; and uses a generic model for the task execution accuracy that captures a rich set of data analytic scenarios. We place emphasis on delay, which is apparently very crucial for this problem, and use a load-dependent delay metric, instead of average constant delay functions.

In order to solve the resulting challenging mathematical program, we rely on Lagrange duality which effectively decomposes it to a set of subproblems (one per node). Furthermore, we use a primal-dual algorithm to allow the coordination of these subproblems in a distributed and scalable fashion. Indeed, we prove that our solution finds the optimal cooperation policy with minimal exchange of messages, involving only one-hop neighbors. Finally, we conduct a trace-driven evaluation of our solution, using measurements from a small testbed of Raspberry Pi 3 devices which run a Java-based face recognition app. Our experimental analysis reveals the potential benefits of such cooperative IoT solutions, and provides interesting insights for the arising trade offs. The contributions of this work can be summarized as follows:

- We introduce the problem of jointly optimizing the average execution delay and accuracy of data analytic tasks in IoT networks. Our model is generic in the sense that it captures a variety of network parameters.
- Our analytical multi-objective framework enables the designer to prioritize one objective over the other, based on the system's requirements.
- We design a scalable algorithm for solving the problem in a distributed fashion. Our algorithm requires minimal exchange of messages among only neighboring nodes to converge to the optimal solution.
- We conduct an extensive trace-driven evaluation of our optimization framework, showing that

the collaborative solution can be up to 60% faster and 23% more accurate than local task execution.

3.2 System Model and Problem Statement

3.2.1 System Model

Network. We consider a wireless IoT network represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ of N nodes and L links. The nodes are IoT devices of low resource footprint. We assume they are equipped with half-duplex radios, hence each node can transmit to, or receive from at most one other node at each time instance. They also operate with existing IoT technologies such as Wi-Fi, Bluetooth or ZigBee. Each link $l_{i,j} \in L$, if node j is within communication range of node i. All nodes within range of node i are called its neighbors and are denoted by $\mathcal{N}_{(i)}$, which contains N_i nodes. Each link $l_{i,j}$ is characterized by its average capacity $c_{i,j}$ measured in bits/s. We assume that $c_{i,j}$ is measured by higher layer capacity estimation mechanisms [102] and accounts for channel fluctuations on the link, and MAC layer scheduling[1].

Each link $l_{i,j}$ is also characterized by its average (Tx/Rx) energy consumption $e^t_{i,j}(c_{i,j}), e^r_{i,j}(c_{i,j})$ measured in Joules/bit. We assume that energy consumption is dominated by the transmission at node i over (the attenuated) reception at node j. In general, energy consumption depends on the link capacity $c_{i,j}$ since it is affected by the channel quality and the transmitter's bit rate [103]. We assume that the nodes experience a certain average bit rate with each of their neighbors and, for simplicity, use $e^t_{i,j}$ and $e^r_{i,j}$ to denote the resulting energy consumption. We consider the general case where $c_{i,j} \neq c_{j,i}$, $e^t_{i,j} \neq e^t_{j,i}$, $e^r_{i,j} \neq e^r_{j,i}$ since nodes i and j may have different wireless adapters.

Computing and Data Analytics. We assume that $\mathcal{N}_T \subseteq \mathcal{N}$ is a subset of the network's nodes that need to execute computationally intensive data analytic tasks. The tasks may correspond to two types of compute models. In a data stream processing model the tasks are data (e.g. images) that are sent to operator function modules (e.g. face recognition) deployed at the nodes. Alternatively, in a SPARK/MapReduce model the tasks may consist of operators that are sent to process data deployed at the nodes.

[1]Our focus is on data analytic task scheduling techniques at higher layers that are MAC agnostic. Optimizing MAC scheduling for IoT systems is an orthogonal problem that is out of the scope of this work.

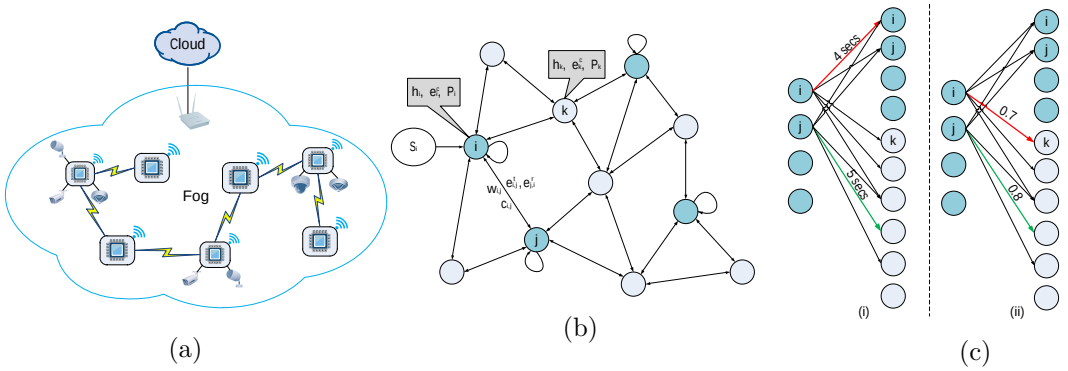


Figure 3.1: (a) The IoT environment where devices have embedded different types of sensors. (b) Example of network graph where the nodes with data analytic task arrivals are highlighted with blue color. (c) Optimal offloading decisions for accuracy and delay may differ.

Each node $i \in \mathcal{N}_T$ generates tasks for processing at a rate of λ_i tasks/sec. The tasks generated by each node i, are characterized by the amount of average input data S_i in bits[2]. Each node $j \in \mathcal{N}$ requires a number of ρ_j cycles to execute each task, depending on the algorithm it uses, based on its CPU frequency h_j. For example, in a face recognition application, node i is a camera that captures images of size S_i with an average rate of λ_i images/sec, and computing node j uses a specific recognition algorithm that yields a processing capacity of $\frac{h_j}{\rho_j}$ tasks/sec. Combining the task arrival rate with its input size, we define each node's data generation rate $R_i = S_i\lambda_i$ (b/s). Furthermore, each node j has an average power budget of P_j Watts, which can be used for processing or data transfers. Such power budgets arise in IoT networks that employ energy harvesting mechanisms [56] or comprise small form-factor power-constrained devices. In addition, the nodes have energy requirement when performing computations e_j^c (Joules/cycle). An annotated view of the model is depicted in Fig. 3.1b.

Each data analytic task has a metric that determines the quality of its outcome. Here, we focus on machine learning and predictive analytic tasks where the metric is the accuracy of the prediction. The output accuracy of the data analytic tasks depends on the executing node. We define by $w_{i,j} \in [0,1], \forall i \in \mathcal{N}_T,\ j \in \mathcal{N}_{(i)}$ the normalized output accuracy of node j when it executes a data analytic task requested by i. Consequently, when node i executes locally its own tasks, the execution accuracy is denoted by $w_{i,i}, \forall i \in \mathcal{N}_T$. Thus the notation $w_{i,j}$ is used to implicitly express j's efficiency to execute i's tasks, and not j's efficiency in general. In a face

[2]We assume that task properties are node-specific, e.g., in a face recognition application it is the camera configuration of each node that determines the size of the captured frames. However, our model can be directly extended to include a set of task classes, each one with different data sizes, computation load, and arrival rate.

recognition application for example, the nodes can use different classifiers or different training data sets [104], which will yield different execution accuracy in the classification task. Note that our model is very generic as it allows performance to depend on the node that generated the tasks, e.g. the image resolution or the surrounding environment of the capturing camera-node may impact the accuracy of the classification.

3.2.2 Problem Statement

We design our system with the aim to devise a data analytic policy which will: *a)* minimize the overall execution delay, and *b)* maximize each node's average task execution accuracy. To this end, we formulate a rigorous mathematical program with a multi-objective optimization metric, that respects power budget, link and computation capacities in the IoT network. The solution of this problem gives us the portion of tasks that the nodes will outsource to each of their neighbors. This is an interesting problem where several trade offs arise.

The reasons for improving performance with outsourcing are multiple. The tasks arriving at the nodes can be too demanding with respect to required computational power to be supported by their power budgets. In addition, these neighboring nodes may be more specialized in executing these tasks, so outsourcing can increase execution accuracy. The additional communication delay when choosing to execute at another node can be large, however there can be some neighboring nodes with large link capacity and powerful processing speed that can render outsourcing faster than local-only execution.

It is interesting to note that minimizing the execution delay and maximizing execution accuracy are *not* necessarily conflicting objectives. In some cases minimizing delays by routing tasks to good channel quality neighbors can lead to maximizing execution accuracy, namely if the selected neighbors are very good at executing the requested tasks. In other cases however, it is impossible to optimize one parameter without degrading the result of the other. The latter cases are obviously more interesting, since the produced solution will be determined by the importance of each single objective.

To further illustrate the above, consider a representation of the network graph where task requesting nodes included in $\mathcal{N}_T$ are on one side and are connected to all the other nodes in $\mathcal{N}$ based on (i) average delay and (ii) average execution accuracy (see Fig. 3.1c). The optimal decisions of i and j regarding execution delay and accuracy, are highlighted with red and green

for i and j respectively. Optimizing delays and execution accuracies may result to different (in case of i) or similar (in case of j) execution decisions. For j, no matter how important each single objective is, the ideal decision is to outsource. In i's case however, local execution is better in terms of delay minimization, but executing at k is optimal regarding execution accuracy. In this case, the routing result will be significantly affected by the relative priority of the two objectives.

3.3 Problem Formulation and Solution

3.3.1 Problem Formulation

Let us define variable vector $\boldsymbol{x} = \{x_{i,j} \in [0,1], \forall i \in \mathcal{N}_T,\ j \in \mathcal{N}_{(i)}\}$. Its values denote the execution policy of the nodes so that $x_{i,j}$ represents the portion of tasks a node i outsources to some neighboring node j. As a result, $x_{i,i}$ denotes local execution. In addition, we define a sub-vector of $\boldsymbol{x}$ as $\boldsymbol{x_i} = [x_{i,1}, x_{i,2}, \dots x_{i,N}]\ \forall i \in \mathcal{N}_T$, where $N = |\mathcal{N}|$ that contains the execution policy variables related to node i. The average execution delay per task involves two parameters, i.e. the communication delay for input/output data transfer and the computation delay. In our analysis we consider both load-dependent queuing and fixed transmission delay. Namely, the queuing delay $d^Q_{i,j}$ is:

$$d^Q_{i,j} = \frac{S_i}{c_{i,j} - x_{i,j} R_i},\ \forall\ i \in \mathcal{N}_T, j \in \mathcal{N} \tag{3.1}$$

as a result of Little's theorem and assuming that the tasks arrive in $M/M/1$ queues and experience a load dependent delay. Transmission delay $d^T_{i,j}$ is given by:

$$d^T_{i,j} = \frac{S_i}{c_{i,j}},\ \forall\ i \in \mathcal{N}_T, j \in \mathcal{N} \tag{3.2}$$

Since the output of the execution is much smaller in size than the input, it is omitted for simplicity. For instance in a face recognition application, the task input size (photo) is much larger than the application's result which can simply be the name of a person in the photo.

The task execution delay in node j is given by:

$$d^C_j = \frac{1}{\frac{h_j}{\rho_j} - \sum_{k \in \mathcal{N}_{(j)}} x_{k,j} \lambda_k}\ \forall\ j \in \mathcal{N} \tag{3.3}$$

We follow the same $M/M/1$ queuing approach in modeling the computation queuing delay at

the nodes. The service and arrival rates are the task execution and arrival rates for all the neighboring nodes that outsource tasks to some node j. The queuing delay at each node j considers the task's load and the CPU power at the executing node to produce the average task execution rate of j. Notice how increasing variables $x_{k,j}$ increases the task arrival rate and decreases the task execution rate of j, hence affecting apart i, all of j's neighbors as well. As the task arrival rate reaches the execution rate, the queuing delay is infinite and the task queue becomes unstable. Note, that if the traffic load in either queuing or task execution delay exceeds the total link or computing capacities we end up with negative delays. Those are considered extreme situations and the study of this model is limited to the cases where the respective $M/M/1$ queues are stable. The average total (communication and computation) delay is then defined as:

$$D_i(\boldsymbol{x}) = \sum_{j \in \mathcal{N}_{(i)}} x_{i,j} \Big(d^T_{i,j} + d^Q_{i,j} + d^C_j \Big), \ \forall \ i \in \mathcal{N}_T \, . \tag{3.4}$$

Considering the average computation accuracy of each node we use the following definition:

$$W_i(\boldsymbol{x_i}) = \sum_{j \in \mathcal{N}_{(i)}} x_{i,j} w_{i,j}, \ \forall i \in \mathcal{N}_T. \tag{3.5}$$

The system's goal is to minimize the analytics' execution delay and maximize their execution accuracy. We notice that $D_i(\boldsymbol{x})$ is convex with $x_{i,j}, \ j \in \mathcal{N}_{(i)}$ since it's second order derivative with respect to $x_{i,j}$ is a positive number, provided that the transmission and execution queues are stable. Instead of directly maximizing the accuracy, we use a convex utility approach for two main reasons. First we capture the diminishing returns effect that naturally arises in such systems. Namely, a certain improvement in accuracy is more important if the accuracy is low compared to the case that it is already high enough. The same holds for the delay since it is a convex function[3], in the sense that a certain delay deterioration is more negative/impactful if the delay is already high. Moreover, by using such utility for the accuracy we ensure a balanced dispersion of the collaboration benefits across the IoT nodes, satisfying this way a fairness criterion. In particular a fairness utility commonly used in network utility maximization is the a-fairness function [105]. Such function is the logarithmic, which we apply to the average execution accuracy, and by using its convex form $-\log W_i(\boldsymbol{x_i})$ we formulate the objective function

[3]This holds under the assumption that the delays in (3.1),(3.3) are positive.

as a sum of the two objectives:

$$U_i(\boldsymbol{x}) = \beta D_i(\boldsymbol{x}) - \gamma \log W_i(\boldsymbol{x_i}),\ \forall i \in \mathcal{N}_T \tag{3.6}$$

where β and γ are balancing parameters between the two objectives. For the balancing parameters we follow a scalarization approach where certain weights are assigned to the two objectives that we have. This ensures that the obtained solution is aligned with the priorities of the designer (delay or accuracy) and in any case they will be Pareto optimal [106, Sec. 4.7].

Regarding the total power consumption for any given node $j \in \mathcal{N}$ we define the following:

$$p_j^C(\boldsymbol{x}) = e_j^c \sum_{i \in \mathcal{N}_{(j)}} x_{i,j} \lambda_i f_{i,j}, \tag{3.7}$$

$$p_j^T(\boldsymbol{x}) = \sum_{i \in \mathcal{N}_{(j)}} e_{j,i}^t x_{j,i} R_j, \quad p_j^R(\boldsymbol{x}) = \sum_{i \in \mathcal{N}_{(j)}} e_{j,i}^r x_{i,j} R_i, \tag{3.8}$$

where p_j^C, p_j^T and p_j^R are the total power spent on computation, transmission and reception by node j respectively. Consequently, the joint execution delay and accuracy optimization problem is:

$$\boldsymbol{P_1}: \quad \underset{\boldsymbol{x}}{\text{minimize}} \sum_{i \in \mathcal{N}_T} U_i(\boldsymbol{x}) \tag{3.9a}$$

$$\text{subject to:} \quad g_{1,j}(\boldsymbol{x}) = \sum_{i \in \mathcal{N}_{(j)}} \frac{x_{j,i} R_j}{c_{j,i}} - 1 \leq 0,\ \forall\ j \in \mathcal{N}, \tag{3.9b}$$

$$g_{2,j}(\boldsymbol{x}) = \sum_{i \in \mathcal{N}_{(j)}} \frac{x_{i,j} R_i}{c_{i,j}} - 1 \leq 0,\ \forall\ j \in \mathcal{N}, \tag{3.9c}$$

$$g_{3,j}(\boldsymbol{x}) = p_j^C(\boldsymbol{x}) + p_j^T(\boldsymbol{x}) + p_j^R(\boldsymbol{x}) - P_j \leq 0,\ \forall j \in \mathcal{N} \tag{3.9d}$$

$$g_{4,i,j}(\boldsymbol{x}) = x_{i,j} - 1 \leq 0,\ \forall\ i \in \mathcal{N}_T\ , j \in \mathcal{N}, \tag{3.9e}$$

$$g_{5,i,j}(\boldsymbol{x}) = -x_{i,j} \leq 0,\ \forall\ i \in \mathcal{N}_T\ , j \in \mathcal{N}, \tag{3.9f}$$

$$h_i(\boldsymbol{x}) = \sum_{j \in \mathcal{N}_{(i)}} x_{i,j} - 1 = 0,\ \forall\ i \in \mathcal{N}_T. \tag{3.9g}$$

Eq. (3.9b)-(3.9c) guarantee the flow feasibility communication wise, i.e. they respect channel capacities [107], [108]. The power budget constraint for each node is given by (3.9d). Note that it accounts for energy spent on both computation and communication. Finally, the sum of decision variables for each node i, over all its neighbors must be equal to 1 according to (3.9g)

so that our routing policy is formed.

$\boldsymbol{P_1}$ is a convex yet challenging optimization problem. Note that $U_i(\boldsymbol{x})$ includes load-sensitive delay components and a logarithm of a summation. Moreover, the outsourcing and routing decisions of each node affect the resource consumption of its neighbors and through them, that of further-distant nodes. Thus, there is strong coupling among the decisions of nodes both in the objective and in the constraint set. Finally, often such IoT systems comprise hundreds of nodes, and their large size renders the above challenges more pronounced.

3.3.2 Problem Solution

In order to address the above challenges, we rely on Lagrange duality that allows the decoupling of the constraint set of $\boldsymbol{P_1}$. In detail, Lagrange relaxation allows for efficient and iterative minimization of the primal variables by embedding the problem's well-defined constraint functions into the objective. Furthermore we define a new set of auxiliary variables for each node i, which serve as local copies of the decisions of the neighbors of j that affect its operation. This facilitates the decoupling of the objectives of the different nodes, and enables the distributed execution of the proposed algorithm. Namely, we define variables $y_{i,j}$ for each $j \in \mathcal{N}_{(i)}$, and introduce the necessary constraints:

$$y_{i,j} = x_{j,i}\,, \qquad \forall (i,j) \in \mathcal{L} \tag{3.10}$$

which ensure the *consistent* operation of the system. This transformation allows the scalable and distributed solution of the problem, which is particularly important for large IoT systems.

We first define the Lagrange function by relaxing all constraints:

$$\begin{aligned} L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\mu},\boldsymbol{v},\boldsymbol{\kappa}) = \sum_{i\in\mathcal{N}_T} U_i(\boldsymbol{x},\boldsymbol{y_i}) + \sum_{k=1}^{3}\Big(\sum_{j\in\mathcal{N}} \mu_{k,j} g_{k,j}(\boldsymbol{x})\Big) + \sum_{k=4}^{5}\Big(\sum_{i\in\mathcal{N}_T}\sum_{j\in\mathcal{N}} \mu_{k,i,j} g_{k,i,j}(\boldsymbol{x})\Big) + \\ \sum_{i\in\mathcal{N}_T} v_i h_i(\boldsymbol{x}) + \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} \kappa_{i,j}\Big(y_{i,j} - x_{j,i}\Big) \end{aligned} \tag{3.11}$$

where $\boldsymbol{\mu}, \boldsymbol{v}$ and $\boldsymbol{\kappa}$ are the Lagrange multiplier vectors for the inequality and equality constraints respectively . $\boldsymbol{y}$ is defined as: $[y_{i,j}, \forall i \in \mathcal{N}, j \in \mathcal{N}_T]$. The dual problem is defined then as:

$$\max_{\boldsymbol{\mu}\geq\boldsymbol{0},\boldsymbol{v},\boldsymbol{\kappa}} \psi(\boldsymbol{\mu},\boldsymbol{v},\boldsymbol{\kappa}) = \inf_{\boldsymbol{x}}\ L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{\mu},\boldsymbol{v},\boldsymbol{\kappa}). \tag{3.12}$$

In order to solve (3.12), we follow a primal-dual decomposition method that gradually converges to the optimal solution. In each iteration, a gradient method is used to obtain improved values for the dual variables, and then use them to calculate the current gradient through the minimization of (3.11).

First, we observe that the dual problem has a separable structure and hence it favours a distributed solution. The latter is beneficial since it is scalable and requires only minimal circulation of coordination messages. Namely each node $i \in \mathcal{N}_T$ can optimize $\boldsymbol{x_i}, \boldsymbol{y_i}$, so the Lagrangian is written as a sum of partial Lagrangians L_i over all nodes $i \in \mathcal{N}_T$ such that:

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{\kappa}) = \sum_{i \in \mathcal{N}} L_i(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\mu_i}, \boldsymbol{v_i}, \boldsymbol{\kappa_i}), \tag{3.13}$$

where $\boldsymbol{y_i}$ is defined as: $\boldsymbol{y_i} = [y_{i,1}, y_{i,2}, \ldots, y_{i,N}], \forall i \in \mathcal{N}$. $\boldsymbol{\mu_i}, \boldsymbol{v_i}$ and $\boldsymbol{\kappa_i}$ are sub-vectors of $\boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{\kappa}$ that only contain the Lagrange multipliers associated with each node i, i.e. those appearing in each node's partial Lagrangian function given by:

$$\begin{aligned} L_i(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\mu_i}, \boldsymbol{v_i}, \boldsymbol{\kappa_i}) = U_i(\boldsymbol{x_i}, \boldsymbol{y_i}) + \sum_{k=1}^{3} \Big(\sum_{j \in \mathcal{N}} \mu_{k,j} g_{k,j}(\boldsymbol{x_i}) \Big) + \sum_{k=4}^{5} \Big(\sum_{j \in \mathcal{N}} \mu_{k,i,j} g_{k,i,j}(\boldsymbol{x_i}) \Big) + \\ v_i h_i(\boldsymbol{x_i}) + \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \kappa_{i,j} \Big(y_{i,j} - x_{j,i} \Big). \end{aligned} \tag{3.14}$$

The dual ascent method consists of a minimization step, where the Lagrangian (3.11) is minimized with respect to $\boldsymbol{x}$ for some given $\boldsymbol{\mu}, \boldsymbol{v}$. After that, the Lagrange multipliers are updated using the solution obtained from the minimization step before repeating the minimization step in an iterative way, until $\boldsymbol{x}$ converges to its optimal value $\boldsymbol{x}^*$. Formally we can write:

$$(\boldsymbol{x_i}^t, \boldsymbol{y_i}^t) = \arg \min_{\boldsymbol{x_i}, \boldsymbol{y_i}} L_i(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\mu_i}^{t-1}, \boldsymbol{v_i}^{t-1}, \boldsymbol{\kappa_i}^{t-1}) \tag{3.15a}$$

$$\boldsymbol{\mu_j}^t = [\boldsymbol{\mu_j}^{t-1} + \theta^t \boldsymbol{g_j^t}]^+, \tag{3.15b}$$

$$\boldsymbol{v_j}^t = \boldsymbol{v_j}^{t-1} + \theta^t \boldsymbol{h_j^t}, \qquad \kappa_{i,j}^t = \kappa_{i,j}^{t-1} + \theta^t (y_{i,j} - x_{i,j}) \tag{3.15c}$$

where the superscript t indicates a variable's value at the t-th iteration and $\theta^t > 0$ is the step size. Vectors $\boldsymbol{g_j^t}$ and $\boldsymbol{h_j^t}$ contain the values of (3.9b)-(3.9f) and (3.9g) respectively at the constraints related to j, while $[x]^+$ denotes that $x = \max\{0, x\}$. Each node $i \in \mathcal{N}$ can then evaluate $\boldsymbol{x_i}^*, \boldsymbol{y_i}^*$ after receiving the dual variable updates in an iterative process, which is briefly described in

Algorithm 1.

Algorithm 1 Distributed Collaborative Task Execution

1: **Input:** $\mathcal{N}, \mathcal{N}_T, \beta, \gamma, \theta, \boldsymbol{\mu_i^0}, \boldsymbol{v_i^0}, \boldsymbol{\kappa_i^0}$
2: **repeat**
3: **for each** node $i \in \mathcal{N}T$ **do**
4: $t \leftarrow t+1$
5: $(\boldsymbol{x_i}^t, \boldsymbol{x_i}^t) \leftarrow \arg\min_{\boldsymbol{x_i}, \boldsymbol{y_i}} L_i(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\mu_i}^{t-1}, \boldsymbol{v_i}^{t-1}, \boldsymbol{\kappa_i}^{t-1})$
6: Transmit $\boldsymbol{x_i}^t$ to neighbors
7: **end for**
8: **for each** node $j \in \mathcal{N}$ **do**
9: Update $\boldsymbol{\mu_j}^t, \boldsymbol{v_j}^t, \boldsymbol{\kappa_{j,i}}^t$ by using (3.15b), (3.15c)
10: Transmit $\boldsymbol{\mu_j}^t, \boldsymbol{v_j}^t, \boldsymbol{\kappa_{j,i}}^t$ to neighbors
11: **end for**
12: **until** $||\boldsymbol{x_i}^t - \boldsymbol{x_i}^{t-1}|| \leq \epsilon$

The algorithm's input is the number of nodes, the balancing parameters, the step size and the initial values for the Lagrange multipliers, all considered known to each node. The step size is chosen as a sufficiently small positive value to allow the algorithm to converge. Steps 2-12 describe the distributed iterative process of converging to the optimal $\boldsymbol{x_i}^*$. Each node $i \in \mathcal{N}_T$ evaluates $\boldsymbol{x_i}, \boldsymbol{y_i}$ (step 5), and transmits $\boldsymbol{x_i}$ to its neighbors, so they update their associated Lagrange multipliers (step 9) and send them to their neighbors (step 10) so that a new iteration can commence. Termination of the algorithm is achieved at some iteration t, for which it holds $||\boldsymbol{x_i}^t - \boldsymbol{x_i}^{t-1}|| \leq \epsilon,\ \forall i \in \mathcal{N}_T$, where ϵ is a small positive, close to 0 value (step 12).

Proposition 1 *Algorithm 1 solves problem* $\boldsymbol{P_1}$ *and converges to within* ϵ *from the optimal solution* $\boldsymbol{x^*}$*, in polynomial time.*

Proof The Dual Ascent method solves the Lagrangian dual problem where the dual function is given in (12), by iteratively increasing the dual variables $\boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{\kappa}$ towards maximizing $\psi(\boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{\kappa})$. The maximum value of $\psi(\boldsymbol{\mu}^*, \boldsymbol{v}^*, \boldsymbol{\kappa}^*) = d^*$ provides a lower bound on the solution p^* of $\boldsymbol{P_1}$. Since the objective $\sum_{i \in \mathcal{N}_T} U_i(\boldsymbol{x})$ is strictly convex, if Slater's conditions apply strong duality holds so that $d^* = p^*$; and solving the dual problem, using steps (3.15a)-(3.15c), solves the primal [106, Sec 5.3]. Algorithm 1 is the distributed implementation (dual decomposition) of the dual ascent method, which has been shown to converge linearly to $\boldsymbol{x^*}$ [109]. The communication overhead from steps 3, 9, 15 is upper bounded by the maximum number of neighbors of each node, which is at most N. In that case $\boldsymbol{x_i}$ contains N variables, and the dual vectors $\boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{\kappa}$ contain

$3 + 2N$, 1 and N variables, respectively. All together we get an exchange of $M = 3(N + 1) + 1$ variables, so the overhead is of $O(NM)$ order.

3.4 Performance Evaluation

In this section we evaluate the performance of the proposed system and the associated algorithm devised to obtain optimal task outsourcing. We first describe the simulation setup, and then conduct a sensitivity analysis for selected key parameters of our model. Finally, we demonstrate the distributed algorithm's convergence and scalability properties for different network sizes.

Evaluation Setup. In order to evaluate our algorithm for relatively large network sizes,in our evaluation we use a custom simulator whose parameters are based on measurements on a small testbed of Raspberry Pi 3 Model B devices.

We measure compute and Wi-Fi power consumption using a digital multi-meter connected to the Pis. We measured compute power consumption by stressing 1-4 CPU cores of each Pi[4] using a methodology similar to [110]. Wi-Fi power consumption was measured by sending iperf UDP traffic at different data rates between two Pis connected with a 802.11g link in ad hoc mode. The power measurements are used to determine $e^t_{i,j}, e^r_{i,j}$ and P_j parameters in our simulator (See Table 3.1).

The Pis are running a face recognition application, implemented in Java OpenCV [47]. The motivation behind using such an application lies in the fact that video analytics, e.g. face/object recognition, are highly data intensive, and generate a huge load of data for processing per task. This makes them ideal for stressing the edge network performance and highlighting the accuracy/delay trade-offs this work aims to analyze. The application has a source that generates frames from a local video feed, detects faces in each frame and compares found faces to a local database. The faces and names of any matches are output to a sink. We ran the application locally at each Raspberry Pi and measured throughput and accuracy for three different image resolutions/sizes S_i (5, 8 and 32 KB) and classification algorithms (LBP, Haar), when the source sends frames in backlogged mode. Each combination of (image resolution, classification algorithm) parameters corresponds to a different type of input source in our simulator that also affects the required number of cycles per task ρ_j.

[4]Each Pi has a 1.2 GHZ quad-core ARM Cortex A53 processor.

Description	Parameter	Value
Number of nodes, Number of nodes requesting tasks	N, N_T	25, 10
Transmission and reception energy consumption per bit	$e^t_{i,j}, e^r_{i,j}$	15-78, 5.5-38 nJ/b
Number of cycles per task	ρ_j	52-889 Mcycles
Energy consumption per computation cycle	e^c_j	0.2 nJ/c
Data per task	S_i	5-32 KB
Energy budget	P_j	0.4 W
Balancing parameters	β, γ	0.5
Task arrival rate	λ_i	1

Table 3.1: Simulation parameters.

We used Matlab R2015a in order to simulate the behavior of the system, and obtain a solution for $\boldsymbol{P_1}$. The assumption that communication and task execution delays are positive (although not guaranteed in general), is enforced here by proper adjustment of other system parameters like the link capacity and task arrival rate. The IoT network is created using a random geometric graph model that is appropriate for such ad hoc networks. Namely, we have randomly placed the nodes in a $100m \times 100m$ area, assuming that they are in communication range when their Euclidian distance is less than 50m, while their link capacities are calculated based on path loss and Rayleigh Fading. For each experiment we present results by averaging 100 simulations that differ in node locations.

Balancing parameters β, γ and objective correlation. First we evaluate the impact of β and γ on the execution delay and accuracy. As it is expected, when the ratio β/γ is increased, the solution of $\boldsymbol{P_1}$ prioritizes delay (which is reduced) over accuracy (which deteriorates). Fig. 3.2 quantifies this effect, where we have averaged the delay and accuracy for all nodes in 100 different random networks with the same parameters except node location. We observe that when β increases up to 0.4, the delay is reduced to less than 0.4 secs (about 60% average improvement to $\beta = 0$), while for higher β values this improvement is smaller (still monotonic). On the other hand, the accuracy deteriorates with almost constant rate, reduced down to 73% for the extreme case the system gives full priority to delay ($\gamma = 0, \beta = 1$). Clearly, the exact values of average delay and task accuracy as β/γ changes depend also on the other system parameters shown in Table I. Moreover, this trade off is largely shaped by the network structure, in particular the properties of neighboring nodes, and it is important to understand this dependency.

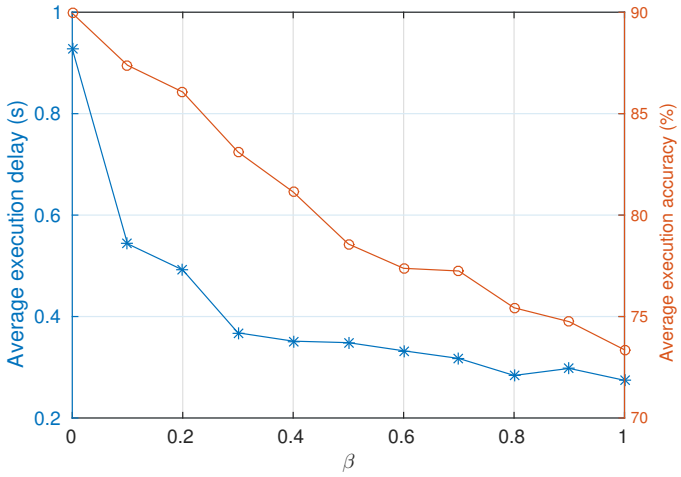


Figure 3.2: The effect of balancing parameter β and $\gamma = 1 - \beta$ on the task execution time and task accuracy. Results are averaged for all nodes in each network, and for 100 different random network instances.

To this end, we go a further step and use the Kendal rank correlation coefficient [111] in order to predict, for a given network, how steep this trade off curve will be. In detail, for each node $i \in N_T$ we create two ordered lists of its neighbors, ranking them according to delay (lower to higher) and accuracy (higher to lower). In other words, the top-ranked node in the delay list is the one that can execute node i's tasks with the lower delay, while the top-ranked node in the accuracy list offers the highest accuracy. Then, for each node, we test for all possible pairings between its neighbors, if they maintain their order in the two lists or not and count the number of concordant and discordant pairs. The Kendal rank correlation coefficient for a node i with N_i neighbors is given by:

$$\tau_i = \frac{P_i^c - P_i^d}{N_i(N_i-1)/2}, \ \forall i \in \mathcal{N}_T \tag{3.16}$$

where P_i^c is the number of concordant pairs and P_i^d the number of discordant pairs. Since $N_i(N_i-1)/2$ is the number of possible pairs it holds that: $-1 \leq \tau_i \leq 1$ and averaging for all $i \in \mathcal{N}_T$ we can obtain a unique objective similarity index τ for our network.

In Fig. 3.3a,3.3b we depict the average execution delay and accuracy (mean values for 100 instances) of the solution of $\boldsymbol{P_1}$, when parameters $w_{i,j}$ are chosen so that $\tau \in \{-1, 0, 1\}$. We observe that when $\tau = -1$, i.e., the objectives are conflicting, the choice of β plays a significant role in the solution. For $\beta = 0$ we get high delay and accuracy, while for $\beta = 1$ the opposite.

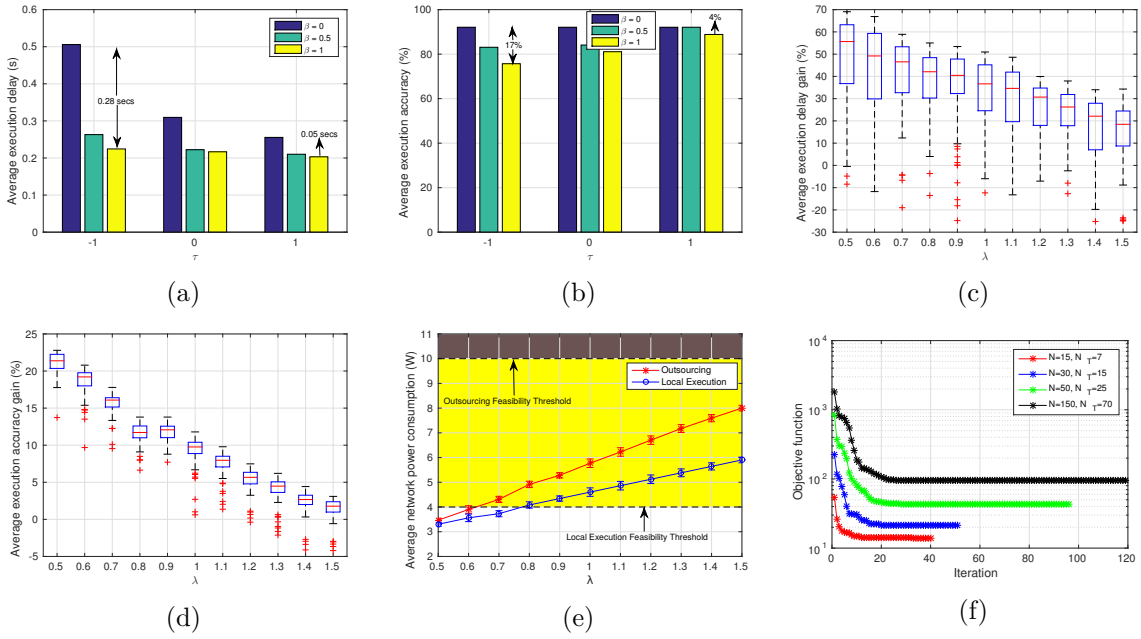


Figure 3.3: Average network execution delay and accuracy for different levels of objective correlation in (a) and (b). The average gain in execution delay and accuracy as a function of the task arrival load λ in (c) and (d). (e) Power consumption of outsourcing and local execution approaches, (f) the objective function at each algorithm iteration.

As the correlation of the objectives becomes stronger, the impact of β becomes smaller. In the extreme case of $\tau = 1$, the objectives are fully correlated and optimizing with respect to delay is roughly the same as optimizing with respect to accuracy.

Impact of load on cooperation benefits. Next, we explore the impact of the average task load λ on the *cooperation delay gain* of the IoT network. The latter is defined as the average task delay improvement (%) when the nodes collaborate based on the policy of $\boldsymbol{P_1}$, over when they execute the tasks locally. In Fig. 3.3c we plot this gain for different values of λ, where we have set $\lambda_i = \lambda, \forall i \in \mathcal{N}_T$. Note that in order to fairly compare collaborative with local execution, we have considered a large power budget ($P_j = 1W, \forall j \in \mathcal{N}$), so that neither mode (local or collaborative) violates the power constraints in $\boldsymbol{P_1}$.

There are some interesting observations here. First, note that the gain is positive, since some nodes expedite their tasks by outsourcing them to neighbors with high processing power with which they are connected with high-capacity links. For larger values of lambda however, the queuing delays increase fast (due to congestion) and hence our policy sacrifices performance (lower delay gains) in order to satisfy the power consumption constraints. Interestingly, in some

cases, the cooperation might increase the delay for some nodes (negative delay gain) in order to achieve higher accuracy.

Similarly, in Fig. 3.3d we depict the *cooperation accuracy gain*, defined as the average task accuracy when the nodes collaborate, minus the average accuracy when all tasks are executed locally. Hence, the gain is positive if collaborative performance is better. We see again that there are important cooperation gains (few outliers are present), which are reduced for high load values (even by 20%). These results show that even if the network is not energy-constrained, our algorithm has merit as it improves both the average delay and accuracy. However, as the load increases, the outsourcing of all tasks becomes very costly in terms of energy or bandwidth overheads, hence the average gains are reduced.

Outsourcing feasibility gains. Clearly, more often than not, the nodes will have tighter energy budgets than assumed above. In these cases, the cooperation among the nodes not only increases the system's performance, but also ensures the completion of the tasks even if they arrive at a (high) rate that cannot be supported by local-only execution. This is demonstrated in Fig. 3.3e, where we have averaged results over 100 network instances. We observe that for $\lambda < 0.8$ both the outsourcing and the local execution solutions are feasible (white area), but for higher loads the local execution exceeds the nodes' aggregate power allowance, i.e., $\sum P_i$. When task outsourcing is allowed however, all nodes $\in \mathcal{N}$ contribute to the power budget which increases and allows the support of more tasks. Finally, it is interesting to note that the cooperative solution consumes more energy than the local-only execution (focus on the white area), as it tries to maximize the objective of $\boldsymbol{P_1}$ by achieving lower delay and higher accuracy than the local solution.

Convergence of Algorithm 1. Finally, in Fig. 3.3f we evaluate the convergence of the distributed Algorithm 1, using 4 different network sizes and a fixed step size. We observe that the optimal solution is reached after few iterations, even for large networks of $N = 150$ nodes, which verifies the scalability properties of the algorithm. Furthermore, we see that in practice, the algorithm achieves approximately 95% of the optimal solution after only 25 iterations for all experiments.

3.5 Conclusions

In this chapter we studied how the heterogeneity of IoT networks plays an important role in motivating the collaboration of devices towards executing data analytic tasks. The different accuracies, delays and power consumption profiles of the devices allows them to adopt mutually beneficial task offloading policies towards optimizing the entire system's performance. In many cases however, the devices are not be a part of the same system, and might misreport their parameters and requirements towards exploiting the resources of others. In the following chapter we deal with such scenarios by properly motivating truthful device cooperation.

Chapter 4

Providing Incentives for the Cooperation of Edge Devices

4.1 Introduction

Edge-based cooperative architectures have been extensively used in communication networks, cf. [112], but IoT analytics raise novel techno-economic challenges. On the technical side, IoT nodes are likely to offer different accuracy and execution delay for each task, e.g., due to their training datasets (see Fig.4.1). Additionally, they might be heterogeneous in terms of their computation, wireless connectivity, and energy resources. Therefore, it is highly non-trivial to identify the assignment of tasks to nodes that maximizes the overall performance while respecting the resource constraints. Besides, some assignments might improve accuracy and others the delay, and it is not clear how (and if) one can combine these different criteria.

On top of that, IoT nodes are often owned by different users who, clearly, will not spend voluntarily their resources for serving others. This *incentive-misalignment* issue has been largely ignored by previous research, yet it arises often in practice. Our cooperative architecture is more scalable than cloud/cloudlet solutions [57], which are inevitably confined by the unavailability of link bandwidth and computing capacity, respectively. Prior proposals for fostering collaboration e.g., in ad hoc networks [113] or file sharing overlays [114], include solutions based on reputation, reciprocation or market mechanisms. However, this problem takes a new twist in IoT analytics because of the different types of involved resources (bandwidth, computing, energy), the heterogeneity of nodes in terms of task execution accuracy and delay, the massive scale of IoT systems,

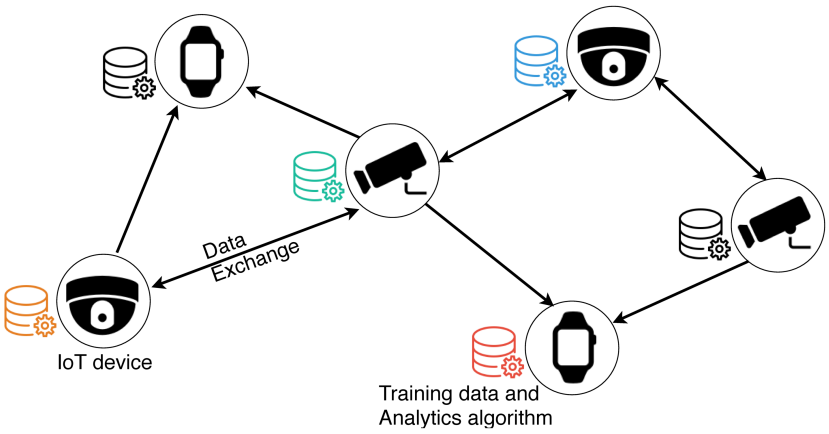


Figure 4.1: Example of IoT network with heterogeneous nodes. Camera-equipped nodes use their neighbors' resources for faster and/or more accurate execution of video analytics.

and the typically low power availability of nodes.

It becomes clear that the cooperative execution of IoT edge analytics is instrumental for the success of these services, but the question of how to maximize their performance and ensure the users participation remains open. Our goal is to address this issue by introducing an auction framework, Coop-IoT, for optimizing the execution delay and accuracy of these tasks, while offering cooperation incentives to nodes. We note that our analysis builds on our previous work which studied the coordination problem [115],assuming cooperative nodes and full information.

4.2 Methodology and Contributions

We consider a general model of a wireless IoT network with heterogeneous nodes and one gateway. Each node generates different types of tasks which can execute locally with a certain accuracy and delay. The nodes are connected with links of different capacity and spend their energy for exchanging data or computing the analytics. The performance target for each node, in terms of accuracy and delay, is captured by a utility function which is private information known only to that node. Our model has minimal assumptions and can be tailored to different IoT scenarios by selecting proper utility functions.

We formulate the execution of analytics as a network utility maximization problem. This yields an intricate mathematical program that cannot be solved by a central entity (the IoT gateway) which, naturally, is not aware of the nodes' utilities. The nodes are prone to misreport their needs and costs so as to increase their performance at the expense of others resource

consumption. To address this issue, we propose an auction-based mechanism, implemented by the gateway, that elicits the unknown utility parameters and solves the NUM problem. This is achieved by carefully-designed pricing and allocation rules that guide the nodes behavior to the system-wide optimal equilibrium. Unlike other auctions, our framework can handle the multi-dimensional resource constraints arising in this setting, it is lightweight in terms of communication requirements and computation load, and can be executed in a decentralized fashion when necessary.

Going a step further, we fully implement our solution as a network protocol and evaluate it in a series of experiments using a wireless testbed of Raspberry Pis (RPis) and state-of-the-art face recognition applications. Moreover, we use these measurements to simulate larger networks and compare with several alternative competitors. We find that Coop-IoT improves substantially the service accuracy and delay compared to simple or more sophisticated heuristics, adapts its operation to account for the resource availability, and ensures the nodes' cooperation.

To the best of our knowledge, this is the first work proposing and testing an *incentive-compatible* cooperative framework for IoT analytics. Our main contributions are therefore:

- *Analytics Optimization.* We introduce the problem of optimizing the execution accuracy and delay of data analytics in IoT networks, where the nodes can cooperate to improve their joint performance. This is a fundamental problem appearing in many IoT scenarios, see [37–41].

- *Incentive Mechanism.* We design a novel auction for maximizing the aggregate analytics performance while aligning the nodes' cooperation incentives. Our algorithm handles the intricate (non-linear, coupled) objectives and constraints of the optimization problem, the co-existence of multiple task sellers and buyers, and it is amenable to distributed implementation. To the best of our knowledge, there is no other mechanism satisfying these requirements.

- *Multi-stage analytics.* We extend the model for multi-stage tasks that involve data preprocessing (e.g., feature extraction) and analytics (e.g., classification) executed at different nodes. This is the first model that enables optimization of multi-stage cooperative analytics.

- *Implementation and Evaluation.* We fully implemented Coop-IoT in a wireless testbed using several face-recognition algorithms and real datasets. We verified that the heterogeneity of nodes induces different execution accuracy and delay; measured the overheads and scalability

of our mechanism; and compared its performance to benchmark policies. We find that Coop-IoT, unlike its competitors, ensures cooperation and is up to 50% faster and 30% more accurate, while consuming less resources.

At the core of our framework lies a novel auction algorithm. Auctions are attractive for designing cooperation mechanisms for e.g., routing [113], file-sharing [114], power control [116], or crowdsensing [34]. Their design is notoriously hard when there are multiple buyers and sellers (double auction, DA) [117]. One prominent type of DAs is VCG which, however, exhibit high computational complexity and budget-imbalanced outcomes [118]. Another option is the McAfee auction [119] used in cooperative routing [113] or spectrum allocation [120], which does not maximize efficiency. Finally, truthful mechanisms for crowdsensing [34, 36] are pertinent to our problem, yet they consider one task requester and information collection and do not account for energy costs or task delay; while social-network based solutions [121] rely on an underlying social graph, a condition that is not required by Coop-IoT.

Our auction design is more challenging because IoT analytics involve *(i)* heterogeneous bandwidth and computing resources, *(ii)* multiple buyers and sellers, *(iii)* user-specific objective functions, and *(iv)* constraints that couple the nodes' actions. Hence, we need a mechanism more sophisticated than VCG or McAfee [119] in terms of constraints, but more lightweight in terms of overheads. An interesting prior effort is the Kelly-VCG mechanism [122], which reduces overheads but is for single-sided auctions and requires still the calculation of VCG prices [118]. Our approach is inspired by [123] which however does not include costs and has one seller. Walrasian DAs have been used by [124] and [125] among others, but do not handle multi-stage analytics and energy costs. Our auction algorithm has minimal overheads, i.e., scalar bids are transmitted and simple function evaluations and gradient updates are involved, and it is amenable to distributed execution.[1] Hence, this is a result of independent importance, beyond the context of IoT analytics.

4.3 Model and Problem Formulation

Network. We consider a wireless IoT network modeled with a directed connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ of N nodes and L links. The nodes are resource-constrained devices operating with

[1] [123] was used for congestion control protocols; Coop-IoT can be used for bandwidth/computing management in IoT.

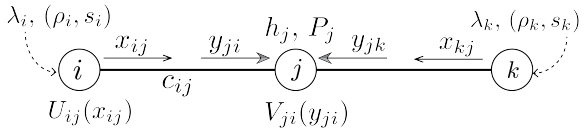


Figure 4.2: Node j serves y_{ji} requests of i using its power P_j and processing H_j, incurring cost $V_{ji}(y_{ji})$; and similarly for k. Node i generates λ_i tasks/sec, with ρ_i computing and s_i data load, and gets $U_{ij}(x_{ij})$ utility if sends $x_{ij}\lambda_i$ tasks to j.

a standard IoT technology such as Wi-Fi or ZigBee. They are equipped with half-duplex radios, hence they transmit to, or receive from at most one other node at each time. A link $l_{ij} \in \mathcal{L}$ exists if node j is within communication range of node i, and we define the set $\mathcal{N}_i = \{j : (j,i), (i,j) \in \mathcal{L}\}$. Every link l_{ij} has average capacity c_{ij} (bits/s) which is measured by higher layer mechanisms[2], e.g., [102].

Data transmissions over link l_{ij} induce energy cost of e^t_{ij} Joules/bit for the transmitting node i, and e^r_{ij} Joules/bit for the receiving node j. These parameters depend on link capacities $c_{ij}, \forall (i,j)$ [103]. We consider the general case where capacities and energy costs of links l_{ij} and l_{ji} might differ. Each node i has an average power budget of P_i Watts that can be used for computing or data transfers. Such constraints arise in IoT nodes that, due to their small form-factor, cannot spend energy at a high rate, or have to rely on limited energy sources [56]. There is also an IoT gateway with a global network view, that can communicate (over one or more hops) with all nodes, but is oblivious to their analytic requirements.

Computing & Data Analytics. Each node i generates tasks with average rate $\lambda_i \geq 0$ tasks/sec, where each task has data load of s_i bits/task and computing load (for node i) of ρ_i cycles/task. Our model is generic and captures two key classes of compute models: *(i)* the data stream processing model where the tasks are data (e.g., images) sent to operator function modules (e.g., face recognition) deployed at the nodes; *(ii)* a Map-Reduce model where the tasks are *operators* that are sent to process local data at the nodes. Each node i has computing capacity H_i (cycles/sec) that constraints the rate at which it executes analytics. Finally, e^c_i denotes the energy that i consumes per computing cycle (Joules/cycle).

We focus on predictive analytics where the performance criterion is accuracy and fast execution. We consider the general case where the tasks of each node i can be executed with different accuracy at each node j, and we introduce the accuracy gain parameter $w_{ij} \in [0,1]$ and

[2]Our focus is on analytics scheduling at higher layers that are MAC agnostic. Optimizing MAC for IoT is an orthogonal problem, and out of the scope of this work.

the matrix $\boldsymbol{w} = (w_{ij} : i, j = 1, \ldots, N)$. The different accuracies arise because the nodes might use different algorithms or training data sets [104], and because the nodes may have different analytic requirements. We verify these issues with our own measurements on our testbed in Sec. 4.6 (see Table 4.1). The delay criterion, on the other hand, is related to the wireless network (time to transmit) and node computing resources (time to compute). Ideally, each node i would like to outsource its tasks to node $j \in \mathcal{N}_i$ with the highest w_{ij} value and highest capacities H_j, c_{ij}.

Decision Variables. The goal of our Coop-IoT system is to maximize its aggregate performance while minimizing the nodes' costs. The performance objective is twofold: *(i)* minimize the execution delay for the tasks, and *(ii)* maximize task accuracy. In terms of costs, the major concern is the availability of energy that underpins both communications and computations. Clearly, the nodes cannot exceed their energy budget, but also they would prefer to reduce their energy consumption as much as possible.

Let us denote with $x_{ij} \in [0, 1]$ the fraction of tasks that node i decides to outsource to node j, and define the vector variable $\boldsymbol{x}_i = (x_{ij} : j \in \mathcal{N}_i), \forall i$; and also use x_{ii} for the fraction of tasks i executes locally. Similarly, we introduce the decision y_{ij} of node i for admitting tasks from node j, and define: $\boldsymbol{y}_i = (y_{ij} : j \in \mathcal{N}_i), \forall i$. In order for the *system operation to be consistent* the nodes have to agree on the task execution policy, hence the constraint $x_{ij} \leq y_{ji}, \quad \forall (i, j) \in \mathcal{L}$ must hold, meaning that j agrees to admit at least as many tasks as i outsources. Each node i consumes energy for transmitting and receiving data, and for computing the tasks. We define the power consumption function

$$E_i(\boldsymbol{x}_i, \boldsymbol{y}_i) \triangleq \sum_{j=1}^{N_i} e_{ij}^t x_{ij} \lambda_i s_i + \sum_{j=1}^{N_i} e_{ji}^r y_{ij} \lambda_j s_j + e_i^c \rho_i \Big(\sum_{j=1}^{N_i} \lambda_j y_{ij} + x_{ii} \lambda_i \Big).$$

Utility Functions. Each node i decides how many tasks to outsource to its neighbors and how many to admit from them, aiming to maximize its performance and reduce its energy cost. We define the node's utility for collaborating with its neighbor j as:

$$U_{ij}(x_{ij}) \triangleq \alpha_{ij} \log(\lambda_i w_{ij} x_{ij} + 1) - \beta_{ij} \Big(x_{ij} \lambda_i \big(\frac{s_i}{c_{ij}} + \frac{\rho_j}{H_j} \big) \Big)^2.$$

The first term quantifies the utility for sending tasks to j, which increases with w_{ij} but has diminishing returns (implying a saturation). The benefits are proportional to parameter α_{ij}

which is private information for i. The second term includes the transmission and computing delay of sending and processing s_i bits at j, and we use a quadratic form to capture that users are increasingly dissatisfied with delay. Parameter β_{ij} shapes this dis-utility.[3]

We model the energy cost that i experiences when executing analytics for $j \in \mathcal{N}_i$ with the function:

$$V_{ij}(y_{ij}) = \gamma_{ij}\left(e^r_{ji} y_{ij} \lambda_j s_j + e^c_i \rho_i \lambda_j y_{ij}\right)^2, \tag{4.1}$$

where γ_{ij} is private information of i and captures how sensitive it is in spending its energy. Our model allows nodes to have serving preferences (as utility/cost parameters depend both on i and j), but applies also when $\gamma_{ij} = \gamma_i, \forall j$. An example is shown in Fig. 4.2.

Optimization Problem. We can now express the IoT cooperative analytics with the network utility maximization program: Operation Problem (IoT-OP):

$$\underset{\boldsymbol{x},\boldsymbol{y}\geq \mathbf{0}}{\text{maximize}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N_i} U_{ij}(x_{ij}) - V_{ij}(y_{ij}) = f(\boldsymbol{x},\boldsymbol{y}) \tag{4.2}$$

$$\text{subject to:} \quad x_{ii} + \sum_{j=1}^{N_i} x_{ij} \leq 1, \;\; i \in \mathcal{N}, \tag{4.3}$$

$$x_{ji} \leq y_{ij}, \;\; (i,j) \in \mathcal{L}, \tag{4.4}$$

$$\lambda_i \rho_i x_{ii} + \sum_{j=1}^{N_i} y_{ij} \lambda_j \rho_i \leq H_i, \;\; i \in \mathcal{N}, \tag{4.5}$$

$$\sum_{j=1}^{N_i} \frac{x_{ij}\lambda_i s_i}{c_{ij}} + \sum_{j=1}^{N_i} \frac{y_{ij}\lambda_j s_j}{c_{ji}} \leq 1, \; i \in \mathcal{N}, \tag{4.6}$$

$$E_i(\boldsymbol{x}_i, \boldsymbol{y}_i) \leq P_i, \;\; i \in \mathcal{N} \tag{4.7}$$

with $\boldsymbol{x} = (\boldsymbol{x}_i, i \in \mathcal{N})$ and $\boldsymbol{y} = (\boldsymbol{y}_i, i \in \mathcal{N})$. Eq. (4.3) couples the outsourced and locally-executed tasks; (4.5) ensures the computing capacity of nodes; and (4.6) captures the transmit/receive limitations due to half-duplex operation. We note that this formulation captures interference using the primary interference model (node-exclusive spectrum sharing model) where only links adjacent to each node interfere due to the half-duplex radio constraints. In Sec. 4.5 we show how our framework can be adapted to the protocol interference model which captures secondary interference of links in a single frequency channel using the Wi-Fi data/ack protocol.

IoT-OP is a convex problem and its solution is given by the KKT conditions. We relax the

[3]Based on the specific IoT application one could employ a different utility function and our model can be directly applied.

constraints and define the Lagrangian:

$$\begin{aligned} L(\boldsymbol{x},\boldsymbol{y}) = &\sum_{i=1}^{N}\sum_{j=1}^{N_i}\Big(U_{ij}(x_{ij}) - V_{ij}(y_{ij}) - \theta_{ij}(x_{ij}-y_{ji})\Big) - \\ &\sum_{i=1}^{N}\Big(\mu_i(x_{ii}+\sum_{j=1}^{N_i}x_{ij}-1) - \phi_i(\lambda_i\rho_i x_{ii} + \sum_{j=1}^{N_i} y_{ij}\lambda_j\rho_i - H_i)\Big) - \\ &\sum_{i=1}^{N}\Big(\psi_i(\sum_{j=1}^{N_i}\frac{x_{ij}\lambda_i s_i}{c_{ij}} + \frac{y_{ij}\lambda_j s_j}{c_{ji}} - 1) - \pi_i(E_i(\boldsymbol{x}_i,\boldsymbol{y}_i) - P_i)\Big) \end{aligned}$$

where $\phi_i, \psi_i, \pi_i, \mu_i,\ \theta_{ij},\ \forall(i,j)$ are the non-negative dual variables. We denote $\boldsymbol{\omega}$ the $1\times LN^4$ vector of all duals and write the KKT:

$$U'_{ij} = \pi_i e^t_{ij}\lambda_i s_i + \frac{\psi_i\lambda_i s_i}{c_{ij}} + \mu_i + \theta_{ij} \triangleq m_{ij}(\boldsymbol{\omega}),\ \forall(i,j)\in\mathcal{L} \tag{4.8}$$

$$U'_{ii} = \phi_i\lambda_i\rho_i + \pi_i\lambda_i\rho_i e^c_i + \mu_i \triangleq m_{ii}(\boldsymbol{\omega}), \qquad \forall i\in\mathcal{N} \tag{4.9}$$

$$V'_{ij} = \theta_{ji} - \phi_i\lambda_j\rho_i - \frac{\psi_i\lambda_j s_j}{c_{ji}} - \pi_i\lambda_j(e^r_{ij}s_j + e^c_i\rho_i) \triangleq n_{ij}(\boldsymbol{\omega})\forall(i,j) \tag{4.10}$$

$$\phi_i(\lambda_i\rho_i x_{ii} + \sum_{j=1}^{N_i} y_{ij}\lambda_j\rho_i - H_i) = 0, \qquad \forall i\in\mathcal{N} \tag{4.11}$$

$$\psi_i\Big(\sum_{j=1}^{N_i}\frac{x_{ij}\lambda_i s_i}{c_{ij}} + \sum_{j=1}^{N_i}\frac{y_{ij}\lambda_j s_j}{c_{ji}} - 1\Big) = 0, \quad \forall i\in\mathcal{N} \tag{4.12}$$

$$\pi_i(E_i(\boldsymbol{x}_i,\boldsymbol{y}_i) - P_i) = 0, \quad \forall i\in\mathcal{N} \tag{4.13}$$

$$\theta_{ij}(x_{ij} - y_{ji}) = 0, \qquad \forall(i,j)\in\mathcal{L} \tag{4.14}$$

$$\pi_i,\ \psi_i,\ \phi_i, \mu_i, \theta_{ij} \geq 0, \qquad \forall i\in\mathcal{N}, (i,j)\in\mathcal{L}. \tag{4.15}$$

where we simplified notation by omitting the optimality superscript $(*)$, and introducing functions $m_{ij}(\cdot), n_{ij}(\cdot)$.

Given the KKT conditions, the broker (here, the gateway) would be able to solve the IoT-OP problem if it had access to parameters $\boldsymbol{\alpha} = (\alpha_{ij}, (i,j)\in\mathcal{L})$, $\boldsymbol{\beta} = (\beta_{ij}, (i,j)\in\mathcal{L})$, $\boldsymbol{\gamma} = (\gamma_{ij}, (i,j)\in\mathcal{L})$. However, most often this information is known only to nodes, which have many reasons not to reveal or misreport these parameters. This is an important problem for (almost) the entire spectrum of cooperative networks where nodes can benefit by overstating their needs and costs. To overcome this obstacle we propose a scalable cooperation algorithm that elicits this hidden information.

4.4 Auction Algorithm Design

In our auction the IoT gateway acts as an auctioneer (broker) and sets the task allocation and pricing rules. The nodes submit offer/ask bids to indicate their outsourcing and reimbursement requirements, and the gateway determines the task allocation. These steps are repeated until an equilibrium is reached. The key idea is to find the rules that drive these interactions to the socially-optimal point.

4.4.1 Finding an Equivalent Problem

Our algorithm belongs to the class of Walrasian auctions, and hence works under the assumption that nodes are *price-takers* [126]. This means that they do not anticipate the impact of their bids on other nodes' strategy (a strict assumption that is used in auction theory), but instead react to prices announced by the auctioneer. This model is valid when nodes have limited capacity or time to dwell in extensive strategy designs, or when there is a large number of nodes, each one with infinitesimal impact on others strategy. Both of these conditions appear in our setting, similarly to [123–125] and others.

Allocation Rules. Let us denote with $p_{ij} \geq 0$ the bid node i submits to declare its interest in outsourcing tasks to $j \in \mathcal{N}_i$, and $q_{ij} \geq 0$ the bid that signals the reimbursement i requests for accepting j's tasks. The gateway collects all these bids and solves the following optimization problem to find the current allocations.

IoT Auction problem (IoT-AP):

$$\max_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{i=1}^{N} \sum_{j=1}^{N_i} p_{ij} \log(x_{ij}+1) - \frac{q_{ij}}{2} y_{ij}^2 \tag{4.16}$$

$$\text{s.t.} \quad (4.3), (4.4), (4.5), (4.6), (4.7) \tag{4.17}$$

IoT-OP and IoT-AP are identical except the objective. The latter has been selected towards extending the seminal pertinent one-side auction mechanism in [123] (see also [127]) while

preserving concavity. Hence, the KKT optimality conditions for IoT-AP are:

$$\frac{p_{ij}}{x_{ij}+1} = m_{ij}(\boldsymbol{\omega}),\ \frac{p_{ii}}{x_{ii}+1} = m_{ii}(\boldsymbol{\omega}),\ q_{ij}y_{ij} = n_{ij}(\boldsymbol{\omega}),\ \forall(i,j) \tag{4.18}$$
$$\text{and } (4.11), (4.12), (4.13), (4.14), (4.15)\,.$$

Now, the important remark is the following: by comparing the KKT optimality conditions for the two problems, IoT-OP and IoT-AP, we see that their solutions would be the same if (4.18) coincided with (4.8)-(4.10) (the other conditions are already identical). This, in turn, would be possible if the nodes are "driven" to bid as follows:

$$p^*_{ij} = (x^*_{ij}+1)U'_{ij}(x^*_{ij}), \quad q^*_{ij} = \frac{-V'_{ij}(y^*_{ij})}{y^*_{ij}},\ \forall i,j \in \mathcal{N}_i. \tag{4.19}$$

The goal of the gateway is to *employ a pricing mechanism for charging task outsourcing, and reimbursing task admissions, so that the nodes will be induced to bid according to this socially-optimal solution.*

Pricing Rules. In detail, let $h_i(\boldsymbol{x}_i)$ and $g_i(\boldsymbol{y}_i)$ denote the charging and reimbursement functions for node i, respectively, which depend on its decisions $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$. It is important to see that these pricing decisions depend indirectly on the nodes' bids p_{ij}, q_{ij}, through the resource allocation decisions (4.18). Given these rules, each node i solves the following optimization problem in order to find its optimal ask and offer bids for each one of its neighbors:

$$\max_{p_{ij}\geq 0}\ U_{ij}(x_{ij}(p_{ij})) - h_i(x_{ij}(p_{ij})),\ \forall(i,j)\in\mathcal{L} \tag{4.20}$$

$$\max_{q_{ij}\geq 0}\ -V_{ij}(y_{ij}(q_{ij})) + g_i(y_{ij}(q_{ij})),\ \forall(i,j)\in\mathcal{L} \tag{4.21}$$

and these yield the conditions (first-order criterion):

$$\frac{\partial U_{ij}(x_{ij})}{\partial x_{ij}}\frac{\partial x_{ij}}{\partial p_{ij}} = \frac{\partial h_i(x_{ij})}{\partial p_{ij}},\ \frac{\partial V_{ij}(y_{ij})}{\partial y_{ij}}\frac{\partial y_{ij}}{\partial q_{ij}} = \frac{\partial g_i(y_{ij})}{\partial q_{ij}}, \tag{4.22}$$

and then from (4.18) we obtain the derivatives of x_{ij} and y_{ij}

$$\frac{\partial x_{ij}}{\partial p_{ij}} = \frac{1}{m_{ij}(\boldsymbol{\omega})}, \quad \frac{\partial y_{ij}}{\partial q_{ij}} = \frac{-n_{ij}(\boldsymbol{\omega})}{q^2_{ij}} \tag{4.23}$$

Algorithm 2 Coop-IoT

Input: $G=(\mathcal{N},\mathcal{L}), \boldsymbol{H}, \boldsymbol{P}, \boldsymbol{c}, \boldsymbol{\lambda}, \boldsymbol{w}$
Output: $\boldsymbol{x}^*$, $\boldsymbol{y}^*$, $h_i(\boldsymbol{p}_i)$, $g_i(\boldsymbol{q}_i)$.
Init/lize: $\boldsymbol{x}^{(0)}$, $\boldsymbol{y}^{(0)}$, $\boldsymbol{\phi}^{(0)}$, $\boldsymbol{\psi}^{(0)}$, $\boldsymbol{\pi}^{(0)}$, $\boldsymbol{\mu}^{(0)}$, $\boldsymbol{\theta}^{(0)}$, $t, cnvg \leftarrow 0$
while $cnvg = 0$ **do**
  $t \leftarrow t+1$
  Each node i bids $(\boldsymbol{p}_i^{(t)}, \boldsymbol{q}_i^{(t)})$ by solving (4.20)-(4.21);
  Broker collects all nodes' bids $(\boldsymbol{p}^{(t)}, \boldsymbol{q}^{(t)})$;
  Broker calculates new $\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}$ using (4.18)
  if $|x_{ij}^{(t)} - x_{ij}^{(t-1)}| \leq \epsilon$ and $|y_{ij}^{(t)} - y_{ij}^{(t-1)}| \leq \epsilon$, $\forall (i,j) \in \mathcal{L}$ **then**
    $cnvg \leftarrow 1$; % broker checks convergence.
  end if
  Broker charges prices $h_i(\boldsymbol{x}_i^{(t)}), g_i(\boldsymbol{y}_i^{(t)})$, $\forall i$, with (4.24).
  Broker updates dual variables, $\forall i \in \mathcal{N}, (i,j) \in \mathcal{L}$ with:

$$\mu_i^{(t+1)} = \Big(\mu_i^{(t)} + s_t\big(x_{ii}^{(t)} + \sum_{j=1}^{N_i} x_{ij}^{(t)} - 1\big)\Big)^+$$

$$\theta_{ij}^{(t+1)} = \Big(\theta_{ij}^{(t)} + s_t\big(x_{ij}^{(t)} - y_{ji}^{(t)}\big)\Big)^+$$

$$\phi_i^{(t+1)} = \Big(\phi_i^{(t)} + s_t\big(\lambda_i \rho_i x_{ii}^{(t)} + \sum_{j=1}^{N_i} y_{ij}^{(t)} \lambda_j \rho_i - Hi\big)\Big)^+$$

$$\psi_i^{(t+1)} = \Big(\psi_i^{(t)} + s_t\big(\sum_{j=1}^{N_i} \frac{x_{ij}^{(t)} \lambda_i s_i}{c_{ij}} + \frac{y_{ij}^{(t)} \lambda_j s_j}{c_{ji}} - 1\big)\Big)^+$$

$$\pi_i^{(t+1)} = \Big(\pi_i^{(t)} + s_t\big(E_i(\boldsymbol{x}_i^{(t)}, \boldsymbol{y}_i^{(t)}) - P_i\big)\Big)^+$$

  Broker transmits all dual variables to the $\mathcal{N}$ nodes.
end while

Now, recall that for the solution of IoT-OP the derivative of U_{ij} and V_{ij} are given by (4.8)-(4.10). Plugging these and (4.22) to (4.23), we obtain the *socially-optimal pricing rules* $\forall (i,j) \in \mathcal{L}$:

$$h'_{ij}(p_{ij}) = 1, \quad g'_{ij}(q_{ij}) = \frac{-\big(n_{ij}(\boldsymbol{\omega})\big)^2}{q_{ij}^2} \quad \Rightarrow$$

$$h_{ij}(p_{ij}) = p_{ij}, \quad g_{ij}(q_{ij}) = \frac{\big(n_{ij}(\boldsymbol{\omega})\big)^2}{q_{ij}}. \tag{4.24}$$

These pricing rules drive the system to the socially optimal solution and are intuitive. Firstly, the nodes pay exactly the amount they bid (p_{ij}). Secondly, using the bidding-allocation relation,

(4.18), the reimbursement rule can be written:

$$g_{ij}(q_{ij}) = y_{ij}(- \theta_{ji} + \phi_i \lambda_j \rho_i + \frac{\psi_i \lambda_j s_j}{c_{ji}} + \pi_i \lambda_j (e^r_{ij} s_j + e^c_i \rho_i)),$$

which states that i is reimbursed based on the service it offers y_{ij}, and its resources' congestion (links, energy, and computing) which is measured by the dual variables — acting here as *shadow prices*.

4.4.2 Algorithm and Properties

The mechanism is executed iteratively implementing a primal-dual algorithm for IoT-OP, where for the subgradient calculation (i.e., the primal update) we leverage the auction; see Algorithm 2. It takes as input the network configuration, node features and demand (step 1), and outputs the cooperation policy and prices (step 2). The algorithm runs until convergence. In each iteration every node announces its offer and ask bids that optimize its utility (step 6). The IoT gateway acts as broker, collecting the bids (step 7), calculating the new allocations (step 8) and checking the termination criterion (step 9). It calculates the prices and reimbursements (step 12) and updates the dual variables (step 13), which are then transmitted to nodes (step 14). If the algorithm has not converged, it repeats.

The updates of the dual variables implement a gradient descent method, solving the dual of IoT-AP which, for proper bids, coincides with the IoT-OP as explained above. For instance, for $\mu^{(t)}$ we use:

$$\boldsymbol{\mu}^{(t+1)} = \left(\boldsymbol{\mu}^{(t)} + s_t \nabla_{\boldsymbol{\mu}} L(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}, \boldsymbol{\mu}, \boldsymbol{\theta}^{(t)}, \boldsymbol{\psi}^{(t)}, \boldsymbol{\pi}^{(t)}, \boldsymbol{\phi}^{(t)})\right)^+$$

and similarly for the other duals, where $(\cdot)^+$ denotes the projection onto the non-negative orthant and s_t is the step size. Due to the fact that the problem is convex and under the assumption that Slater's conditions are satisfied, strong duality holds and this method will yield the optimal primal solution.

Theorem 1: Algorithm 1 solves IoT-OP, achieving a budget-balanced allocation.

Proof ***(i)*** For the convergence, we extend the proof in [127]. We consider very small time slots

approximating a continuous model and employ the Lyapunov function

$$\Lambda(\boldsymbol{\omega}) = \sum_{i=1}^{N} \Big(\frac{(\mu_i - \mu_i^*)^2}{2} + \frac{(\phi_i - \phi_i^*)^2}{2} + \frac{(\psi_i - \psi_i^*)^2}{2} + \frac{(\pi_i - \pi_i^*)^2}{2} + \sum_{j=1}^{N_i} \frac{(\theta_{ij} - \theta_{ij}^*)^2}{2} \Big).$$

Taking the time derivative of this function:

$$\frac{\partial \Lambda(\boldsymbol{\omega})}{\partial t} = \frac{\partial \Lambda(\boldsymbol{\omega})}{\partial \boldsymbol{\omega}} \frac{\partial \boldsymbol{\omega}}{\partial t} \,. \tag{4.25}$$

and using the gradient updates for each dual variable:

$$\frac{\partial \mu_i}{\partial t} = \Big(x_{ii} + \sum_{j=1}^{N_i} x_{ij} - 1 \Big)_{\mu_i}^{+}, \quad \frac{\partial \theta_{ij}}{\partial t} = \Big(x_{ij} - y_{ji} \Big)_{\theta_{ij}}^{+}, \, \forall i, j$$

and similarly for the other duals, where note that the RHS of these equations is simply the partial derivative of the Lagrangian w.r.t. the dual variable. Also, here $(g)_r^+$ means that when $r > 0$ we use any g, and when $r = 0$ only non-negative g values (to ensure duals are non-negative). Now, using more compact notation (ω for all duals) we write the Lyapunov function:

$$\begin{aligned} \frac{\partial \Lambda(\boldsymbol{\omega})}{\partial t} &= \sum_{k=1}^{K} (\omega_k - \omega_k^*) \Big(\frac{\partial L(x, y, \boldsymbol{\omega})}{\partial \omega_k} \Big)_{\omega_k}^{+} \leq \sum_{k=1}^{K} (\omega_k - \omega_k^*) \frac{\partial L(x, y, \boldsymbol{\omega})}{\partial \omega_k} \\ &= \sum_{k=1}^{K} (\omega_k - \omega_k^*) \Big(\frac{\partial L(x, y, \boldsymbol{\omega})}{\partial \omega_k} - \frac{\partial L(x^*, y^*, \boldsymbol{\omega})}{\partial \omega_k} \Big) + (\omega_k - \omega_k^*) \Big(\frac{\partial L(x^*, y^*, \boldsymbol{\omega})}{\partial \omega_k} \Big) \end{aligned}$$

where the first inequality holds by the definition of the projection (see also [127, Sec. 3.4]), and then we add and subtract the Lagrange derivatives at the optimal primal point x^*, y^*. Finally, using the complementary slackness conditions (4.11)-(4.14) and eq. (4.8)-(4.10), and after restructuring we obtain:

$$\frac{\partial \Lambda(\boldsymbol{\omega})}{\partial t} \leq \sum_{i=1}^{N} \sum_{j=1}^{N} \Big((x_{ij} - x_{ij}^*) \big(\frac{\partial U_{ij}(x_{ij})}{\partial x_{ij}} - \frac{\partial U_{ij}(x_{ij}^*)}{\partial x_{ij}} \big) + (y_{ij} - y_{ij}^*) \big(\frac{\partial V_{ij}(y_{ij}^*)}{\partial y_{ij}} - \frac{\partial V_{ij}(y_{ij})}{\partial y_{ij}} \big) \Big) \leq 0 \,,$$

where the last inequality holds as $U_{ij}(x_{ij})$, $-V_{ij}(y_{ij})$ are concave.

(ii) The broker's budget is payments minus reimbursements:

$$K(\boldsymbol{p},\boldsymbol{q}) = \sum_{i=1}^{N}\sum_{j=1}^{N_i}\Big(h_{ij}(p_{ij}) - g_{ij}(q_{ij})\Big) = \sum_{i=1}^{N}\sum_{j=1}^{N_i}\Big(p^*_{ij} - \frac{n_{ij}(\boldsymbol{\omega}^*)}{q^*_{ij}}\Big)$$
$$= \sum_{i=1}^{N}\sum_{j=1}^{N_i}\Big((x^*_{ij}+1)m_{ij}(\boldsymbol{\omega}^*) - y^*_{ij}n_{ij}(\boldsymbol{\omega}^*)\Big),$$

where the last equality holds due to (4.19), (4.8), (4.10). This is a non-negative quantity (hence, we have non-negative budget): $m_{ij}(\boldsymbol{\omega})$ are always positive by (4.18), and similarly for $-n_{ij}(\boldsymbol{\omega})$ except from its component $-\theta^*_{ji}$. However, the latter can be bundled with θ^*_{ij} from $m_{ij}(\boldsymbol{\omega})$ and zeroed-out due to complementary slackness (4.14).

4.5 Model and Algorithm Extensions

Next, we discuss the model extension for the case of 2-stage analytics, and explain how it can be executed in a distributed fashion, and how it can cater for interference limitations.

Distributed Execution. Algorithm 1 requires a central node, the gateway, with access to node parameters such as their processing capacity. However, for various IoT applications we might prefer to have a fully decentralized implementation and our algorithm, unlike other auctions, can be executed in a distributed fashion with small modifications. In detail, for implementing the pricing rules $h_i(\cdot)$ and $g_i(\cdot)$, each node i can update independently its dual variables μ_i, ϕ_i, ψ_i, π_i, $\theta_{ij}, \forall j \in \mathcal{N}_i$. Note that each node will have all the necessary information for performing these iterations if all 1-hop neighbors exchange their primal and dual variables in each round. The constraints that couple the different nodes actions, namely $x_{ji} \leq y_{ij}, \forall (i,j)$ (i.e., (4.4)), can be relaxed by introducing a new dual variable which will be circulated to ensure that this constraint will be satisfied at equilibrium. Then, the nodes can directly determine their optimal bids, using eq. (4.20)-(4.21) as before, and subsequently decide their admission and outsourcing strategy based on eq. (4.18).

The algorithm is lightweight as only 1-hop message passing is required for circulating the duals and the bids among neighboring nodes. From a practical point of view, this algorithm can be implemented as a distributed protocol, e.g., similar to TCP [123, 127], where nodes observe signals (here, the duals) and adjust their resource allocation decisions, which in our scenario involve both the link bandwidth and node computing capacity.

2-Stage Analytics. Many types of analytics are performed in two stages [5]: processing the collected data (e.g., extracting features) and classifying the results. These stages can be executed in different nodes, and our model can be extended to such scenarios. We assume that each task creates s_{i1} bits to be processed at the 1st stage and s_{i2} for the 2nd stage; and similarly we define parameters ρ_{i1} and ρ_{i2} for the computing load of each stage. For every node i we denote with $x_{ijk} \in [0,1]$ the portion of generated tasks that i outsources to j (for executing stage 1) and then to k for executing stage 2. Similarly, y_{jik} denotes j's decision for executing the stage 1 for the tasks of i and then forwarding the results to k; and z_{kij} is k's decision to execute the stage 2 for tasks of i (after receiving the stage 1 output from j). Clearly, a consistent cooperation policy needs to satisfy $x_{ijk} \leq y_{jik} \leq z_{kij}$, $\forall\,(i,j),(j,k) \in \mathcal{L}$.

Using this model extension, we can redefine the utility and cost functions, and solve the 2-stage problem with a similar auction as Coop-IoT. In this case, node i receives different utility $U_{ijk}(x_{ijk})$ for every pair (j,k) of outsourcing nodes it uses. Similarly, node i incurs energy cost V_{ijk} for executing the stage 1 and transmitting the stage 2 data to another node, and J_{ijk} costs for executing the second stage. We consider here the most general case where these two cost functions can be different.[4] Following our analysis for the one-stage execution we can write the utilities and costs $\forall (i,j),(j,k) \in \mathcal{L}$ as:

$$
\begin{aligned}
U_{ijk}(x_{ijk}) &= \alpha_{ijk} \log\left(\lambda_i w_{ijk} x_{ijk} + 1\right) - \beta_{ijk}\Big(x_{ijk}\lambda_i\big(\frac{s_{i1}}{c_{ij}} + \frac{s_{i2}}{c_{jk}} + \frac{\rho_{j1}}{h_j} + \frac{\rho_{k2}}{h_k}\big)\Big)^2,\\
V_{ijk}(y_{ijk}) &= \gamma_{ijk}\Big(e^r_{ij} y_{ijk}\lambda_j s_{i1} + e^c_i y_{ijk}\lambda_j \rho_{j1} + e^t_{jk} y_{ijk}\lambda_j s_{j2}\Big)^2,\\
J_{ijk}(z_{ijk}) &= \delta_{ijk}\Big(e^r_{ki} z_{ijk}\lambda_j s_{j2} + e^c_i z_{ijk}\lambda_j \rho_{i2}\Big)^2.
\end{aligned}
$$

Parameters γ_{ijk} and δ_{ijk} capture the sensitivity of each node i for executing stage 1 or stage 2 of its neighbors' tasks. Using these functions we can now define the 2-stage problem as follows:

[4]Note that the model can be simplified and aggregate all energy costs in one function, i.e., with a common cost-sensitivity coefficient.

$$\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}{\text{maximize}} \quad & \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} U_{ijk}(x_{ijk}) - V_{ijk}(y_{ijk}) - J_{ijk}(z_{ijk}) \\
\text{subject to:} \quad & x_{iii} + \sum_{j=1}^{N_i}\sum_{k=1}^{N_j} x_{ijk} \leq 1, \;\; i \in \mathcal{N}, && (4.26) \\
& x_{ijk} \leq y_{jik}, \;\; y_{jik} \leq z_{kij}, \quad (i,j),(j,k) \in \mathcal{L}, && (4.27) \\
& \lambda_i(\rho_{i1}+\rho_{i2})x_{iii} + \sum_{j=1}^{N_i}\sum_{k=1}^{N_i} y_{ijk}\lambda_j\rho_{j1} + \sum_{k=1}^{N_i}\sum_{j=1}^{N_k} z_{ijk}\lambda_j\rho_{j2} \leq H_i, \;\; i \in \mathcal{N}, && (4.28) \\
& \sum_{j=1}^{N_i}\sum_{k=1}^{N_j} \frac{x_{ijk}\lambda_i s_{i1}}{c_{ij}} + \sum_{j=1}^{N_i}\sum_{k=1}^{N_i} \frac{y_{ijk}\lambda_j s_{j1}}{c_{ji}} + \sum_{j=1}^{N_k}\sum_{k=1}^{N_i} \frac{z_{ijk}\lambda_j s_{j2}}{c_{ki}} \leq 1, \;\; i \in \mathcal{N}, && (4.29) \\
& E_i(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i) \leq P_i, \;\; i \in \mathcal{N} && (4.30)
\end{aligned}$$

This new problem has the same structure with IoT-OP and can be solved with a similar auctioning mechanism where now each node submits two type of ask bids, one for admitting stage 1 tasks and one for stage 2 tasks. The reimbursement rule for the latter is an adaptation of the respective rule from Sec. 4.4.

Protocol interference model: In addition to half-duplex interference constraints, the IoT-OP formulation can be extended to two-hop interference constraints. According to the popular protocol interference model, a transmission over link (i,j) is successful only if all nodes in range with i or j, i.e.,

$$\mathcal{I}(i,j) = \{(c,b),(a,c) : c \in \mathcal{N}_i \cup \mathcal{N}_j, b, a \in \mathcal{N}_c\}, \tag{4.31}$$

do not transmit or receive data at that time instance. It has been shown in the seminal work [108] that the necessary and sufficient conditions for a flow-level policy such as the one we consider to respect this scheduling requirement, is to include the constraint:

$$\frac{x_{ij}\lambda_i s_i}{c_{ij}} + \frac{y_{ij}\lambda_j s_j}{c_{ji}} + \sum_{(k,m)\in\mathcal{I}(i,j)} \frac{x_{km}\lambda_k s_k}{c_{km}} + \frac{y_{km}\lambda_m s_m}{c_{mk}} \leq 1. \tag{4.32}$$

Adding (4.32), $\forall (i,j) \in \mathcal{L}$, to the constraint set of IoT-OP we will obtain a policy that accounts for interference.

Note that since this is a convex constraint, our problem preserves its properties and hence it still has a unique solution. Moreover, our Coop-IoT auction mechanism can solve this new

problem with only small changes. Namely, we will need to introduce new dual variables and additional exchanged messages in order to relax (4.32).

4.6 Performance Evaluation

We have *fully implemented* the proposed system using a face recognition app and a RPi-based wireless testbed. Our goal is to explore the impact of the system parameters on the cooperation benefits, compare our algorithm to other (non-)cooperative policies, and study also the 2-stage execution policy. Our main findings are:

- The node configuration affects significantly the analytics accuracy, execution delay, and consumed energy.
- Coop-IoT improves both the accuracy and delay, over other benchmark policies.
- Coop-IoT respects the resource constraints and expands the system's capacity, i.e., the supportable task rate.
- The cooperation benefits are more pronounced for dense and heavily loaded networks.
- The 2-stage execution adds flexibility and improves the accuracy and delay over the 1-stage policies.

4.6.1 Testbed and Evaluation Setup

Face Recognition App. We built a face recognition application using OpenCV in Python. For face detection we used Haar and LBP cascades combined with the OpenCV recognizers Local Binary Patterns Histogram (LBPH), Eigen-faces and Fisher-faces. This created 6 face detector/recognizer combinations. We used the Georgia Tech face database [128] which includes 15 pictures of 50 people. From this dataset, 14 pictures of each person were used for training the recognizers, and the 15th to test it. We created two training sets to capture task diversity at the source node. In the first, the testing images were frontal, while in the second they were tilted to the side. Using a Monsoon Power Monitor we measured the power cost of the face recognition task. Table 4.1 shows the results, where the two accuracy sub-columns correspond to the frontal and side face tests.

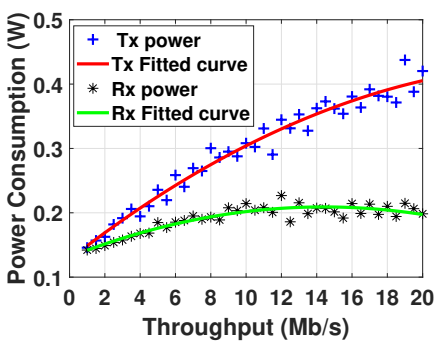
	Haar			
	Accuracy		Delay (s)	Power (W)
LBPH	82%	62%	0.76	0.28
Eigen Faces	74%	50%	0.59	0.40
Fisher Faces	42%	24%	0.41	0.25
	LBP			
LBPH	80%	56%	0.50	0.12
Eigen Faces	70%	48%	0.33	0.24
Fisher Faces	30%	18%	0.15	0.09

Table 4.1: Face recognition configurations.

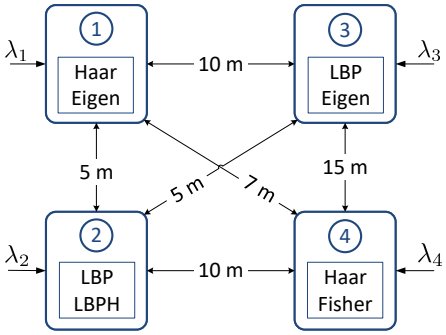
Wireless Environment. We first measured the RPis average power consumption when transmitting and receiving at different rates. We fit the collected measurements to first and second-order polynomials, where the latter offers a better result (using the AIC criterion), see Fig. 4.3a. Using this model, we can estimate the transmission and reception power consumption as a function of the data rate, and evaluate the parameters $e^t_{i,j}, e^r_{i,j}$ (which we use also in our simulations). Note that the above values generally depend on the data rate and, as usually is the case, Tx power is higher than Rx power [103]. We also measured the (average) data rate for each link in the topology of Fig. 4.3b, using a random task outsourcing policy.

We note that in the implementation we used our framework based on the primary interference model, complemented with link measurements that capture secondary interference indirectly. This was a practical design decision because it leads to a lower-complexity mechanism where nodes exchange information with only their one-hop neighbors, as opposed to two-hop neighbors in the protocol interference model. Besides, the latter yields in practice very conservative allocations as it assumes all nodes in the two-hop vicinity are blocking the transmission at the same time. Such a practical design decision has also been used in work that implements a practical distributed version of optimal Wi-Fi scheduling algorithms [129–131].

Trace-based simulation setup. For evaluations on a larger network we used Matlab. The IoT network graph is created with the random geometric graph model, that has been extensively used in simulating ad hoc or sensor networks [132]. The nodes are randomly placed in a $100m \times 100m$ area, and considered to be in communication range when their Euclidian distance is less than 50m. The link capacities in the simulations are calculated based on path loss and Rayleigh Fading, and we average the outcomes of 100 simulations for random node positions and application configurations.



(a)



(b)

Figure 4.3: (a): Tx/Rx measurements and fitted curves. (b): The RPi testbed topology for the experiments.

4.6.2 Results

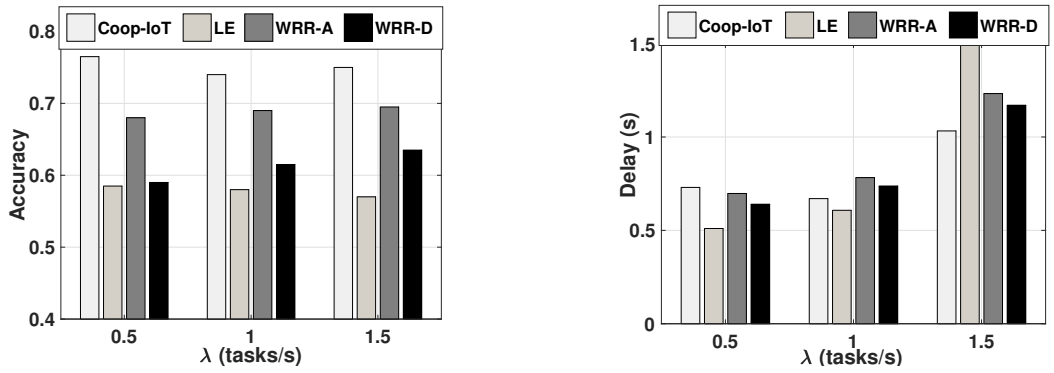
We implemented the Coop-IoT policy in our RPi testbed. We used the topology shown in Fig. 4.3b to measure the analytics performance under different task loads[5]. We compare our algorithm with a set of carefully selected benchmarks which are informed policies, more sophisticated than basic greedy approaches, namely:

- *Local execution (LE).* Each node executes its tasks locally.
- *Weighted Round Robin for Accuracy (WRR-A).* Each node outsources tasks proportionally to its neighbors' accuracy[6].
- *Weighted Round Robin for Delay (WRR-D).* Same as WRR-A but w.r.t total delay (data transmission and task execution).

Performance Comparison. We conducted our first experiment using the setup of Fig. 4.3 and a *vanilla* version of Coop-IoT that gives equal priority to accuracy and delay, i.e., $\alpha_{ij} = \beta_{ij} = 1, \forall(i,j)$. We present the results in Fig. 4.4. We find that it achieves higher accuracy than the benchmark policies (up to 17% from LE), and higher even from WRR-A that focuses only on accuracy (by up to 8%). At the same time, Coop-IoT yields low delay as well, and for high-load scenarios ($\lambda = 1.5$) it outperforms all other policies (even WRR-D, by 11.5%). It is interesting to note that for low loads, LE is the fastest policy, but this comes at the expense of accuracy. For instance, Coop-IoT sacrifices 14.9% delay performance compared to LE but

[5]For this setting, λ can reach 1.5 tasks/sec, without violating IoT-OP feasibility, which is the high task load scenario. We use higher λ values by altering the system parameters.

[6]If node 1 is neighbor with nodes 2 and 3, with $w_{12} = 2w_{13}$; WRR-A offloading decisions will be $x_{12} = 0.66$ and $x_{13} = 0.33$ (ignoring local execution).



(a) (b)

Figure 4.4: Accuracy and delay of Coop-IoT and benchmark policies, for the setup in Fig. 4.3 and selected task loads.

Description	Parameter	Value
Number of nodes	N	25
Tx/Rx energy per bit	e_{ij}^t, e_{ij}^r	1-21, 0.3-10 nJ/b
Cycles per task	ρ_j	180-912 Mcycles
CPU speed	H_j	1.2 GHz
Energy per cycle	e_j^c	0.2 nJ/c
Data per task	S_i	180 KB
Energy budget	P_j	0.3 W
Balancing parameters	$(\alpha_{ij}, \beta_{ij}, \gamma_{ij})$	(1,1,10)

Table 4.2: Simulation parameters.

gains 17% in accuracy for $\lambda = 0.5$. Note that these gains can be further increased if we modify $\boldsymbol{\alpha}, \boldsymbol{\beta}$ to prioritize accuracy or delay, as we do for the simulations. **Findings**: *Coop-IoT improves* ***concurrently*** *both performance metrics compared to the benchmarks, especially for high loads.*

Coop-IoT Agility. Next, we resort to trace-based simulations of 25-node networks, using the above measured parameters and models. Our goal is to explore the impact of load λ on the performance of Coop-IoT, compared to LE, and WRR-A/D. We employ two variations of Coop-IoT:

• *Coop-A.* The nodes apply the solution of (IoT-OP) focusing on maximizing accuracy ($\alpha_{ij} = 1,\ \beta_{ij} = 0.1\ \forall(i,j)$).

• *Coop-D.* The nodes apply the solution of (IoT-OP) focusing on minimizing delay ($\alpha_{ij} = 0.1,\ \beta_{ij} = 1\ \forall(i,j)$).

Coop-IoT not only improves the system performance as we saw in the experiment above, but additionally *it supports higher task rates* than LE, WRR-A and WRR-D which may violate the

link capacity and node energy constraints (4.5)-(4.7). In detail, Fig. 4.5a shows the percentage of nodes for which any of these constraints is not satisfied for various task loads. We observe that this percentage increases very fast, yielding completely (near 100% for WRR) constraint-violating policies in some cases. On the other hand, for all these λ values, Coop-A/D manage to optimize the analytics performance while respecting the nodes' resource availability. This highlights the importance of our *resource-adaptive* collaborative policy.

Fig. 4.5b depicts the power cost of each policy as λ increases. We see that LE has the lowest power consumption as it never offloads tasks. On the other hand, our sophisticated Coop-A and Coop-D policies have higher energy cost, but still smaller than WRR-A and WRR-D respectively, especially for high loads. This is due to the fact that they manage to satisfy the problem's constraints (by keeping power consumption low) that are often violated by those policies. **Findings**: *Coop-IoT increases the system's capacity and does not violate the constraints. At the same time, it consumes less energy than WRR-A/D but, naturally, more than the non-cooperative LE policy.*

In Fig. 4.5c we see that the average accuracy for LE and WRR is independent of the task load, and for these values of λ they already violate the capacity constraints for some nodes (see Fig. 4.5a). Our policy not only manages to respect all resource constraints, but at the same time has higher performance than the benchmark policies (even if we ignore the constraint violations). Namely, Coop-A, which prioritizes accuracy over delay, achieves up to 11% higher accuracy than LE, and 6% higher than WRR-A. For high λ values, its performance drops to satisfy the constraints, e.g., see how the power consumption of Coop-A remains lower than WRR-A for high λ. Remember that WRR-A/D and LE do not respect the link/computing capacity and power consumption constraints, and thus do not adapt their performance as λ changes.

Coop-D optimizes the execution delay as it is shown in Fig. 4.5d. Although the average delay increases with the task load, Coop-D can achieve up to 25% lower delay than LE, when the load is high. Also, Coop-D demonstrates better performance than WRR policies for most loads. In practice, heuristics like WRR might have even worse performance as the nodes will not agree to execute others' tasks, but even if we assume full cooperation still Coop-IoT achieves higher performance. **Findings**: *Coop-IoT adapts its operation to each node's performance priorities and to the available network resources, unlike the rigid benchmark policies that are resource-oblivious.*

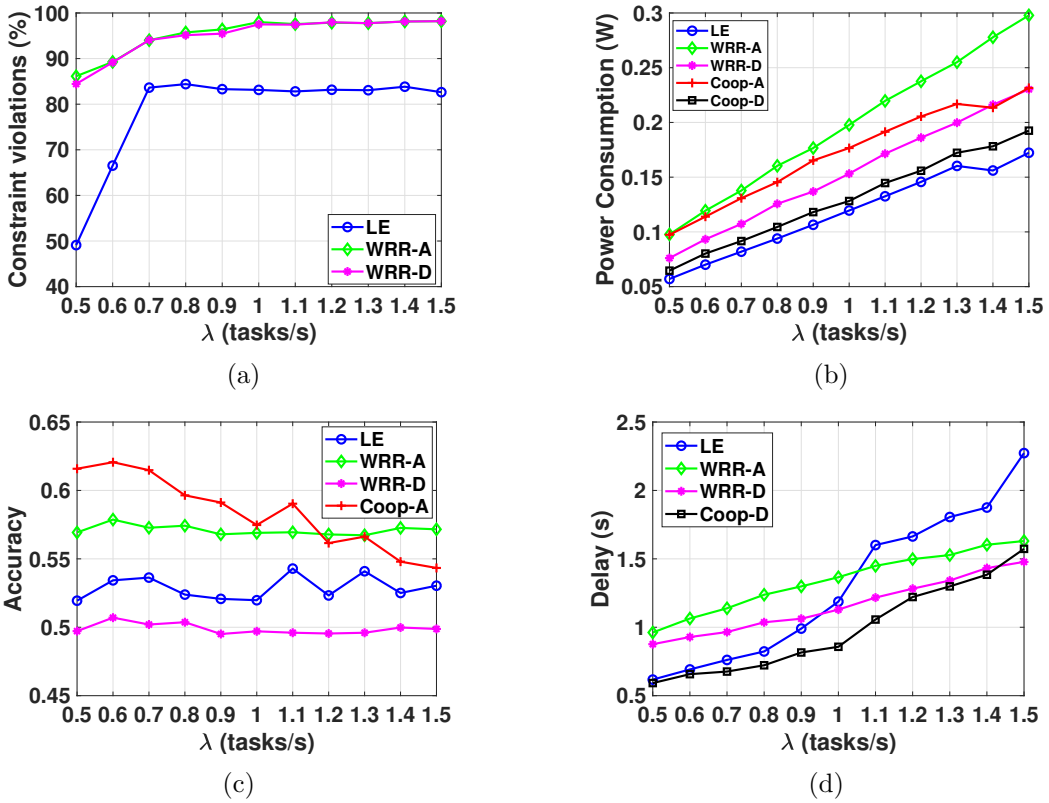
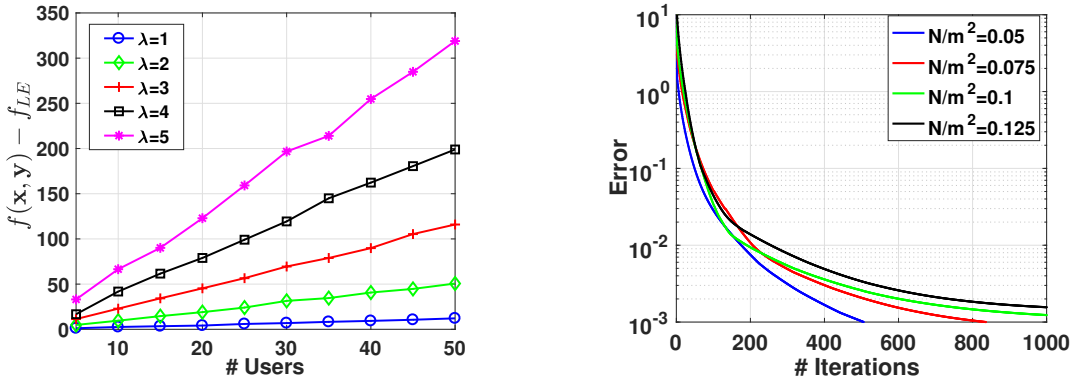


Figure 4.5: (a): Percentage of nodes that violate any of the constraints of Coop-IoT for LE and WRR policies. (b)-(d): Simulation results for power consumption, accuracy and delay.

Scaling of Cooperation Benefits. In Fig. 4.6a we present the difference of the IoT-OP solution, $f(\mathbf{x}, \mathbf{y})$, from the utility function of LE, i.e. f_{LE}. This difference quantifies the *cooperation benefits* as LE is the non-cooperation strategy. Here, we have increased the values of H_j, P_j in order to support higher λ values. As it is expected, the cooperation benefits increase with the number of nodes since more task exchange opportunities arise in dense IoT deployments. This reveals the importance of designing a scalable algorithm (as Coop-IoT is) that can enable large numbers of nodes to cooperate. Interestingly, these benefits are larger for heavily loaded networks. For instance, when the nodes increase from 10 to 50, $f(\mathbf{x}, \mathbf{y}) - f_{LE}$ increases about fivefold for any value of λ. Similarly, for a given number of users, increasing the load from $\lambda = 1$ to 5 increases also the cooperation benefits. **Findings**: *The cooperation benefits of Coop-IoT increase fast with the number of nodes, even more so for high load scenarios.*

Convergence & Scalability. In Fig. 5.3 we evaluate the convergence of Coop-IoT for different network density values, which is defined as the ratio of users over the area in which



(a) (b)

Figure 4.6: (a): Cooperation benefits $f(\boldsymbol{x}, \boldsymbol{y}) - f_{LE}$ vs the number of users, for different values of λ (tasks per node). (b): Convergence of Coop-IoT, with step size $s_t = 10^{-4}$.

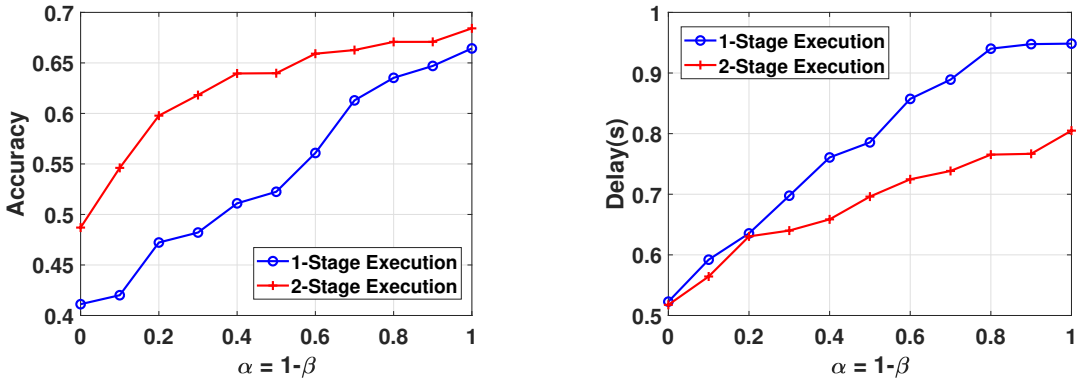
	Haar	LBP	LBPH	Eigen	Fisher
Delay (s)	0.38	0.12	0.38	0.21	0.03
Power (W)	0.22	0.06	0.06	0.18	0.03

Table 4.3: Delay and power consumption measurements for the stages of detection and recognition.

they are spread (N/m^2). Clearly, in denser networks there are more connections among the nodes, and hence their outsourcing decisions are more intertwined. This, inevitably increases the number of iterations required for the algorithm;s convergence, as can be seen in Fig. 4.6b. However, still *the algorithm can reach sufficient convergence accuracy of* 10^{-2} *within less than 200 iterations in any case*, and this demonstrates the inherent scalability of Coop-IoT.

2-Stage Analytics. Finally, since this is the first 2-stage analytics optimization proposal, we implemented and evaluated a 2-stage task execution policy for our face recognition application, and compared its performance with the respective single-stage policy (both subtasks run in the same node). First, it is necessary to measure the separate performance of the stages of detection and recognition. Table 4.3 shows the delay and power required by each detector and recognizer.

Using these measurements, we can evaluate parameters ρ_{i1}, ρ_{i2} for the 2-stage execution model. Let us focus on Fig. 4.7. As $\boldsymbol{\alpha}$ increases (and $\boldsymbol{\beta}$ decreases) the nodes' priority is to optimize accuracy rather than delay, and vice versa. It is evident that the 2-stage analytics provides a more flexible way of executing the tasks by exploiting the different properties of the available detectors and recognizers installed at each node. This yields significantly improved performance (about 10% for accuracy and 16% for delay), especially at extreme cases, i.e. low



(a) (b)

Figure 4.7: Comparison between 1 & 2-stage execution ($N = 25, \lambda_i = 1$).

$\boldsymbol{\alpha}$ values for accuracy (Fig. 4.7a) and high $\boldsymbol{\alpha}$ values for delay. **Findings**: *2-stage policies are more flexible in exploiting the nodes configuration, and improve performance compared to 1-stage execution.*

4.7 Conclusions

This chapter deals with device cooperation incentive misalignment that leads IoT devices to purposefully misreport their system parameters at the cost of others. The proposed auction framework manages to find a socially optimal task offloading policy that forces the devices to fairly bid for others' resources. In the following chapter, we study how to optimally schedule task offloadings and executions in a dynamic environment with interfering links.

Chapter 5

Dynamic Scheduling for IoT Analytics

Exploiting edge devices for collaborative execution of analytics, requires making fast scheduling decisions, i.e. which network links will be activated and which nodes should execute each task. Finding an efficient scheduling policy for such a heterogeneous network is not an easy task. The nodes produce a time-varying, often non-iid traffic load, since the latter is often event-driven, eg. a motion-sensor/camera node that starts recording upon detecting movement. Moreover, data analytics services are unique, in the way that their performance metric is also dynamic, e.g. the recognition accuracy of an object recognizer varies over time, as the classifier can be presented with more samples from other external sources.

In light of these new intricacies, we revisit the classic problem of scheduling both task transmissions, i.e. link activation, and task processing at the heterogeneous IoT nodes. Face recognition tasks are generated in the network, and the goal is to formulate a dynamic scheduling policy that respects interference constraints and optimizes the service's aggregate performance, considering the network's transmission and processing capacities. Such a solution can be applied to many existing services, e.g. [37, 38], that target computing at the edge for IoT nodes and improve their performance.

5.1 Motivation and Related Work

Unlike other state-of-the-art solutions based on max-weight scheduling like e.g. the drift-plus-penalty (DPP) algorithm [133], our solution is designed to adapt not only to varying traffic load, but also to a varying objective function, which is ideally suited for analytics. Furthermore, our proposed algorithm (FWDS) is designed towards having robust complexity per schedule evaluation, since it only solves a linear program in each iteration, even if the objective function is convex. This is achieved by using the Frank-Wolfe (FW) algorithm [16], and effectively linearizing the objective function, yielding schedules that are feasible, i.e they do not violate interference constraints, and are efficiently evaluated. To deal with the rest of networking and processing capacity constraints, we employ a dual method that relaxes them, and use the FW algorithm for the update of the primal variables (scheduling decisions). In contrast, max-weight based solutions either need to solve a (usually convex) admission control problem before max-weight scheduling [134], or assume a-priori knowledge of the set of interference-free schedules [9–12].

The main advantages of FWDS over other wireless network control algorithms are that it: *(i)* offers deterministic non-asymptotic bounds, i.e., hold per sample path; *(ii)* supports serving non-i.i.d. or non-Markovian task arrivals; and *(iii)* can handle changing objective functions.

Network control techniques were revolutionized by the seminal backpressure and maxweight stability algorithms of Tassiulas and Ephremides, [135, 136]. These works spurred a flurry of related efforts [137] and were eventually extended by Neely et al with the Drift-Plus-Penalty (DPP) algorithm that optimizes any convex objective, thus enabling the design of cross-layer algorithms, cf. [134]. DPP algorithms *run in two stages*. First, they select a continuous action, e.g., data admission, by solving the convex program $x_t \in \arg\max_{x \in X} V f(x) - J_t^\top x$, where J is a vector of virtual queues (modeling constraints) and $V > 0$ a penalty parameter. And second, they find a link activation schedule s_t that minimizes $\Delta Q + \Delta J$, i.e., the drift of packet and virtual queues. They are proven to converge asymptotically, and on expectation, to the optimal point while ensuring bounded constraint violation; and by tuning V we can trade-off optimality for feasibility.

Unlike the above approaches, FWDS solves only a linear program for finding implementable actions. This operation is almost identical with the schedule selection in maxweight and DPP algorithms, the only difference being that we use the Lagrangian. Hence, FWDS has the same

complexity with the respective step in those algorithms, namely it is NP-hard as it involves an exponential number of constraints[1]; but overall FWDS is simpler as it does not require solving additionally a convex program (first stage of DPP). Moreover, FWDS can readily include a range of techniques developed for *reducing the computation time* of maxweight/DPP algorithms, e.g., using pick-and-compare matchings, see discussion in [134, Ch. 7] and [138]; or for *enabling their distributed execution* through message passing or randomization, cf. [139]. Many of these suggestions can be directly applied to FWDS, and this is a promising direction for future work.

In a different line of work, Stolyar proposed the Greedy-Primal Dual (GPD) algorithm in [9] which selects control actions *greedily* by solving $s_t \in \arg\min_{s\in S} (\nabla f(x_t) + \beta Q_t \Delta Q)^\top s$. The continuous variables are updated with $x_{t+1} = (1-\beta)x_t + \beta s_t$. GPD is shown to converge asymptotically, but we note that it uses exponential averaging $\{s_\tau\}_{\tau=1}^t$ and hence does not ensure ergodic convergence. Moreover, the objective is fixed and known, and the data arrivals are constant or follow i.i.d. processes. Our approach is inspired from [140] which we extend to include perturbations and to obtain discrete decisions through the Frank-Wolfe operation. Finally, we note the similarities of our work with the recent works in [141, 142]. In [141] the authors propose a greedy/FW algorithm to solve deterministic convex programmes; while [142] uses DPP with FW updates, yet the convergence bounds are not deterministic and do not handle time-varying objective functions.

The works in [14] and [15] propose max-weight-based scheduling algorithms for processing networks. They perform utility maximization scheduling by controlling the amount of data that is admitted to the network towards providing queue stability. [43] studies the joint task and network flow allocation towards overall task completion time minimization, for an edge computing network. [143] presents a task scheduling framework for a fog network, where the IoT devices generate tasks with deadlines. The authors formulate a knapsack problem and propose an algorithm to maximize the service provider's revenues.

Contrary to [14] and [15], our solution optimizes a possibly time-varying task utility for the nodes, on top of satisfying stability, while [43] does not embed routing, and a separate algorithm is required. Many related works [54,143], do not consider interference constraints that complicate scheduling. On the other hand, our solution deals with interference without heavily increasing the complexity of the proposed algorithm by utilizing fast FW primal updates.

[1]In fact the complexity of this program depends on the interference model. For 1-hop interference model it has polynomial complexity, but for other models it is as hard as the Weighted Maximum Independent Set problem.

Finally, in-network processing gains increasing attention today because it can effectively support latency-sensitive data-heavy applications, and there are several proposals for joint routing and computing policies [115, 144–146]. For example, [144] was among the first works to apply a maxweight policy for routing and processing, focusing on big data applications. Similarly, [145] designs *stability* policies for wired networks with i.i.d. arrivals; while our previous work [146] optimizes the network lifetime assuming, however, interference-free transmissions. The current work builds on and expands these efforts to account for interference, varying network state and objectives, and non-i.i.d. task requests.

5.2 Model and Problem Setup

5.2.1 Network Model

We consider a wireless IoT network that connects a set $\mathcal{N}=\{i_1, i_2 \ldots, i_N\}$ of possibly heterogeneous nodes through a set $\mathcal{E}=\{(i,j) \mid i,j \in \mathcal{N}\}$ of E links, see Fig. 5.1. The system operation is time-slotted and w.l.o.g. we assume that slots have unit duration. The nodes communicate through one or more hops to jointly execute a set $\mathcal{C}=\{c_1, c_2, \ldots, c_C\}$ of data analytic tasks. For example, c_1 may correspond to image classification, c_2 to filtering data collected by sensors, etc. A task can be executed at the nodes which generate the data or elsewhere in the network. This decision depends on the computing and network resources, and the performance, i.e. the accuracy with which each task is performed at each node.

Each node i injects $b_{i,t}^{(c)} \geq 0$ data of task (or commodity) c into the network in slot t, following a stochastic process $\{b_{i,t}^{(c)}\}_{t=1}^{\infty}$ with average value of $b_i^{(c)}$ bits/sec. The capacity of each link (i,j) may vary over time, $\{\mu_{ij,t}\}_{t=1}^{\infty}$, with average value μ_{ij} bits/sec. Similarly, every node i has a possibly time-varying processing capacity $\{\pi_{i,t}\}_{t=1}^{\infty}$ cycles/sec, and π_i is its long-term average. These random variables are uniformly upper-bounded by b_{max}, μ_{max} and π_{max}, respectively. Finally, parameter $\rho_i^{(c)}$ is the processing load of i, in cycles/bit, for tasks of type c. We consider the general case where $\rho_i^{(c)}$ may vary across nodes since they might have different hardware or analytic algorithms.

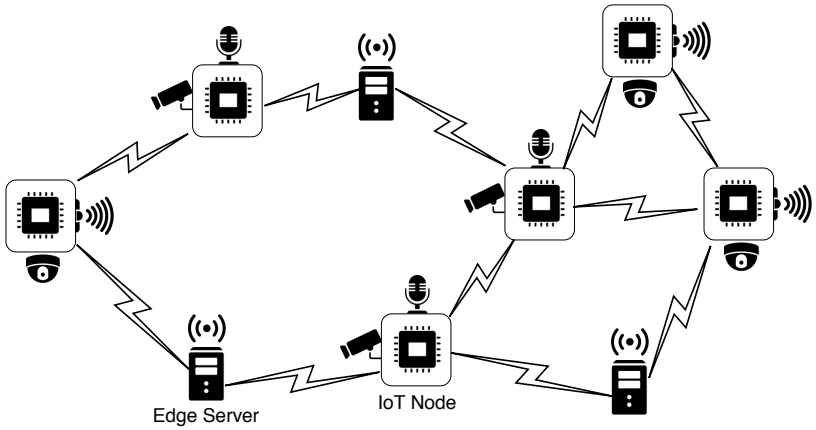


Figure 5.1: An IoT network with heterogeneous nodes running various types of analytic tasks with different accuracy.

5.2.2 Variables and Constraints

We begin by describing the static operation of the system. Variables $y_{ij}^{(c)}, z_i^{(c)} \geq 0$ denote the *average* rate (bits/sec) at which commodity $c \in \mathcal{C}$ is transmitted over link (i,j) and processed at node i, respectively. These should satisfy:

$$b_i^{(c)} + \sum_{j \in \mathcal{N}_i} y_{ji}^{(c)} = \sum_{j \in \mathcal{N}_i} y_{ij}^{(c)} + z_i^{(c)}, \quad i \in \mathcal{N}, c \in \mathcal{C}, \tag{5.1}$$

where $\mathcal{N}_i = \{j \in \mathcal{N} \mid (i,j) \in \mathcal{E}\}$ are the one-hop neighbors of node i. Eq. (5.1) ensures that the incoming and locally-generated data are equal to the outgoing and locally-processed[2] data. Also, the data routed over each link should not exceed its capacity:

$$\sum_{c \in \mathcal{C}} y_{ij}^{(c)} \leq \mu_{ij}, \quad (i,j) \in \mathcal{E}, \tag{5.2}$$

and the processing is constrained by the computing capacity:

$$\sum_{c \in \mathcal{C}} \rho_i^{(c)} z_i^{(c)} \leq \pi_i, \quad i \in \mathcal{N}. \tag{5.3}$$

The transmissions are subject to interference, which is captured using the 2-hop interference model [147]. Hence, if link (i,j) is active, no neighbor of i can receive data, and no neighbor of j

[2]Processing is technically equivalent to "extracting" data from the network, as we obtain a certain utility and terminate their routing.

can transmit. This requirement leads to the necessary and sufficient conditions for a transmission policy to be implementable [148]:

$$\sum_{c\in\mathcal{C}} \frac{y_{ij}^{(c)}}{\mu_{max}} + \sum_{c\in\mathcal{C}} \sum_{(k,l)\in I(i,j)} \frac{y_{kl}^{(c)}}{\mu_{max}} \leq 1,\ \forall (i,j) \in \mathcal{E}, \tag{5.4}$$

which captures the fraction of time that link (i,j) can be active with respect to its interfering links:

$$I(i,j) = \{(m,k),(l,m),\ k \in \mathcal{N}_i, l \in \mathcal{N}_j\}.$$

Note that the interference sets $I(i,j), \forall (i,j) \in \mathcal{E}$ are assumed fixed, but we do allow changes in link capacities over time.

5.2.3 Network Control Problem and Challenges

Let $U_i^{(c)}(z_i^{(c)})$ be a utility function that expresses the reward derived from processing tasks $c \in \mathcal{C}$ at node i. For example, this function quantifies the benefits from making a successful image classification, or prediction using sensor measurements. It is a continuous and increasing function of $z_i^{(c)}$, and modulated by parameters $w_i^{(c)}$ which capture the performance offered by each node. In practice, it is unknown and expresses the average reward of process $\{w_{i,t}^{(c)}\}_{t=1}^{\infty}$. We select $U_i^{(c)}(z_i^{(c)}) = w_i^{(c)} z_i^{(c)}$, but our framework can be readily applied to other functions. We define the *Transmission and Computation Rate Allocation* problem:

$$\text{TCRA}: \quad \underset{\{y_{ij}^{(c)}, z_i^{(c)}\}\geq 0}{\text{maximize}} \quad \sum_{i\in\mathcal{N}} \sum_{c\in\mathcal{C}} U_i^{(c)}(z_i^{(c)})$$
$$\text{subject to}: \quad (5.1) - (5.4)$$

This is a convex program that, in theory, could be solved with off-the-shelf solvers. However, in most practical systems this is not possible for the following reasons:

• (R1): The task requests $\{b_{i,t}^{(c)}\}$, link capacities $\{\mu_{ij,t}\}$, and rewards $\{w_{i,t}^{(c)}\}$ are time-varying, with unknown mean values.

• (R2): Its solution is not directly implementable, because it would create collisions between interfering nodes, i.e., it is optimal only on average.

In the following, we define the set of implementable schedules that can be selected during a scheduling slot.

Let us elaborate on (R2). While (5.4) captures the *time-average* interference coupling, in each slot the network can support only one of the eligible *link-processing schedules*, i.e.:

$$S = \Big\{ (y_{ij}^{(c)}, z_i^{(c)}) \mid \mathbb{I}(\Psi_{ij}) + \sum_{(k,l)\in I(i,j)} \mathbb{I}(\Psi_{kl}) \leq 1 \Big\},$$

where $\Psi_{ij} = \sum_c y_{ij}^{(c)}$, and $\mathbb{I}(x) = 1$ if $x > 0$ and 0 otherwise. The optimal solution of TCRA does not necessarily belong to S, but the elements of S satisfy (5.4), namely:

Proposition 2 (Link-Processing schedules) *Any average schedule that satisfies* (5.4) *belongs to conv*(S).

This means that we can select a series of schedules, the convex combination of which approximates the optimal solution of TCRA. The caveat however is that finding this "time-sharing" is a computationally cumbersome problem (see Sec. 5.5) and presumes knowing the system parameters.

We present next a practical *online* algorithm that makes *implementable* scheduling decisions, i.e., the selected schedules belong to S, and converges to the optimal TCRA solution without knowing the system statistics, hence overcoming both issues (R1) and (R2).

5.3 Reformulation and Dynamic Problem

We first reformulate the problem to streamline its presentation, explain technically how (R1)-(R2) hinder its solution, and introduce the t-slot problems that we will be using to solve TCRA dynamically. We use superscripts "$^\circ$" and "*" for the optimal solution of the static and t-slot problems, respectively.

5.3.1 Preliminaries

We first define the bounded polytopes:

$$Y = \{ y_{ij}^{(c)} \in [0, \mu_{max}] \mid (5.4),\ c \in \mathcal{C},\ (i,j) \in \mathcal{E} \},\ \text{and}$$
$$Z = \{ z_i^{(c)} \in [0, \pi_{max}/\rho_i^{(c)}],\ c \in \mathcal{C},\ i \in \mathcal{N} \}.$$

and express our variables in a single set $X = Y \times Z \subseteq \mathbf{R}_+^e$, with $e = C^2NE$. Vector $x_t = (y_{ij,t}^{(c)}, z_{i,t}^{(c)})$ is the amount of transmitted and processed data in slot t. We assume that TCRA has a finite optimal solution[3] x° (*Assumption 1*) which is a mild assumption as data can be dropped locally at the nodes that generate them, with zero utility.

We write constraints (5.1)-(5.3) in form[4] $Ax + \delta \leq 0$, where $A \in \mathbf{R}^{m \times e}$, $\delta \in \mathbf{R}^m$, $m = (CN+E+N)$; and define:

$$f(x) = -\sum_{i \in N} \sum_{c \in C} U_i^{(c)}(z_i^{(c)}) = -w^\top x,$$

where we used vector notation to represent the sum for all nodes and commodities. We wish to stress that our framework can handle also general convex functions (as it will be evident from the analysis below), and hence one can use, for instance, $f(x) = -\log(w^\top x)$ to enforce some type of fairness or load-balancing. We can now rewrite TCRA as:

$$\texttt{Primal:} \qquad \underset{x \in X}{\text{minimize}} \quad f(x) \quad \text{s.t.} \quad g(x) = Ax + \delta \preceq 0,$$

and the respective dual (concave) problem is:

$$\texttt{Dual:} \qquad \underset{\lambda \succeq 0}{\text{maximize}} \quad h(\lambda) \triangleq \min_{x \in X} L(x, \lambda),$$

where $\lambda \in \mathbf{R}_+^m$ are the dual variables and $L(x, \lambda) = f(x) + \lambda^\top (Ax + \delta)$ is the Lagrangian. This problem can be solved with the subgradient method, which consists of the update:

$$\lambda_{t+1} = \left[\lambda_t + \alpha h'(\lambda_t)\right]^+, \tag{5.5}$$

where $\alpha > 0$ is the step size and $h'(\lambda)$ is a subgradient in the subdifferential $\partial h(\lambda_t)$ of h at λ_t, that is given by:

$$x_t \in \arg\min_{x \in X} \left\{f(x) + \lambda_t^\top g(x)\right\}. \tag{5.6}$$

[3]This in practice means that the rates $b_i^{(c)}$ are such that they lie in the capacity region of the system; but one can modify slightly eq. (1) by introducing a variable that dictates how much data are dropped locally at each node, and thus drop this assumption.

[4]Each equality constraint in (5.1) can be replaced by two inequality constraints that should be satisfied concurrently; alternatively we can use lagrange multipliers that can take negative values.

One could use (5.5)-(5.6) to gradually approach the optimal TCRA solution $f(x^\circ)$; or even apply the obtained $\{x_t\}_t$ as they are generated in each iteration, see [140]. However, due to (R1)-(R2) either approach is not possible here, as we have time-varying parameters and (5.6) might yield $x_t \notin S$.

5.3.2 The t-slot Problem

Our strategy is to employ the t-slot version of TCRA that uses only information that is available up to t. In detail, we define the functions $f_t(x) = -w_t^\top x, \forall t$, where we assume that the sequence of observed precisions w_t converges to the final precision parameters of TCRA, i.e., $\lim_{t\to\infty} w_{i,t}^{(c)} = w_i^{(c)} \forall i, c$, (*Assumption 2*). This is because, as the nodes collect more data, they eventually achieve their maximum possible analytic performance.[5] Also, the changes across functions are bounded, i.e., $|f_{t+1}(x) - f_t(x)| \leq \sigma_f,\ \forall t, x$. Similarly, we define:

$$g_t(x) = Ax + \bar{\delta}_t, \ \text{ with } \ \bar{\delta}_t = \sum_{\tau=1}^{t} \delta_\tau / t$$

where the perturbations (link/node capacities, and arrivals) are assumed to converge, i.e. $\lim_{t\to\infty} \bar{\delta}_t = \delta$ (*Assumption 3*). Then, we can define the t-slot Lagrangian and dual problem:

$$\underset{\lambda_t \succeq 0}{\text{maximize}}\ h_t(\lambda_t) \triangleq \min_{x\in X} L_t(x, \lambda_t) = \min_{x\in X} \{f_t(x) + \lambda_t^\top g_t(x)\}.$$

In the sequel, we use these t-slot problems and functions to create a sequence of implementable decisions $\{s_t\}_t \in \mathcal{S}$ that ensure near optimal (average) performance, i.e.,

$$\frac{1}{t}\sum_{\tau=1}^{t} f_\tau(s_\tau) \longrightarrow f(x^\circ) \quad \text{and} \quad \frac{1}{t}\sum_{\tau=1}^{t} g_\tau(s_\tau) \longrightarrow g(x^\circ),$$

under *Assumptions 1-3*, that hold in most practical systems.

5.4 Online Optimization Framework

We propose to combine the Frank-Wolfe (FW) algorithm [16] with the approximate dual subgradient algorithm [149]. FW yields implementable actions by solving a Linear Program

[5]For classification tasks, for instance, the transmitted images can be used to re-train/update the classifiers, achieving eventually their limiting classification error; see also the discussion in Section 5.5.

Algorithm 3 FW with constant step and changing objective

1: **Set:** $\beta \in (0, 1]$ and $x_1 \in X$.
2: **for** $t = 1, 2, \ldots,$ **do**
3: $\quad s_t \leftarrow u \in \arg\min_{u \in X} \; u^\top \nabla f_t(x_t)$
4: $\quad x_{t+1} \leftarrow (1 - \beta)x_t + \beta s_t$
5: **end for**

(LP), and the dual subgradient algorithm handles the perturbed constraints. We synthesize these methods via the use of "errors" at the time to compute subgradients of the Lagrangian.

5.4.1 The Building Algorithmic Blocks

5.4.1.1 Frank-Wolfe algorithm

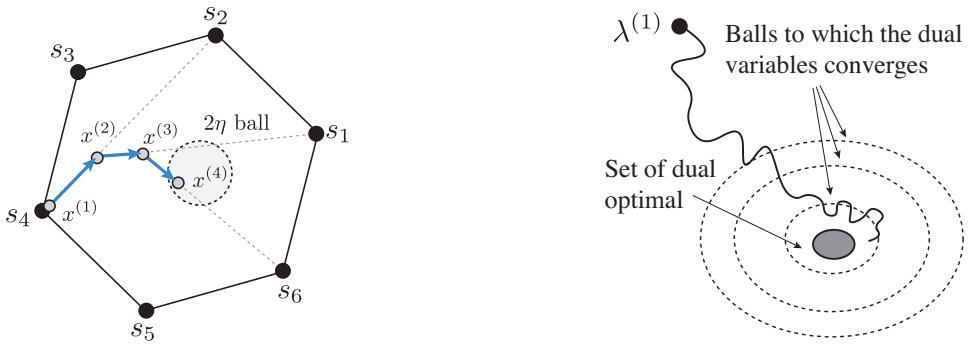
FW is a projection-free algorithm for unconstrained convex problems with smooth objective.[6] It was designed to solve static problems (fixed f) by making convex combinations of actions [150]. In our previous work [141], we showed[7] that FW converges also when we have a sequence of functions $\{f_t\}_t$, in the sense that, as t increases, it manages to reduce the gap $|f_t(x_t) - f_t(x_t^*)|$ until it becomes smaller than a set bound 2η. The steps are detailed in Algorithm 3 and presented schematically in Fig. 5.2(a). In each iteration, we compute the gradient of f_t at x_t and select a vector s_t that minimizes $u^\top \nabla f(x_t)$, w.r.t u (Step 3). Then, we update the running average x_{t+1}, where $\beta \in (0, 1]$ is a selected parameter, and repeat the process until convergence. FW is useful for our problem due to the following lemma.

Lemma 1 (Discrete Actions) *The FW minimization step 3 in Algorithm 3 yields an eligible schedule in S.*

Proof First, recall that solving a LP over a polytope results to an extreme point [149, Prop. 3.4.2]. By Proposition 2, conv(S) satisfies (5.4). The same holds for X by definition, and the two sets are identical. Since the FW step yields an extreme point of X [150], we conclude that it is also a point in S, i.e., an implementable action.

[6] A convex function f is M-smooth if there exists a constant $M \geq 0$ such that $f(v) \leq f(u) + \nabla f(u)^\top (v - u) + 2^{-1} M \|v - u\|^2$ for all $u, v \in X$.

[7] Similar results were developed in the context of online convex optimization where FW is applied to sequence of changing functions; albeit the performance criterion is the ergodic convergence and not $|f_t(x_t) - f_t(x_t^*)|$.



(a) Frank-Wolfe (b) Dual subgradient

Figure 5.2: (a): In each iteration we select an extreme point and move the average x_t in that direction. (b): Convergence of the dual subgradient algorithm.

Algorithm 4 Approximate Dual Subgradient Method

1: **Set:** $\alpha > 0$ and $\lambda_1 = 0$
2: **for** $t = 1, 2, \ldots$ **do**
3: $x_t \leftarrow u \in \mathcal{X}(\gamma_t, \lambda_t)$
4: $\lambda_{t+1} \leftarrow [\lambda_t + \alpha g_t(x_t)]^+$
5: **end for**

Lemma 1 implies that we do not need to test each schedule in S, but instead we can directly minimize $u^\top \nabla_x f_t(x)$. The basic idea is that in each iteration of Algorithm 3, either the value of the function at x_{t+1} decreases, i.e., $f_{t+1}(x_{t+1}) < f_{t+1}(x_t)$; or converges to a 2η ball around the optimum, i.e., $f_{t+1}(x_{t+1}) - f_{t+1}(x_t^*) \leq 2\eta$.

5.4.1.2 The Approximate Dual Subgradient Method (ADSM)

This algorithm leverages the dual functions $\{h_t\}_t$ to maximize the time-average $\sum_{\tau=1}^{t} f_\tau(x_\tau)/t$ which, based on *Assumption 2*, is equivalent in limit to the static TCRA. See Algorithm 4 and the example in Fig. 5.2(b). In each slot, we select x_t as:

$$\mathcal{X}(\gamma_t, \lambda_t) := \{x \in X \mid h_t(\lambda_t) \leq L_t(x, \lambda_t) \leq h_t(\lambda_t) + \gamma_t\}$$

This approximate minimization of the Lagrangian $L_t(\cdot, \lambda_t)$ amounts to using ϵ-subgradients [149, pp. 235] with upper bound of error γ_t in each slot. The algorithm ensures bounded optimality and constraint violation:

Table 5.1: Constants and Bounds

Parameter	Definition
σ_f	$\lvert f_{t+1}(x) - f_t(x)\rvert \leq \sigma_f, \forall x \in X, t$
σ_g	$\lVert g_t(x)\rVert \leq \sigma_g, \forall x \in X, t$
σ_L	$\lvert h_{t+1}(\lambda_{t+1}) - h_t(\lambda_t)\rvert \leq \sigma_L, \forall \lambda_t$
R	$\lVert x - y\rVert \leq R, \forall x, y \in X$
Λ_t	Upper bound on $\lVert\lambda_t\rVert$; see Lemma 5.
Λ	$\Lambda = \max_t \Lambda_t$
Perturbations	$\epsilon_t = w_t - w,\ \ \phi_t = \bar{\delta}_t - \delta$
Assumption 2	$\lim_{t\to\infty} w_{i,t}^{(c)} = w_i^{(c)},\ \forall i, c$
Assumption 3	$\lim_{t\to\infty} \bar{\delta}_t = \delta$

Lemma 2 (Approximate dual subgradient method) *Algorithm 4 achieves the following bounds:*

$$\text{(i)}\ \frac{1}{t}\sum_{\tau=1}^{t} f_\tau(x_\tau) - f(x^\circ) \leq \frac{\alpha\sigma_g^2}{2} + \frac{1}{t}\sum_{\tau=1}^{t} \gamma_\tau + \epsilon_\tau^\top z_\tau + \lambda_\tau^\top \phi_\tau$$

$$\text{(ii)}\ \lVert\frac{1}{t}\sum_{\tau=1}^{t} g_\tau(x_\tau)\rVert \leq \frac{\Lambda_{t+1}}{\alpha t},$$

where $z_t = \arg\min_{x\in\mathcal{X}} f(x) + \lambda_t^\top g(x), \epsilon_\tau = w_\tau - w$, $\phi_\tau = \bar{\delta}_\tau - \delta$, *and see also Table 5.1.*

We explain next how we can combine these two algorithms to solve TCRA in an online fashion.

5.4.2 Online Approximate Scheduling Algorithm

The key idea is to apply Algorithm 1 in the t-slot Lagrangian (instead of f_t) and use the dual subgradient Algorithm 2 with the t-slot constraints. The steps are shown in Algorithm 5. We first set the design parameters β and α, and initialize the variables (Step 1). In each slot t, we observe the current f_t and g_t, construct the t-slot Lagrangian $L_t(v_t, \lambda_t)$ and solve an LP to find the schedule s_t (Step 3). As explained above, this ensures that $s_t \in \mathcal{S}$ since changing from f_t to L_t does not affect Lemma 1. We can therefore implement directly this schedule (Step 4). Next we update the average value of the primal variables (Step 5) and perform a subgradient update for the dual variables (Step 6). In each slot t we only use information that has been made available up to that slot.

Algorithm 5 Frank-Wolfe Dual Subgradient (FWDS)

Set: $\beta \in (0,1]$, $\alpha > 0, v_1 \in X$ and $\lambda_1 = 0$.
for $t = 1, 2, \ldots$ **do**
  $s_t \leftarrow \arg\min_{u \in X} \; u^\top \nabla_v L_t(v_t, \lambda_t)$
  Implement schedule $s_t = (y_{ij}^{(c)}, z_i^{(c)}) \in S$
  $v_{t+1} \leftarrow (1-\beta)v_t + \beta s_t$
  $\lambda_{t+1} \leftarrow [\lambda_t + \alpha g_t(v_t)]^+$
end for

Algorithm 5 generates a sequence of vectors $\{s_\tau\}_{\tau=1}^t$ which ensure that the system operation approaches the performance prescribed by the solution of TCRA. The following theorem formalizes the performance of FWDS.

Theorem 1: Convergence of FWDS

Theorem 1 *Consider the updates in Algorithm 5 with α and β selected as per Lemma 6 (see Sec.10.1). It holds:*

$$\text{(i)} \; \frac{1}{t}\sum_{\tau=1}^{t} f_\tau(s_\tau) - f(x^\circ) \leq \frac{\alpha\sigma_g^2}{2} + 2\Lambda\sigma_g + \frac{1}{t}\sum_{\tau=1}^{t}\Big(\epsilon_\tau^\top z_\tau + \lambda_\tau^\top \phi_\tau + \max\{\zeta_\tau, 2\eta\}\Big)$$

$$\text{(ii)} \; \|\frac{1}{t}\sum_{\tau=1}^{t} g_\tau(s_\tau)\| \leq \frac{\Lambda_{t+1}}{\alpha t}(1 + \frac{1}{\beta})$$

where $\zeta_t = L_{t+1}(v_t, \lambda_{t+1}) - h_{t+1}(\lambda_{t+1})$.

Discussion. The algorithm's performance depends on parameters α, β and η, which need to be carefully selected, namely:

$$0 < \beta \leq 1, \quad 0 < \alpha < \frac{\beta\eta - (\beta^2 M_L R^2)/2 - \sigma_f - \sigma_L - 2\Lambda\sigma_g}{\sigma_g^2};$$

see Lemma 6 for the proof and Table 5.1 for the parameters.[8] It is evident that α negatively affects the optimality gap, while both α and β do not impact the constraint violation as they are amortized by t. Parameter β however, affects η in the selection of α (which must be positive), and can thus indirectly decrease the optimality gap, see Lemma 6. One needs to select these parameters based on the desirable system performance, e.g., whether the priority lies in execution accuracy, or in reducing the backlogs (i.e., delay).

[8]Similarly to the definition of M, $L_t(\cdot, \lambda)$ is a M_L-smooth function where $L_t(v, \lambda) \leq L_t(u, \lambda) + \nabla L_t(u, \lambda)^\top (v - u) + 2^{-1} M_L \|v - u\|^2$ for all $u, v \in X, t$.

Let us now elaborate on the bounds of the Theorem. Starting with (i), the first term can be made arbitrarily small by selecting α. The first term in the summation is upper bounded by $|\epsilon_\tau^\top z_\tau| \leq \|\epsilon_\tau\|\|z_\tau\|$ where $\|\epsilon_\tau\|$ diminishes since $w_t \to w$. And in the same way we can show that the second term diminishes as long as the dual variables $\|\lambda_t\|$ are bounded, which is proven in Lemma 5 of the Appendix. The last term in the summation depends on parameter η and is affected by α and β, as explained above, while ζ_t is a decreasing sequence as shown in Lemma 6; hence the term $\max\{\zeta_\tau, 2\eta\}$ is replaced by 2η for τ sufficiently large. Regarding the feasibility bound, we showed that it diminishes with α, t, but we have used the running average of perturbations on g_t, i.e. $\bar{\delta}_t$, and not δ_t. However, it is easy to see that we can write

$$\|\frac{1}{t}\sum_{\tau=1}^{t} As_\tau + \delta_\tau\| \leq \|\frac{1}{t}\sum_{\tau=1}^{t} As_\tau + \bar{\delta}_\tau\| + \|\frac{1}{t}\sum_{\tau=1}^{t} \delta_\tau - \bar{\delta}_\tau\|.$$

This adds the residual $\|\frac{1}{t}\sum_{\tau=1}^{t} \delta_\tau - \bar{\delta}_\tau\|$, which is a decreasing with t sequence, since as more input samples are added, the running average will converge to the true average $\bar{\delta}_t$.

Finally, in terms of overheads and comlexity, the bad news is that FWDS requires the solution of an NP-hard problem, but the good news is it has still smaller complexity than the state-of-the-art network control algorithms. In particular, while Steps 4-6 involve simple calculations that can be executed in polynomial time and in a decentralized fashion; Step 3 requires the centralized solution of a problem that is at least as hard as the *weighted maximum independent set* problem. This issue, however, arises in all wireless scheduling algorithms, cf. [134], – we elaborate in Section 5.1 – and the various proposals for reducing the complexity or enabling its distributed solution, at the expense of achievable throughput or increased backlogs, apply also to FWDS. Moreover, in many IoT networks that include small nodes, there is already provision for central gateways or cluster-head nodes, which can undertake the role of executing these computations.

5.5 Performance Evaluation

We have utilized a wireless testbed comprised of small IoT and edge nodes to evaluate the performance of an image classification application. The operation of the proposed FWDS algorithm is simulated and compared against several benchmarks, with respect to complexity, utility performance and network delay; and for random networks generating non-i.i.d. traffic

Table 5.2: Application parameters.

Algorithm/Node	**Accuracy**	**Cycles/bit**
LBPH/Edge	82 %	670
Eigen/Edge	74 %	520
Fisher/IoT	42 %	360

loads. We first explain the setup of the experiments and then present the results.

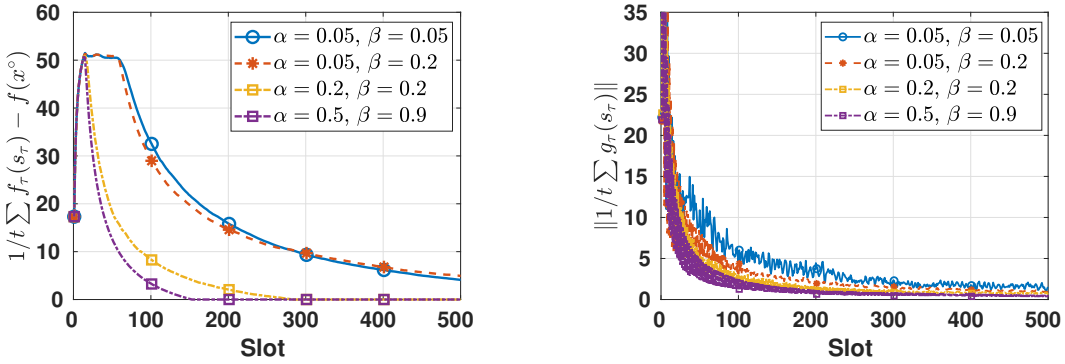
5.5.1 Experimental Setup

We simulate the operation of an IoT network of wireless cameras where the nodes run face recognition tasks. We used random geometric graphs to create the networks, i.e., we place randomly the nodes in a 100×100 meters area and connect two nodes if their euclidean distance is less than 30 meters. There are two type of nodes, namely Raspberry Pis and edge nodes[9], that differ on their computation and memory capacity.[10] The nodes create images to be classified using a typical face recognition application developed in Python with OpenCV. As mentioned earlier, face recognition is an ideal example for edge analytics as the generated data load will create communication delays that are comparable to the high computation delays of IoT analytics. We applied 3 different algorithms on the Georgia Tech face database [128] that has 750 images.

We performed initial measurements in order to calculate the various system parameters. First, we measured the average image size of the dataset and the average execution time for each of the algorithms and type of node. Based on the CPU frequency of the nodes we measured the average $\rho_i^{(c)}$. We report the results, along with the achieved average accuracy $w_i^{(c)}$ of each algorithm in Table 5.2. Observe that the more accurate an algorithm, the more computationally expensive it is. Thus we assign the algorithms so that low computation power nodes (RPis) run the least demanding algorithm (Fisher), and the edge run Eigen and LBPH, respectively. We fully trained the algorithms on our dataset to obtain the reported accuracy values. Hence, in the following, we have $w_{i,t}^{(c)} = w_i^{(c)},\ \forall t$, and also $f_t(x) = f(x),\ \forall t$. This means that the bounds of Algorithm 5 still hold, but in Lemma 6 we have $\sigma_f = 0$.

[9]The RPis are Model 3B with 1.2 GHz CPU, 1GB RAM; and the edge nodes have 2.3 GHz CPU and 16 GB RAM memory.

[10]Our experimental setup can be naturally extended to include other types of nodes, e.g., smaller form-factor nodes, since it is not necessary for all nodes to run the application and they can simply create images.



(a) Optimality gap (b) Constraint violation

Figure 5.3: Convergence of FWDS with $N = 40, C = 20$.

5.5.2 Parameter Sensitivity Analysis

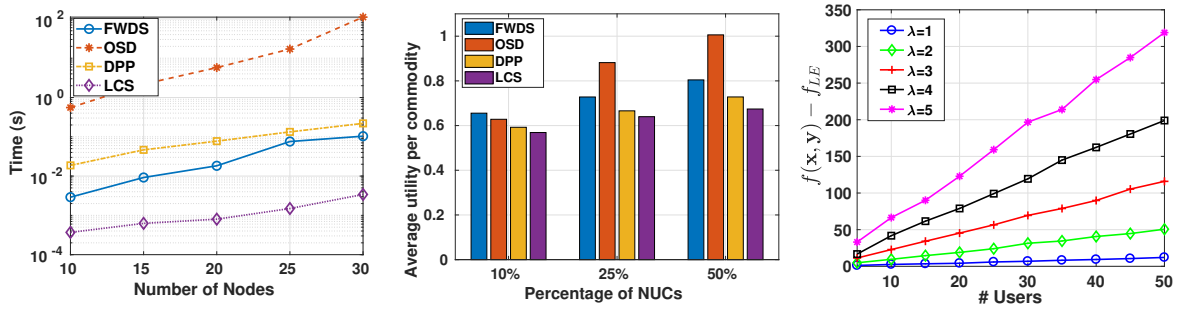
We start by exploring numerically the impact of algorithm parameters α and β, on its performance. We simulate FWDS for 500 slots and study how the optimality and feasibility bounds of Theorem 1 evolve. Fig. 5.3 presents the results for different α and β values. Note that the selected values for α and β affect the convergence bounds of FWDS both directly (see Theorem 1) and indirectly through η.

We observe that the optimality gap in Fig. 5.3a decreases with t for all cases. Notice that from the bound of Theorem 1, the gap decreases with α. However, increasing α results in slightly better convergence performance. This counter-intuitive behavior is due to the intricate relations between α, β, η. Reducing α requires decreasing β, which in turn might increase η and undermine the optimality gap. Regarding the feasibility gap in Fig. 5.3b, note that it decreases with t. Hence the differences between the cases of selected α, β are smoothed as t increases, being noticeable only in the early stages of execution. These experiments *demonstrate numerically the convergence of our algorithm and its dependence on the system parameters*, verifying our theoretical analysis.

5.5.3 Comparison with Benchmarks

We evaluate the performance of Algorithm 5 w.r.t. computation complexity for acquiring a schedule in each slot, and congestion created as a result of the scheduling decisions. We compare FWDS with the following benchmarks:

1) Optimal Solution Decomposition (OSD). We minimize $L_t(x, \lambda)$ to obtain the optimal



(a) Time delay (b) Utility for $C = 20$ (c) Utility for 10 edge nodes

Figure 5.4: (a) Delay of computing one schedule. (b)-(c) Average network utility for $N = 40$ and varying number of commodities/edge nodes.

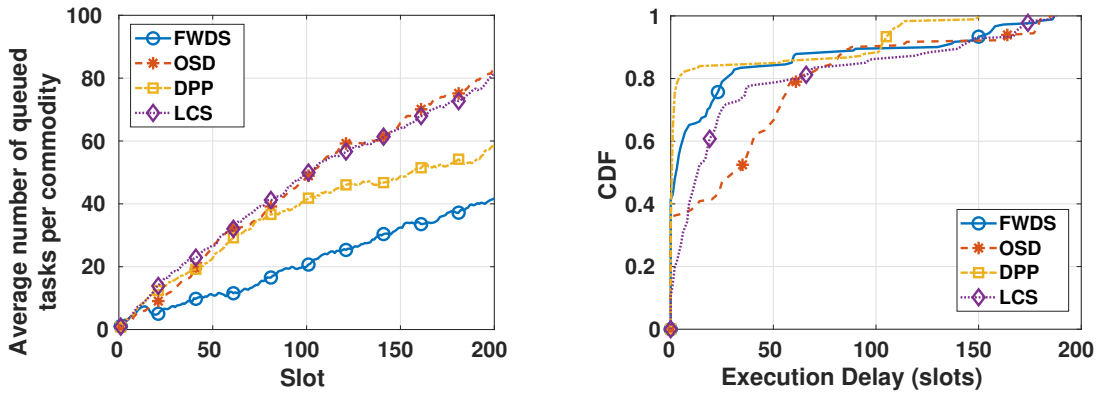
x_t^*, and we decompose it to a set of implementable schedules (time-sharing). This can be accomplished by, e.g., using the FW algorithm on $\|x - x_t^*\|^2$. Hence, we obtain a set of schedules and their associated weights, such that their convex combination approaches $-w^\top x_t^*$, and choose a schedule randomly based on its weight. Although this approach leads to optimal behavior, many times the size of the problem makes the computation of the implementable schedules very slow, and thus prohibits its implementation in practical systems.

2) Drift Plus Penalty (DPP). Based on the max-weight algorithm [134], the DPP [133] minimizes the queue backlogs while also maximizing some other performance metric for the network, which for TCRA is the aggregate node utility, multiplied by the control parameter V.

3) Linear Complexity Scheduling (LCS). Inspired by [12], we implement a simple policy where, having knowledge of the possible schedules, we pick a schedule $s \in S$ randomly. At each iteration, we pick another schedule $\hat{s}$ at random and compare $s^\top \nabla_x L_t(x_t, \lambda_t)$ to $\hat{s}^\top \nabla_x L_t(x_t, \lambda_t)$. We then apply the schedule that yields the minimum value.

Fig. 5.4a depicts the required time for each algorithm to compute the schedule in each time slot, for different network sizes. Note that this delay will impact the system's slot duration. It is evident that OSD is by orders of magnitude slower than the rest, and thus its practical application in a large IoT network is challenging if not impossible. FWDS is faster than DPP, since it does not have to solve a convex resource allocation problem (DPP's first stage). However, it is slower than the LCS algorithm but the difference is much more tractable, making it suitable for fast scheduling. In essence, *FWDS can be utilized in systems where resource allocation decisions need to be taken every few seconds.*

Next we evaluate the average network utility obtained by each algorithm (Fig. 5.4b). We



(a) Congestion (b) Task execution delay

Figure 5.5: Network congestion in tasks per commodity and task execution delay CDF with $N = 40$, $C = 20$.

consider different scenarios where we modify the number of edge nodes, in a 40-node network. The results indicate that, as expected, the average utility increases with the percentage of edge nodes. Moreover, we observe that OSD manages to clearly outperform the competition when there is more than 25% of edge nodes, while our solution has the edge over the max-weight based algorithms. Fig. 5.4c shows again the network's average utility, but this time for a fixed number of edge nodes and a varying load, as we increase the number of nodes that generate traffic, i.e. the commodities. The results, once again demonstrate that FWDS roughly maintains the same performance, despite the load increase, compared to DPP and especially LCS, while is also competitive to OSD in all load cases.

Fig. 5.5a presents the network congestion for the operation of a 40-node hybrid network, where 20 of the nodes create tasks and the rest of them act as relays and computing nodes. We see that having high utility in OSD comes at the cost of high network congestion. This is because decomposing the optimal solution to a set of feasible schedules takes many slots; hence OSD keeps implementing the same set of (outdated) schedules until the next primal update iteration. However, the dynamics of the system require better response to the implemented schedules and congestion builds up. LCS although very fast, makes random, "naive" scheduling selections in each slot and thus performs quite poorly, in both utility and congestion. On the other hand, DPP is better for both metrics, and allow us to trade-off one for the other by tuning the design parameter V. Still, *FWDS performs better than DPP in both accuracy and network congestion.*

The performance of data analytics services is also affected by their execution delay. Fig. 5.5b

displays the CDF of execution delay for the same setup as with Fig. 5.5a. Observe that the CDF of OSD and LCS lay below FWDS and DPP for most delay values. We also observe that FWDS performs slightly worse than DPP, which can be attributed to having superior utility performance. However, *FWDS and DPP achieve a delay of less than 30 slots for more than 80% of tasks.*

5.6 Conclusions

In this chapter we provided an iterative algorithm for providing feasible task offloading schedules for an IoT network. This is enabled by the linearization of the primal variable optimization step using the Frank-Wolfe algorithm, embodied in a dual subgradient method. Next, we turn our attention to centralized architectures, where the devices generating the analytic tasks exploit a single, more powerful edge server to improve their performance.

Chapter 6

Enabling Edge-Assisted Mobile Analytics

6.1 Introduction

The recent demand for mobile machine learning (ML) analytic applications, such as image recognition, natural language translation and health monitoring, has been unprecedented [5]. These services collect data streams generated by various hand-held or other IoT devices, and analyze them locally or at distant cloud servers. The challenge with such services is that they are both resource intensive and delay sensitive. On the one hand, the cloud offers more powerful ML models and abundant compute resources, but requires data transfers which consume valuable network bandwidth and device power, as well as induce significant delays (e.g., due to intermittent connectivity). On the other hand, executing these services directly at the devices, economizes network bandwidth, but degrades their performance due to the devices' limited resources. For example, these nodes may have insufficient memory to support accurate deep-learning neural networks.

A promising approach to tackle this problem is to follow a middle-ground solution where the devices outsource their tasks to nearby cloudlets [60]. These edge servers are typically deployed in locations close to cellular base stations or Wi-Fi access points, and hence are in close proximity with the users. Therefore, they can increase the service performance by *augmenting* the devices' ML components with more accurate models, while offering tolerable communication and execution delay. Nevertheless, the success of such solutions requires intelligent decision

algorithms for selecting which tasks from each device will be outsourced. This is a new problem that raises intricate challenges for the network and the involved computations.

6.1.1 Background and Motivation

Clearly, the cloudlets, unlike the cloud, have limited computing capacity and hence cannot support all the requests from all devices. If overloaded, they will eventually become unresponsive. At the same time, the task execution often involves the transfer of large data volumes (e.g., video streams). This calls for prudent transmission decisions in order to avoid wasting the energy of the devices, and congesting the network when link bandwidth is also a bottleneck. Furthermore, unlike general computation offloading solutions [25], in ML analytic services it is imperative to identify and outsource only the tasks that can significantly benefit from cloudlet execution. Otherwise, the system might as well spend resources only to gain an insignificant improvement in service performance. Finally, these decisions need to be made in a dynamic fashion, accounting for the time-varying network conditions,user requests and cloudlet availability; and the statistical properties of these random parameters are unknown in practice.

Our goal is to design and evaluate an online decision framework that supports edge-augmented mobile analytic services. While prior works have studied the problem of offloading computation-intensive tasks and others proposed system architectures for mobile analytics, we lack an analytical framework for maximizing the performance of such services under resources (un)availability, time-varying network conditions and unpredictable user requests. Our solution works under such practical limitations (which we measure experimentally), and is general enough to be applied to different architectures and services.

6.1.2 Methodology and Contributions

We consider a system where a cloudlet improves, upon request, the execution quality of data analytic tasks, e.g. classification, running on edge devices such as wireless IoT cameras or small robot nodes. Each device has a low-precision classifier, while the cloudlet can execute the task with higher precision. The devices classify the received objects upon arrival (e.g., captured images), and decide whether to transmit them to the cloudlet or not[1]. Making this decision

[1]We observed that local classifications can be much slower than offloading and executing at the cloudlet, hence the users can tolerate an extra delay in depicting the results from the cloudlet, if an offloading decision is taken.

requires an assessment of the potential performance gains, which are measured in terms of analytics accuracy improvements. To this end, we propose the usage of a *predictor* that is installed at each device and leverages the local classification results.

In terms of resource constraints, we focus on power consumption, a bottleneck issue in small devices; and the computing capacity of the cloudlet which - unlike the cloud - is finite. The former couples the decisions of each device across time, while the latter ties the decisions of all devices sharing the cloudlet. We consider the practical case where resource availability is unknown and possibly time-varying, not necessarily following an i.i.d. or Markov process, and we observe their instantaneous values. We aim to design a distributed algorithm that enables the coordination of devices and dictates the task outsourcing policy by carefully tuning the trade-off between maximizing the aggregate analytics accuracy and constraining resource consumption.

We formulate the operation of the system as an optimization program with unknown parameters appearing both in the objective (performance gains) and in the constraints (power and capacity), which are learned gradually in an online fashion. This program is decomposed via Lagrange relaxation to a set of device-specific problems, and this enables its distributed solution through an *approximate* – due to the fact that there are several unknown parameters – dual ascent method. Leveraging the ϵ-(sub)gradient information that is produced in the dual space by each different device, we calculate primal solutions which are applied in real time and implement our policy. Our method is inspired by primal averaging schemes for *static* problems, e.g., see [140, 151], and achieves a *bounded and tunable* optimality gap compared to the hypothetical average benchmark policy that has access to a complete-information oracle. Our algorithm is distributed and lightweight in terms of communication overheads, and adapts on the fly to resource availability and user requests. More importantly, it works under minimal assumptions for the stochastic perturbations of the system parameters. Namely, the requests and resource costs/availability perturbations need only to be uniformly bounded and have well-defined mean values. This is in contrast to other optimization toolboxes, see [71] and references therein. Furthermore, our performance bounds are deterministic, i.e., hold for any problem realization (sample-path) and not only in expectation, as in [71].

We extend our framework for settings where the bottleneck resource is the wireless link capacity, and for services that wish to optimize both accuracy and execution delay. Similar scenario-specific amendments are also possible (e.g., using the cloud). Given that this is a new

problem, we investigate experimentally its properties and assess the performance of our algorithm. We used a wireless testbed of Raspberry Pi devices and a cloudlet server, and evaluated an image classification service using a KNN classifier and an advanced Convolutional Neural Network (CNN) classifier. The system performance has been tested using two datasets, MNIST [152] and CIFAR-10 [153], and we compared our solution with several benchmark algorithms.

The contributions of this work can be summarized as follows:

- We introduce the novel problem of augmenting the performance of mobile analytics using edge infrastructure (e.g., cloudlets), which is increasingly important for mobile computing services and IoT networks. Our model can be tailored to different system architectures or types of analytics.
- A *distributed* task outsourcing policy is proposed that achieves near-optimal performance in a deterministic fashion, while being oblivious to the system and request statistics. To the best of our knowledge, our algorithm is the first to offer deterministic bounds (not only on expectation) under such general conditions, e.g. system perturbations are uniformly bounded and converge to some mean.
- The architecture is evaluated in a wireless testbed using a typical ML application, and several classifiers and datasets. We show that our algorithm can be indeed implemented as a lightweight protocol, and find that in practice can increase the accuracy (up to 12%) and reduce the energy costs (down to 50%) compared to several benchmark policies.

6.2 Model and Problem Formulation

We introduce the system model and the problem, and define the respective mathematical program. Table 6.1 summarizes the key notation we use throughout the chapter. We use caligraphic capital letters for sets, bold typeface letters for vectors, and $\|\cdot\|$ denotes the ℓ_2 (Euclidean) norm.

6.2.1 Classifiers and Predictors

Time is slotted and we index the slots. There is a set $\mathcal{C}$ of $C = |\mathcal{C}|$ disjoint object classes; and a set $\mathcal{N}$ of $N = |\mathcal{N}|$ devices. Each device n may receive at slot t an object s_{nt} for classification, from a set $\mathcal{S}$ of possible objects, e.g., images captured by its camera. Every device n is equipped with

Description	Parameter / Variable
Classifier of device n (cloudlet)	J_n (J_0)
Inferred class of object s_{nt} at device n (cloudlet)	$J_n(s_{nt})$ $(J_0(s_{nt}))$
Predictor of device n	Q_n
Classification confidence of n (cloudlet)	d_n (d_0)
Actual (predicted) offloading improvement	ϕ_{nt} (Q_n)
Average (instantaneous) power constraint	B_n (B_{nt})
Average (instantaneous) computing constraint	H (H_t)
Computing (Power) consumption of task s_{nt}	h_{nt} (o_{nt})
Weighted improvement gain for device n at slot t	w_{nt}
j-th interval of possible w_{nt} values (and quantized value)	$\mathcal{I}^j$ (w_n^j)
Arrival probability of object at user n in interval I^j	λ_n^j $(\lambda_{n,t}^j)$
Outsourcing probability for task in interval $\mathcal{I}^j$	$y_n^j \in [0,1]$

Table 6.1: Key Parameters and Variables.

a *local* classifier J_n which outputs the inferred class $J_n(s_{nt}) \in \mathcal{C}$ of object s_{nt} and a normalized *accuracy* (or, confidence) value $d_n(s_{nt}) \in [0,1]$ for that inference[2]. There is also a classifier at the cloudlet J_0 which can classify any object $s_{nt} \in \mathcal{S}$ with confidence $d_0(s_{nt})$. The local classifiers may have different performance due to, e.g., their different training datasets, while the cloudlet classifier has higher accuracy, $d_0(s_{nt}) \geq d_n(s_{nt}), \forall n \in \mathcal{N}$, $s_{nt} \in S$, as demonstrated also in our experiments (Sec. 8.6).

Let $\phi_{nt} \in [0,1]$ denote the accuracy improvement when the cloudlet classifier is used:

$$\phi_{nt}(s_{nt}) = d_0(s_{nt}) - d_n(s_{nt}), \quad \forall n \in \mathcal{N},\, s_{nt} \in \mathcal{S}. \tag{6.1}$$

Every device is also equipped with a predictor[3] that is trained with the outcomes of the local and cloudlet classifiers for K_n labeled items. This predictor can estimate the cloudlet's improvement for each object $Q_n(s_{nt})$, where this assessment might not be exact, i.e., $Q_n(s_{nt}) \neq \phi_{nt}(s_{nt})$, and we denote with $\sigma_{nt}(s_{nt}) \in [0,1]$ the respective normalized predictor confidence for object s_{nt}.

6.2.2 Wireless System

The devices access the cloudlet through high-capacity cellular or Wi-Fi links, see Fig. 6.1, that do not impose data transfer constraints (we relax this assumption in Sec. 6.4). Each device n

[2]The classifier might output only the class with the highest confidence, or a vector with the confidence for each class and allow the user to decide (typically selecting again the more likely class). Our analysis works for both cases.

[3]This can be a model-based or model-free solution, e.g., a regressor or a neural-network; our analysis and framework work for any of these solutions. In the implementation we used a mixed-effects regressor, see [154].

has an *average power budget* of B_n Watts that it can spend on transmitting the objects to the cloudlet. Local classifications can induce significant energy costs to the devices, but they are not considered for the budget B_n since, in our system, every object undergoes local classification and only some of the objects are further sent to cloudlet. Clearly, power consumption is a key limitation in this problem because: the devices might have a small energy budget to spend during this time interval; their small form-factor imposes power consumption limitations; there are protocol-induced transmission constraints; or users might impose constraints on the power consumption of this service. Our model follows prior studies that use average power constraints, e.g., [155], and captures these scenarios. Similarly, the cloudlet has an *average processing capacity* of H cycles/sec. This resource is shared by all devices, and when the total load exceeds its capacity the task delay increases fast, eventually rendering the system non-responsive.

We consider the realistic scenario where the parameters of devices and the cloudlet change over time in an unknown fashion. Namely, they are created by unknown random processes $\{B_{nt}\}_{t=1}^{\infty}$ and $\{H_t\}_{t=1}^{\infty}$, and our decision framework has access only to their instantaneous values in each slot. Unlike other network optimization frameworks [71], here we only ask that these perturbations are uniformly bounded, $H_t \leq H_{max}$, $B_{nt} \leq B_{max}, \forall t, n \in \mathcal{N}$, and their averages converge to some finite values which we do not require to be known, i.e., $\lim_{T\to\infty} \sum_{t=1}^{T} B_{nt}/T = B_n, \forall n$, and similarly $\lim_{T\to\infty} \sum_{t=1}^{T} H_{nt}/T = H$. We also define the vector $\boldsymbol{B}_t = (B_{nt} \leq B_{max}, n \in \mathcal{N})$.

When an image is transmitted in slot t from device n to the cloudlet, it consumes part of the device's power budget. We assume that this power cost, denoted o_{nt}, follows a random process $\{o_{nt}\}_{t=1}^{\infty}$ that is uniformly upper-bounded and has well-defined mean values[4]. Also, each transmitted object requires a number of cloudlet processing cycles, which might vary with time, e.g., due to different object types, and we assume it follows the random process $\{h_{nt}\}_{t=1}^{\infty}$, with $\lim_{T\to\infty} \sum_{t=1}^{T} h_{nt}/T = h_n$. We define $\boldsymbol{o}_t = (o_{nt} \leq o_{max}, n \in \mathcal{N})$, and $\boldsymbol{h}_t = (h_{nt} \leq h_{max},\ n \in \mathcal{N})$. Our model is general as the requests, power and computing cost per request and resource availability, can be arbitrarily time-varying and with unknown statistics.

[4]Such assumptions are minimal since, e.g. in real systems there is a maximum data rate that can be supported, inducing a certain power cost, the mean value of which, can easilly be tracked.

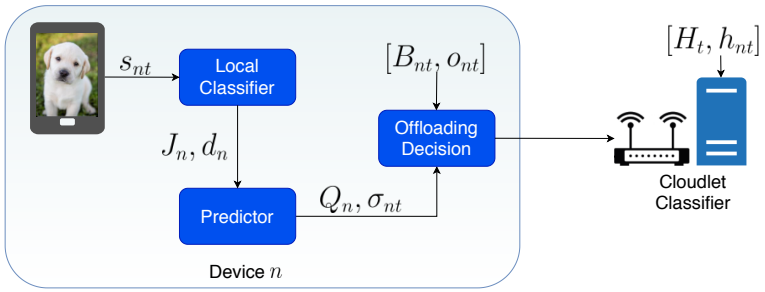


Figure 6.1: System model including the local/cloudlet classifiers and predictors. Each device is constrained by its average power budget, and the cloudlet has a limited computation capacity.

6.2.3 Problem Definition and Formulation

The devices wish to involve the cloudlet only when they expect high classification precision gain, and with high confidence. Otherwise, they will consume the cloudlet's capacity and their own power without significant benefits. Therefore, we make the outsourcing decision for each object s_{nt} based on the *weighted improvement gain*, $w_{nt}(s_{nt}) = Q(s_{nt}) - \rho_{n,th}\sigma_{nt}(s_{nt})$, $\forall\, n, t$, where σ_{nt} is the confidence, and $\rho_{n,th} \geq 0$ is a *risk aversion* parameter set by the system designer or each user. For example, assuming normal distribution for the outputs of the predictor, we could set $\rho_{n,th} = 1$ and use a threshold rule of 1 standard deviation. We use hereafter these modified parameters $w_{nt}, \forall n$, and partition the interval of their possible values $[-1, 1]$ into subintervals $\mathcal{I}^j, j = 1, \ldots, M$ such that $\cup_{j=1}^{M} \mathcal{I}^j = [-1, 1]$ and $\mathcal{I}^i \cap \mathcal{I}^j = \emptyset$ for all $i \neq j$; with w_n^j being the center point of $\mathcal{I}^j$. We denote with $\mathcal{M}$ the set of these intervals. This quantization facilitates the modeling and implementation of our algorithm (Sec. 6.5), and is without loss of generality since we can define very short intervals. Besides, many systems in practice use quantized values to measure the inference accuracy.

Let λ_{nt}^j denote the event of an object arrival at device n belonging to subinterval $\mathcal{I}^j$ at slot t. If such arrival occurs at slot t, we have $\lambda_{nt}^j = 1$; otherwise we get $\lambda_{nt}^j = 0$. We assume these *arrivals* are generated by an unknown process $\{\lambda_{nt}^j\}_{t=1}^{\infty}$, with $\lim_{T\to\infty} 1/T \sum_{t=1}^{T} \lambda_{nt}^j = \lambda_n^j, \forall n, j$, where λ_n^j denotes the probability of an object arrival belonging to subinterval $\mathcal{I}^j$, at any slot. Our goal is to maximize the aggregate long-term analytics performance gain for all objects and devices. We introduce the variables $y_n^j \in [0, 1]$, $\forall n, j$, where each of them indicates the outsourcing probability of objects in $\mathcal{I}^j$. We also define the vector $\boldsymbol{y} = (y_n^j : n = 1, \ldots, N, j = 1, \ldots, M)$ and introduce the respective set $\mathcal{Y} = [0, 1]^{NM}$. Then we formulate the following program:

$$\text{(P1)}: \quad \underset{\boldsymbol{y}\in\mathcal{Y}}{\text{minimize}} \quad -\sum_{j=1}^{M}\sum_{n=1}^{N} w_n^j \lambda_n^j y_n^j = f(\boldsymbol{y}) \tag{6.2}$$

$$\text{subject to:} \quad \sum_{j=1}^{M} y_n^j \lambda_n^j o_n \leq B_n,\ n \in \mathcal{N}, \tag{6.3}$$

$$\sum_{j=1}^{M}\sum_{n=1}^{N} y_n^j \lambda_n^j h_n \leq H. \tag{6.4}$$

Eq. (6.3) imposes the average power budget of each device, and (6.4) bounds the cloudlet utilization. Note that if we were to fully model the total power consumption of the device, we should add a term related to the computation energy cost at the LHS of (6.3). However, this term is independent of the decision variable y_n^j (since the local classifier is used either way), and thus it is omitted. Additional constraints can be included if needed; and we can also replace the linear objective with any other concave function. For instance, we might wish to enforce a fairness criterion by using a type of α-fair functions [105] or an objective that maximizes accuracy while minimizing the total delay, if the latter is also a priority. We elaborate on these extensions in Sec. 6.4.

The solution of (P1) requires to know all the parameter values, and yields the outsourcing decisions for every device n and interval $\mathcal{I}^j$ where the expected improvement w_n^j for s_{nt} lies. We assume that this problem admits a solution, which we denote with $\boldsymbol{y}^*$ and define $f^* = f(\boldsymbol{y}^*)$. This means that the system parameters and number of objects arriving at each device are such that (P1) is feasible. Then, our goal is to devise a distributed dynamic policy that manages to approach asymptotically the value f^*.

6.3 Decision Framework and Online Algorithm

We first study the system when its parameters are known, and accordingly relax this assumption and design an online algorithm that relies only on information that is available at each slot.

6.3.1 Algorithm with Complete Information

In order to solve (P1) we first dualize it and define the Lagrangian:

$$L(\boldsymbol{y},\boldsymbol{\mu}) = -\sum_{n=1}^{N}\sum_{j=1}^{M} w_n^j \lambda_n^j y_n^j + \sum_{n=1}^{N} \mu_n \Big(\sum_{j=1}^{M} o_n \lambda_n^j y_n^j - B_n\Big) + \xi\Big(\sum_{n=1}^{N}\sum_{j=1}^{M} h_n \lambda_n^j y_n^j - H\Big), \tag{6.5}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_N, \xi)$, is the vector of the non-negative dual variables for (6.3) and (6.4). This relaxation implies that the constraints need to be satisfied only on average, i.e. they might be violated in certain slots, and in practice corresponds to installing queues for transmitting the data packets and processing the images. Note that we only consider the case where problem (P2) is feasible, meaning that the aforementioned queues will not grow indefinitely, as our algorithm will prohibit the devices from sending their images to the cloudlet, and as a result, respecting the average constraints (6.3),(6.4)[5]. The dual function is defined as:

$$\text{(D)}: \qquad V(\boldsymbol{\mu}) = \min_{\boldsymbol{y}\in\mathcal{Y}} L(\boldsymbol{y},\boldsymbol{\mu}). \tag{6.6}$$

The dual problem (D) amounts to maximizing $V(\cdot)$ subject to the non-negative constraints for all dual variables. We solve this problem with an iterative dual ascend algorithm, where the iterations are in sync with the system's time slots t and are applied in real time. In other words, instead of waiting the algorithm to converge and then apply its solution, we implement the outcomes of each iteration in the same slot.

First, observe that $V(\boldsymbol{\mu})$ does not depend on B_n, H, and it is separable w.r.t. the primal variables and independent of λ_n^j. Hence, in each iteration t we can minimize the Lagrangian by:

$$(y_{nt}^j)^* \in \arg\min_{y_{nt}^j\in[0,1]} y_{nt}^j(-w_n^j + \mu_{nt} o_n + \xi_t h_n), \;\; n = 1,\ldots,N,\; j = 1,\ldots,M. \tag{6.7}$$

This yields the following easy-to-implement *threshold rule* for every device n and interval j:

$$y_{nt}^j = \begin{cases} 1 & \text{if } \lambda_{nt}^j > 0 \;\text{ and } \mu_{nt} o_n + \xi_t h_n < w_n^j \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

[5]In a more practical setup where the feasibility of (P1) is not guaranteed, the task queues at the devices could indeed grow indefinitely if the generated load λ_n^j is too large for the devices' capabilities to processes the images. In these cases, the devices can simply throttle the data generation rate by dropping the necessary number of frames.

Hence, we obtain an implementable policy when solving (P1), namely a 0-1 decision for the requests at each device. Then, we can update the dual variables for (P1) using the step $\alpha > 0$:

$$\xi_{t+1} = \left[\xi_t + \alpha \left(\sum_{n=1}^{N}\sum_{j=1}^{M} h_n \lambda_n^j y_{nt}^j - H\right)\right]^+, \quad \mu_{n,t+1} = \left[\mu_{nt} + \alpha \left(\sum_{j=1}^{M} o_n \lambda_n^j y_{nt}^j - B_n\right)\right]^+, \; n \in \mathcal{N}. \tag{6.9}$$

This iterative subgradient algorithm produces a series of primal variables y_{nt}^j which, if averaged, yield an approximate primal solution for (P1). Namely, let $\gamma = \min\{B_1, B_2, \ldots, B_N, H\}$, and $g(\boldsymbol{y}) \preceq \mathbf{0}$ to be the vector of $N+1$ constraints of (P1). We have the following Lemma:

Lemma 1. The average $\bar{\boldsymbol{y}}_T = (\frac{1}{T}\sum_{t=1}^{T} y_{nt}^j : \forall n, j)$ converges to the optimum value $f^* = f(\boldsymbol{y}^*)$ of (P1), with the following optimality gap and constraint violation bounds:

$$f(\bar{\boldsymbol{y}}_T) - f^* \leq \frac{\alpha\Big(H + \sum_{n=1}^{N} B_n\Big)}{2}$$

$$\|g(\bar{\boldsymbol{y}}_T)\| \leq \frac{1}{\alpha T}\left(\frac{-3f^*}{\gamma} + \frac{\alpha\left(H + \sum_{n=1}^{N} B_n\right)}{2\gamma} + \alpha\left(H + \sum_{n=1}^{N} B_n\right)^{1/2}\right)$$

Proof First, we define the vector of the constraint functions $g(\boldsymbol{y}) = (g_1(\boldsymbol{y}), \ldots, g_N(\boldsymbol{y}), g_{N+1}(\boldsymbol{y}))$, with $g_n(\boldsymbol{y}) = \sum_{j=1}^{M} y_n^j \lambda_n^j o_n - B_n, n = 1, \ldots, N$, and $g_{N+1}(\boldsymbol{y}) = \sum_{j=1}^{M}\sum_{n=1}^{N} y_n^j \lambda_n^j h_n - H$. These functions define a constraint set that is bounded, with an upper bound:

$$G = \max_{y \in \mathcal{Y}} \|g(\boldsymbol{y})\| = \left(H + \sum_{n=1}^{N} B_n\right) \tag{6.10}$$

which is attained for $\boldsymbol{y} = \mathbf{0}$. The latter is also a Slater vector for (P1), as it holds $g_n(\mathbf{0}) < 0$ for $n = 1, \ldots, N+1$. Now, we can use zero initial values for all dual variables and calculate the minimum constraint function for the selected Slater vector, namely $\gamma = \min_{n=1,\ldots,N}\{-g_n(\mathbf{0})\} = \min\{B_1, \ldots, B_N, H\}$. Then the Lemma follows from [17, Prop. 3].

6.3.2 Algorithm with Instantaneous Information

In most practical cases however, the above solution approach cannot be applied as the parameters and requests vary in each slot. We will be using the following approximate functions, which can

be calculated at each slot t since they are using only information that has been revealed until t:

$$\bar{f}_t(\boldsymbol{y}) = -\sum_{j=1}^{M}\sum_{n=1}^{N} w_n^j \bar{\lambda}_{nt}^j y_n^j = -\sum_{j=1}^{M}\sum_{n=1}^{N} w_n^j \lambda_n^j y_n^j + \sum_{j=1}^{M}\sum_{n=1}^{N} w_n^j y_n^j (\lambda_n^j - \bar{\lambda}_{nt}^j) = f(\boldsymbol{y}) + \boldsymbol{y}^\top \boldsymbol{\epsilon}_t$$

with $\boldsymbol{\epsilon}_t = \big(w_n^j(\lambda_n^j - \bar{\lambda}_{nt}^j), n \in \mathcal{N}, j \in \mathcal{M}\big) \in \mathbb{R}^{N \cdot M}$, and $\bar{\lambda}_{nt}^j = \sum_{\tau=1}^{t} \lambda_{n\tau}^j / t$ is the running average of arrival probabilities. We also define $\bar{g}_t(\boldsymbol{y}) = g(\boldsymbol{y}) + \delta_t(\boldsymbol{y})$, with $\delta_t(\boldsymbol{y}) = \big(\delta_{nt}(\boldsymbol{y}), n = 1, \ldots, N+1\big)$ where:

$$\delta_{nt}(\boldsymbol{y}) = B_n - \bar{B}_{nt} + \sum_{j=1}^{M} y_n^j (\bar{o}_{nt}\bar{\lambda}_{nt}^j - o_n \lambda_n^j), \forall n \in \mathcal{N},\ \delta_{N+1,t}(\boldsymbol{y}) = H - \bar{H}_t + \sum_{j=1}^{M}\sum_{n=1}^{N} y_n^j (\bar{h}_{nt}\bar{\lambda}_{nt}^j - h_n \lambda_n^j),$$

with $\bar{B}_{nt} = \frac{1}{t}\sum_{\tau=1}^{t} B_{n\tau}$, and $\bar{H}_t, \bar{o}_{nt}, \bar{h}_{nt}$ being defined similarly. Note that $\bar{f}_t(\boldsymbol{y})$ and $\bar{g}_t(\boldsymbol{y})$ can be calculated at each slot, while $f(\boldsymbol{y})$ and $g(\boldsymbol{y})$ are unknown.

Algorithm 6 OnAlgo

Initialization: $t = 0, \boldsymbol{\mu}_0 = 0,\ \forall\ n, j$
while True **do**
  for each device $n \in \mathcal{N}$ **do**
    $y_{nt}^j = 0, \forall j$
    Receive object s_{nt}
    Classify objects and obtain $J_n(s_{nt}), d_n(s_{nt}),\ \forall s_{nt} \in \mathcal{S}_{nt}$
    Use classification results on predictor to obtain $Q_n(s_{nt})$, σ_{nt}
    $(w_{nt}(s_{nt}), w_n^j) \leftarrow Q_n(s_{nt}) - \rho_{n,th}\sigma_{nt}$
    for $j = 1, \ldots, M$ **do**
      Observe λ_{nt}^j and calculate average $\bar{\lambda}_{nt}^j$
      if $\mu_{nt}\bar{o}_{nt} + \xi_t \bar{h}_{nt} < w_n^j$ **then**
        $y_{nt}^j \leftarrow 1$ % Send object to cloudlet
      end if
    end for
    Observe $o_{nt}, h_{nt}B_{nt}$, and calculate new running averages $\bar{o}_{nt}, \bar{h}_{nt}, \bar{B}_{nt}$
    $\mu_{n,t+1} \leftarrow [\mu_{nt} + \alpha(\sum_{j=1}^{M} \bar{o}_{nt}\bar{\lambda}_{nt}^j y_{nt}^j - \bar{B}_{nt})]^+, \forall n \in \mathcal{N}$
    Send $\bar{\lambda}_{nt}^j, \forall j$, to cloudlet
  end for
  Cloudlet: Compute tasks and receive $\bar{\lambda}_{nt}^j, \forall n$, from all devices
  Observe H_t and calculate new running average $\bar{H}_t$
  $\xi_{t+1} \leftarrow [\xi_t + \alpha(\sum_{n=1}^{N}\sum_{j=1}^{M} \bar{h}_{nt}\bar{\lambda}_{nt}^j y_{nt}^j - \bar{H}_t)]^+$;
  Send ξ_{t+1} to devices
  $t \leftarrow t + 1$
end while

Assumption 1 *The perturbations of all system parameters* $\{\lambda_{nt}^j, o_{nt}, h_{nt}, B_{nt}, H_t, \forall n, j, t\}$ *are independent to each other, uniformly bounded, and their averages converge to well-defined constant values, e.g.,* $\lim_{t\to\infty} \bar{B}_{nt} = B_n$.

Under this assumption, it holds that $\lim_{t\to\infty} \delta_t(\boldsymbol{y}) = \mathbf{0}$ and $\lim_{t\to\infty} \boldsymbol{y}^\top \epsilon_t = 0$ for any $\boldsymbol{y}$. Furthermore, we define the following quantities for every slot t:

$$\|g(\boldsymbol{y})\| \leq \sigma_g, \quad \|\delta_t(\boldsymbol{y})\| \leq \sigma_{\delta_t}, \quad \|\bar{g}_t(\boldsymbol{y})\| = \|g(\boldsymbol{y}) + \delta_t(\boldsymbol{y})\| \leq \sigma_g + \sigma_{\delta_t}, \tag{6.11}$$

where for the last one we have used Minkowski's inequality. These bounds are well defined due to boundedness of the parameters and the fact that $y_n^j \in [0,1], \forall n, j$. Finally, it holds that $\lim_{t\to\infty} \sigma_{\delta_t} = 0$. Following the above, we can replace (6.7), (6.9) in the dual ascent with:

$$\boldsymbol{y}_t \in \arg\min_{\boldsymbol{y}\in\mathcal{Y}} \left\{ \bar{f}_t(\boldsymbol{y}) + \boldsymbol{\mu}_t^\top \bar{g}(\boldsymbol{y}) \right\}, \quad \boldsymbol{\mu}_{t+1} = \left[\boldsymbol{\mu}_t + a\bar{g}(\boldsymbol{y}_t) \right]^+. \tag{6.12}$$

This is a modified version of the dual ascent method as we use the running averages of the system parameters instead of the static values or the per-slot instantaneous values.

We can now design our online distributed task outsourcing algorithm, which we call *OnAlgo*. The details are presented in Algorithm 1. If each device n receives an object s_{nt} in slot t, it uses its local classifier to predict its class, and the predictor to estimate the cloudlet's classification improvement (Step 5-8). Then, the device uses its threshold decision rule (Step 11) that compares the expected benefits for each interval j with the outsourcing costs for the device and the cloudlet (comparing with averages $\bar{o}_{nt}, \bar{h}_{nt}$, respectively). The devices update their running averages of the system parameters $\bar{B}_{nt}$, and the local dual variable for its power constraint violation (Step 16), and send their updated statistics to the cloudlet (Step 17). The clouldet classifies the received objects and updates its parameter estimates (Step 20) and its congestion (Step 21), which is then sent to devices.

The following Theorem characterizes the performance of the algorithm.

Theorem 1: Convergence of OnAlgo

Under Assumption 1, OnAlgo ensures the following optimality and feasibility gaps:

$$(i): \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} f(\boldsymbol{y}_t) \leq f^* + \frac{a\sigma_g^2}{2} \qquad (ii): \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} g(\boldsymbol{y}_t) \preceq 0$$

Proof See Sec. 10.2.

This theorem shows that OnAlgo achieves asymptotically, as $T \to \infty$, zero feasibility gap (no

constraint violation), and a fixed optimality gap that can be made arbitrarily small by tuning the step size. Moreover, as we prove analytically and show experimentally, smaller step sizes increase the time interval until constraints are fully satisfied. Also, observe that we do not make any assumption about the correlation and distribution of the arrivals. It is important to note that the above limits are satisfied for any sequence of random variables, i.e., every *sample path*, and not only for the average values (on expectation)[6]. Such deterministic bounds are of special interest as they offer guarantees for any problem instance and not only for their ensembles.

6.4 Model and Algorithm Extensions

We extend our framework by jointly optimizing prediction accuracy *and* total execution delay, since the latter can also be crucial for many edge services. Then, we explain how it can cope with massive demand scenarios, where the wireless bandwidth becomes a bottleneck; and finally we elaborate on alternative designs/usages of the predictor.

Joint Accuracy/Delay Optimization: We extend our model to capture both the accuracy gains and the impact of offloading decisions on delay. We do so by adding the total delay for processing the tasks of all users in the objective function and using a scaling parameter $\zeta \in [0, 1]$ to balance between the two objectives. In detail, we can express the total delay as:

$$D_{tot}(\boldsymbol{y}) = \sum_{n=1}^{N} \sum_{j=1}^{M} (1 - y_n^j) \lambda_n^j D_n^{pr} + y_n^j \lambda_n^j (D_n^{pr} + D_0^{pr} + D_n^{tr}) \tag{6.13}$$

where D_n^{pr}, D_0^{pr} are the delays for processing images at device n or the cloudlet, respectively; and D_n^{tr} the delay for transmitting images to cloudlet. These quantities can vary with time, similarly to the other system parameters, because each image has different size, or the channel conditions change. The processing delays can be modeled with linear functions, since we enforce the total processing capacity constraints. Namely, we can write $D_n^{pr} = k_n / H_{d,n}$, where k_n is the number of CPU cycles required for processing the images of device n, and $H_{d,n}$ is the processing speed of device n (cycles/sec). Similarly, we can define the processing delay at the cloudlet as $D_0^{pr} = k_n / H$ which, again, we allow to vary either because of changes in the cloudlet $\{H\}_t$ or changes in the load $\{k_n\}_t$; we refer the reader also to [88] and references therein.

[6]That is, for every sequence of observed random parameters, the performance and constraint violation comply with Theor. 1.

Regarding the transmission delay, this depends on the actual system architecture. For example, if different channels are employed for the users, we can express it as $D_n^{tr} = \ell_n/(r_n W)$, where ℓ_n is the size of each image, r_n the channel gain for user n, and W the link bandwidth. If there is a CSMA-type network where users need to share their links, we need to replace W with the actual airtime W_n that user n receives; and in the case we have a fair round-robin (vanilla version of CSMA) we can approximate this with $W_n = \sum_{j=1}^{M} y_n^j \lambda_n^j / \sum_{n=1}^{N} \sum_{j=1}^{M} y_n^j \lambda_n^j$. This model has been used extensively in Wi-Fi service allocation, see [156], and in mobile code offloading, e.g., in [88]. Note that in our online algorithmic setup, we can *track* the changes in the above delays in the same way we adapt to varying energy costs, processing loads, etc,

$$\text{(P2)}: \quad \underset{y_n^j \in [0,1]: \forall n,j}{\text{minimize}} \quad -\sum_{j=1}^{M}\sum_{n=1}^{N} w_n^j \lambda_n^j y_n^j + \zeta D_{tot}(\boldsymbol{y}) \tag{6.14}$$

$$\text{subject to:} \quad \sum_{j=1}^{M} y_n^j \lambda_n^j o_n \leq B_n,\ n \in \mathcal{N}, \tag{6.15}$$

$$\sum_{j=1}^{M}\sum_{n=1}^{N} y_n^j \lambda_n^j h_n \leq H, \tag{6.16}$$

where $\zeta \geq 0$ underpins the importance of minimizing delay in contrast to maximizing accuracy.

Following the analysis in Sec. 6.2 we can verify that the offloading rule will be:

$$y_{nt}^j = \begin{cases} 1 & \text{if } \lambda_{nt}^j > 0 \ \text{ and } \mu_{nt} o_n + \xi_t h_n < w_n^j - \zeta(D_n^{tr} + D_0^{pr}) \\ 0 & \text{otherwise.} \end{cases}, \tag{6.17}$$

where we observe that the local device execution delay is nullified since it is independent of the offloading decision, and the condition in line 12 of Algorithm 6 (OnAlgo) will be replaced by $\mu_{nt}\bar{o}_{nt} + \xi_t \bar{h}_{nt} < w_n^j - \zeta(\bar{D}_{nt}^{tr} + \bar{D}_{0t}^{pr})$, where $\bar{D}_{nt}^{tr}, \bar{D}_{0t}^{pr}$ are the running averages of the measured transmission and cloudlet computation delay respectively.

Wireless Bandwidth Constraints: We have assumed the system operation is constrained by the devices' power budget and the computing capacity of the cloudlet. Indeed, most often these are the bottleneck resources [23, 25, 60]. However, in scenarios of massive demand the wireless link capacity might also be a bottleneck constraint. Our analysis can be readily extended for this case. If we denote with $\{W_t\}_{t=1}^{\infty}$ the link capacity process (uniformly bounded; well-

defined mean value W) assuming a wireless link shared by all devices[7], we can add to (P1) the constraint:

$$\sum_{n=1}^{N}\sum_{j=1}^{M} y_n^j \lambda_n^j \ell_n \leq W, \tag{6.18}$$

where ℓ_n is the size of objects device n transmits. Eq. (6.18) can be handled as the computing constraint (6.4), and will only affect the convergence bounds. Similarly, we can include other constraints, coupling the actions of all devices $\forall t$, or separately of each device across time.

Alternative System Architectures: A different mechanism is possible, where the devices send objects to the cloudlet before using their own classifier. This approach can reduce the consumed energy, since it avoids low-accuracy local classifications. However, it requires a different type of a predictor, namely one that can estimate the expected accuracy gain using some basic features of the object (e.g., its file size), and without requiring input from the local classifier. In this case, modeling the power consumption of the devices would modify constraint (6.3) of (P1) as:

$$\sum_{j=1}^{M} \left(y_n^j \lambda_n^j o_n + (1 - y_n^j)\lambda_n^j \nu_n \right) \leq B_n, \ \ \forall n \in \mathcal{N},$$

where the second term indicates the power ν_n consumed by each device when only local classification is performed. OnAlgo can be extended to this case by changing the predictor, as long as Assumption 1 holds.

Similarly, it is possible to have services that are executed in multiple stages, e.g., a video stream is compressed, then frames of interest are selected, and objects are identified on each frame. In this case, the devices might decide to outsource some of the tasks in the first stage, some others after the second stage, and so on. Again, our optimization algorithms can be extended to include these decisions, by defining a separate set of variables for each stage while accounting for the costs and properties (e.g., data volumes) in each case. In specific, (6.3) would be transformed to:

$$\sum_{j=1}^{M} \left(y_n^j \lambda_n^j o_n + (1 - y_n^j)\lambda_n^j \nu_n^{cl} \right) \leq B_n, \ \ \forall n \in \mathcal{N},$$

where ν_n^{cl} denotes the classification computing cost, which is significantly smaller than ν_n. Observe that the computing load of stage 1, i.e. feature extraction, is not accounted for since it is again induced regardless of the offloading decision.

[7]This can be either an OFDM-based cellular link or a coordinated access WiFi link; in the case we have a CSMA-type of mechanism, one needs to account for the additional bandwidth loss due to collisions, etc.

6.5 Implementation and Evaluation

We have fully implemented the proposed architecture, evaluated OnAlgo with real datasets, and complemented our analysis with large-scale synthetic simulations. This section has four goals: *(i)* investigate the accuracy performance of well-known classifiers for different sizes of training datasets, hence revealing why the *edge augmentation* is needed; *(ii)* Measure the energy and computing costs of image classification tasks; *(iii)* Perform a parameter-sensitivity analysis of OnALgo; and *(iv)* Compare OnALgo with several benchmark algorithms.

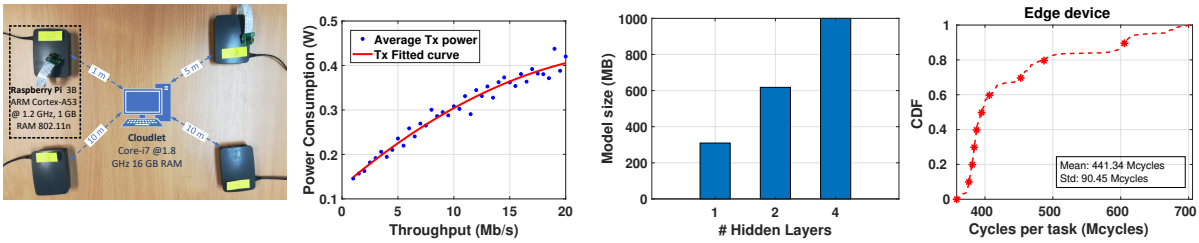
6.5.1 Experiments Setup

We used 4 Raspberry Pis (RPis) as end-nodes, and a cloudlet with specs as in [157], see Fig. 6.2a. The RPis are placed in different distances from the cloudlet, and all plots are using data from at least 50 experiments. We measured energy using a Monsoon Power Monitor, and used Python libraries and TensorFlow for the classifiers. We have used *vanilla* versions of libraries and classifiers so as to facilitate observation of the results.[8]

We measured the average power consumption when a RPi transmits with different rates, Fig. 6.2b. Then we fitted a linear regression model that estimates the consumed power (Watts) as a function of the rate r, $p(r) = -0.00037r^2 + 0.0214r + 0.1277$. This result is used by OnAlgo to estimate the energy cost for each image, given the data rate in each slot (which might differ for the RPis). Also, we measured the average computing cost (cycles/task) for the classification task for a neural network (CNN) in the RPis and cloudlet. Since the images have different sizes, we observed that the computation load varies, with a mean of 441 $Mcycles$ and std. 90 $Mcycles$ for the cloudlet (see Fig. 6.2d), and a mean of 3044 $Mcycles$ and std. 173 $Mcycles$ for RPis. Regarding the delays, we measured device and cloudlet average processing and transmission delays and found that $D_n^{pr} = 2.537$, $D_0^{pr} = 0.191$ and $D_n^{tr} = 0.157$ ms. This result suggests that local processing is about 10 times slower than offloading in our system. Hence, it is possible that the extra offloading delay experienced by the devices can be worth trading off for the enhanced accuracy of the cloudlet.

We focus on image classification, a widely employed analytic task that is known to generate big data loads that quickly consume device resources, and use two well-known data sets: *(i)*

[8]For instance, the memory footprint of NNs can be made smaller [72, 73] but such actions possibly affect their performance. Our analysis is orthogonal to such interventions.



(a) (b) (c) (d)

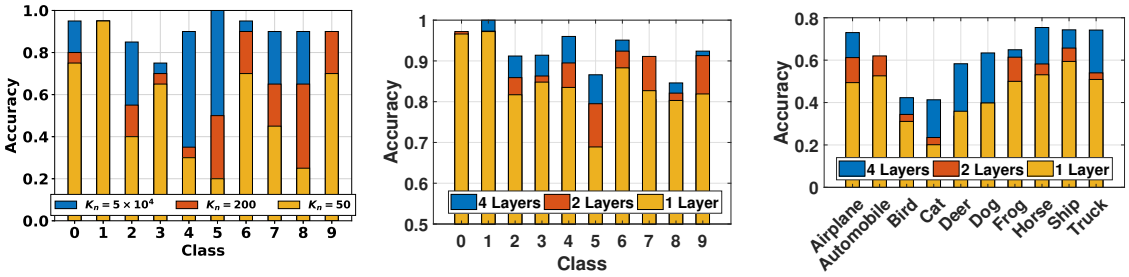
Figure 6.2: (a): Testbed: 4 RPis and a cloudlet (laptop). (b): Transmit power consumption measurements and the fitted curve for the RPis. (c) Increasing the number of layers in CNN increases the model size. (d) CDF of computing cycles per task for the cloudlet.

MNIST which consists of 28×28 pixel handwritten digits, and includes 60K training and 10K test examples; *(ii)* **CIFAR-10** that consists of 50K training and 10K test examples of 32×32 color images of 10 classes. We used two very different classifiers: the normalized-distance weighted k-nearest neighbors (KNN) algorithm [158], and the more sophisticated Convolutional Neural Network (CNN), implemented with TensorFlow [159]. Both classifiers output a vector where each coordinate represents the probability that the object belongs to the respective class. These classifiers differ substantially in their performance and resource requirements, hence allowing us to build diverse experiment scenarios. Our goal is to evaluate both and determine which one is more suitable depending on other system parameters like the number of available training samples at each location.

The predictors are trained with labeled images and the outputs of the local ($d_n(s_{nt})$) and cloudlet ($d_0(s_nt)$) classifiers. We implemented an ordinary least squares regressor and a model-free random forest that estimate ϕ_{nt} (dependent variables) based on the classifier outputs (independent variables). Recall that the dependent variables are calculated using (6.1), where we additionally use that $w_{nt} = d_0(s_{nt})$ if device n has given a wrong classification (for a labeled object) and $w_{nt} = -d_0(s_{nt})$ if the cloudlet is mistaken. We have used training sets of different sizes and two different regressors: *(i)* a general model, where the prediction does not consider the locally inferred class as an independent variable; and *(ii)* a class specific model that is based on the output of the local classifier.

We compare OnAlgo with three different algorithms:

- *Accuracy-Threshold Offloading* (ATO), where a task is offloaded when the confidence of the local classifier is below a threshold, without considering the resource consumption. This is



(a) KNN on MNIST (b) CNN on MNIST (c) CNN on CIFAR

Figure 6.3: Per class Accuracy of MNIST and CIFAR-10 for KNN and CNN classifiers of various labeled data sizes and hidden layers.

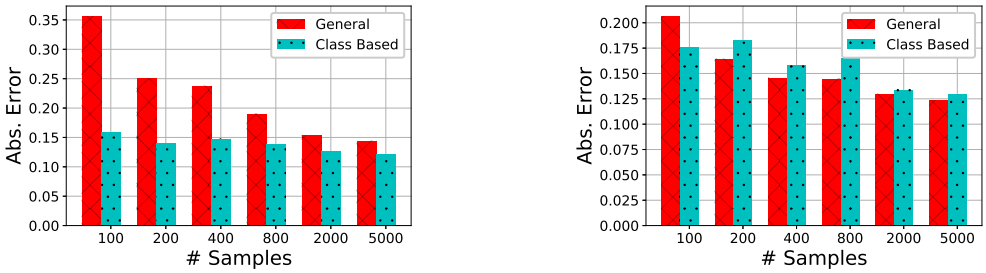
basically the non-distributed version of [74], where if the local result is not sufficiently reliable, further CNN layers in the edge or cloud are invoked.

- *Resource-Consumption Offloading* (RCO), where a task is offloaded when there is enough energy, without considering the expected classification improvement.
- *Online Code Offloading and Scheduling* (OCOS) [160], where the devices always try to exploit the cloudlet's classifier, and the cloudlet tries to schedule as many tasks as possible in each slot, given its available resources.

6.5.2 Initial Measurements

We used our testbed to verify these small resource-footprint devices require the assistance of a cloudlet. These findings are in line with previous studies, e.g., [74, 76]. The performance of a CNN model increases with the number of layers (as we will show next), but so does the model size, see Fig. 6.2c. We find that, even with 4 layers, a CNN trained for CIFAR has 1GB size and hence cannot be stored in the RPis (e.g., even more so in a smaller IoT node). Similar conclusions hold for the KNN classifier, the accuracy of which is directly linked to the number of labeled local data (KNN needs the training data available locally). Clearly, despite the efforts to reduce the size of ML models by using, for instance, compression [72] or residual learning [73], the increasing complexity of analytics and the small form-factor of devices will continue to raise the local versus cloudlet execution trade off.

Here we evaluate the different classifier and predictor designs towards building a more efficient system. In Fig. 6.3a we see that the accuracy (defined as the ratio of correct predictions over the sum of all predictions) of the KNN classifier improves with the size K_n of labeled data when



(a) Linear regressor (b) Random forest

Figure 6.4: Predictor assessment.

applied to MNIST. Figure 6.3b depicts the accuracy improvement for CNN as more hidden layers are added. The performance increase is higher for the digits that are more difficult to recognize (e.g., 4 and 5), up to about 20%. Notice, that the performance of the CNN classifier is superior to KNN, when we use fewer layers, or samples respectively. In addition, we present the CNN performance on CIFAR, for 1, 2 and 4 hidden layers in Fig. 6.3c. CIFAR is more complex than MNIST due to the properties of its objects (colored images, etc.), and this results in lower accuracy. Overall, we see that the classifier performance depends on the algorithm (KNN, CNN, etc.), the settings (datasets, layers, etc.), and differ also for each object class. *Hence, an algorithm is necessary that can adapt to all these parameters* (as OnAlgo does). Since we have verified the superiority of CNN classifiers, we continue our evaluation using only these, instead of KNN.

Finally, we studied the training dataset impact on the predictor's error, using both general and class-specific *(i)* linear regressors and *(ii)* random forests. In Fig. 6.4, we plot the prediction error of the accuracy improvement for both cases of general and class-specific predictors for CNN local device and cloudlet classifiers. We observe that the random forest is superior to the simpler linear regressor only when the number of samples is small. Moreover, random forests display an inconsistency when comparing general to class-based models as the number of training samples varies. The class specific linear regressor for 5K samples achieves the lowest average absolute error, thus it is used throughout the following experiments, while its error is rapidly decreasing from 35% for 100 points to 12.3% for 5K points on the CIFAR dataset.

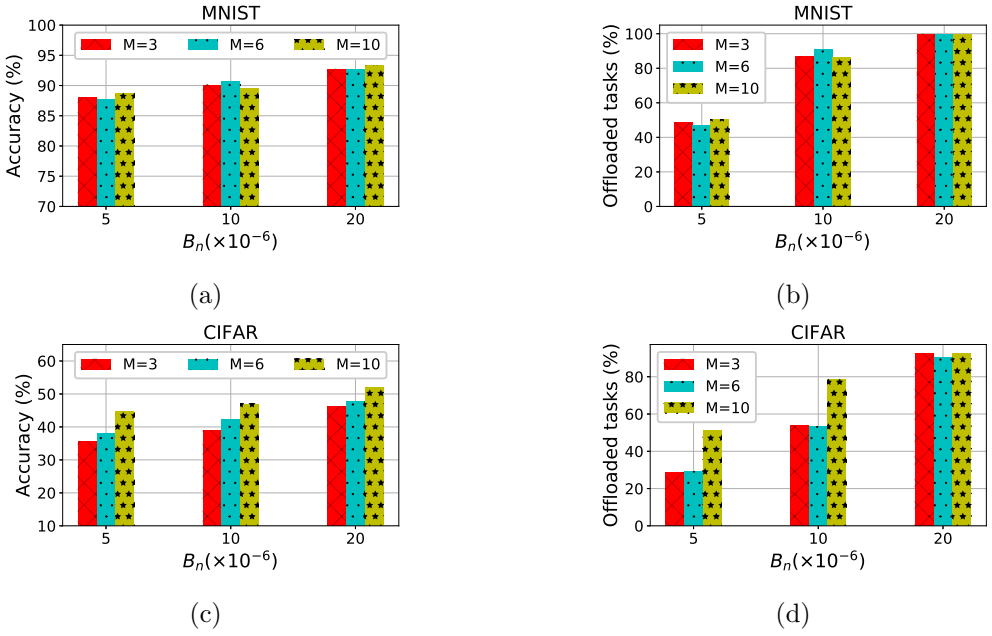
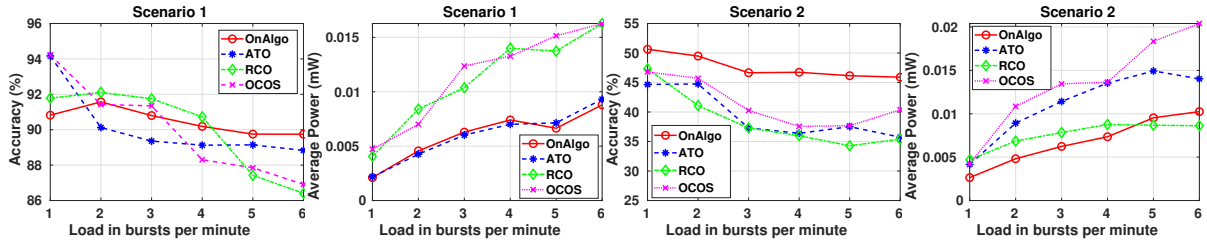


Figure 6.5: Accuracy and offloading percentage of OnAlgo for various resource constraints and values for M, on MNIST and CIFAR-10.

6.5.3 Performance Evaluation

Next, we evaluate the performance of OnAlgo in terms of achieved accuracy, offloading frequency and resource consumption. First, we evaluate OnAlgo for different values of the power consumption constraint B_n and quantization level M. Then, we use a variable non-i.i.d. traffic load to compare its performance against the competitors. The traffic load is an exponentially distributed sequence of task bursts, each of which has a uniform duration of $5 - 10$ seconds. This way we emulate the real-world scenario of sensor-activated cameras that generate images for short time periods.

Resource Availability and Quantization Impact. We evaluate the performance of OnAlgo, by using a 1-layer CNN for the RPis and a 4-layer CNN for the cloudlet. In Fig. 6.5 we show the average accuracy achieved by the four devices, as well as the fraction of requests offloaded to the cloudlet when we vary the devices' power budget B_n, for MNIST and CIFAR. Evidently, as B_n increases there are more opportunities for exploiting the cloudlet and obtaining a better result than the local classifier. Interestingly, the selection of M seems to positively affect the offloading frequency, resulting also in higher accuracy, in specific cases only. In detail, increasing M results in better performance just for CIFAR (the improvement gains are higher),



(a) (b) (c) (d)

Figure 6.6: Performance comparison of different offloading algorithms vs the generated task load.

and in the low resource availability regime. This practically means that a higher quantization level for the accuracy gain is mostly effective when the cloudlet classifier is substantially better than the local one, provided that the increased algorithm complexity from the increased M can be handled by the system. Furthermore, some interesting remarks can be made by comparing the two datasets. As shown in Fig. 6.3(b-c), MNIST is easier to classify and the gain of using a better classifier is not as important as on the CIFAR dataset. In particular, with MNIST the gains are about 6% in accuracy as the resources (and thus the offloaded tasks) increase, while M does not affect the performance. With CIFAR, on the other hand, the potential performance gain when using the cloudlet is larger; and as B_n and M increase, the accuracy gains are up to 25%. These two experiments demonstrate *the agility of our algorithm, which assesses the potential accuracy gains and shapes accordingly the offloading strategy, based on resource availability (cloudlet computing capacity), and the level of quantization.*

Comparison with Benchmarks. Next, we compare OnAlgo to ATO, RCO and OCOS for a varying non-i.i.d. traffic load in Fig. 6.6. To ensure a realistic comparison, we set the rule for all algorithms that the cloudlet will not serve any task if the computing capacity constraint is violated; while for RCO the availability of energy is determined by computing the running average consumption by each device during the experiment. We employ two testbed scenarios, and a simulation with larger number of devices.

Scenario 1: *Low accuracy improvement; high resources.* In this case, we set[9] $B_n = 0.02\ mW$ and $H = 2\ GHz$ allowing the devices to offload many tasks, the MNIST dataset (has small improvement between 1 layer and 4 layer CNNs), and $M = 6$. We depict the average accuracy achieved by the devices and the average power consumption versus the task load in bursts

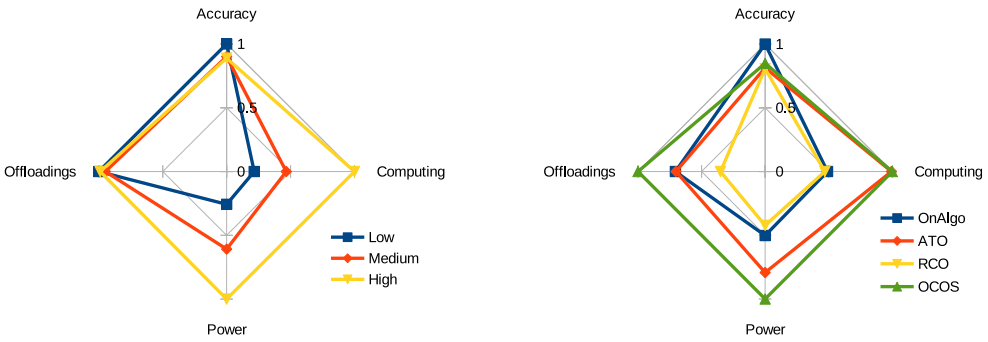
[9]We have explicitly set a small power budget so as to highlight the impact of power constraints on the system performance; higher power budgets will still be a bottleneck for higher task request rates or images of larger size.

per minute in Fig. 6.6a and 6.6b respectively. We observe that OnAlgo shows a smaller slope in the decrease of accuracy, as the load increases than all the competitors. The performance of ATO quickly drops because the cloudlet's resources are insufficient for high loads. RCO's performance is good for the most part (at the cost of increased power consumption), but it quickly deteriorates for high task loads as the devices many times refrain from offloading due to the power constraints. OCOS performs similarly to RCO since performance degradation is caused by cloudlet resource exhaustion. The problem with both algorithms is that they do not offload intelligently, based on both the improvement potential *and* the availability of resources, as opposed to OnAlgo.

Scenario 2: *High accuracy improvement; low resources.* The settings for this scenario are $M = 10$, $B_n = 0.01\ mW$, $H = 500\ MHz$ not allowing many offloadings and cloudlet classifications. We used the CIFAR dataset that demonstrates a substantial performance difference between local and cloudlet classifiers. We see from Fig. 6.6c that OnAlgo is up to 12% more accurate than ATO/RCO for high task load, and in any case significantly higher than in Scenario 1. OCOS performs slightly better than ATO/RCO, but at the cost of very high power consumption. Since the potential of improvement is higher in Scenario 2, ATO marginally outperforms RCO by spending up to 50% more power than RCO (see Fig. 6.6d). OnAlgo consumes about 50 % less power than OCOS since the latter always tries to offloads tasks but does not leverage the cloudlet efficiently due to the lack of computing capacity.

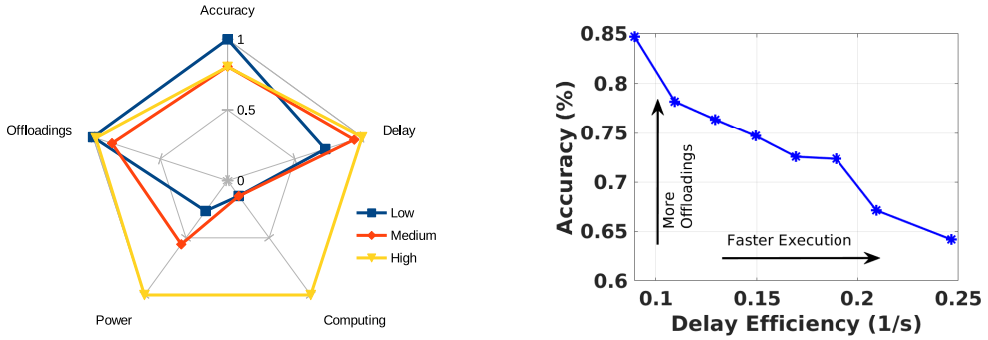
Summing up the 2 scenarios above, we see that OnAlgo achieves a smooth performance across varying traffic loads, while its competitors struggle, especially as the load increases. Moreover, it achieves reasonable power consumption regardless of the resource availability as opposed to RCO in Scenario 1, ATO in Scenario 2, and OCOS in both scenarios.

Trade-off Analysis. Next we demonstrate the trade-offs between number of offloadings, accuracy and resource consumption between OnAlgo and the competitor algorithms using net graphs. Fig. 6.7a displays the performance of OnAlgo for low medium and high task load. Observe that as the load increases, OnAlgo rapidly increases resource consumption to maintain high accuracy. For instance, comparing low to high load, we see that performance drops only by about 7% as the computing and power consumption is increased by 75%. In Fig. 6.7b we compare the same metrics for high traffic load, and the different competitors. Observe that OnAlgo achieves the highest accuracy, while being (closely) second best (behind RCO) in terms



(a) (b)

Figure 6.7: Comparison of different key metrics (normalized): (a) OnAlgo for low, medium and high traffic load. (b) Algorithm comparison for high load in scenario 2.



(a) (b)

Figure 6.8: (a) OnAlgo performance for low, medium and high traffic load for problem (P3). (b) Pareto front between accuracy and delay efficiency.

of computing resource and power consumption. Moreover it achieves high accuracy despite offloading less frequently than OCOS, due to the intelligent way it makes the offloading decisions. In summary, OnAlgo achieves the *highest* accuracy between the competitors, and at the same time has a *moderate* resource consumption.

Next, in Fig. 6.8 we explore the accuracy-resource consumption-delay trade-off when problem (P3) is considered, i.e. total delay is jointly optimized with accuracy. Notice in Fig. 6.8a, that the increasing traffic load will not only result in lower accuracy (about 20%) and higher resource consumption, but also in significantly higher delay (up to 25%). Hence, despite consuming extra resources in high load cases, OnAlgo still maintains high accuracy standards. Finally, Fig. 6.8b displays the Pareto front between accuracy and delay[10]. This shows the effect of parameter ζ

[10]In fact delay is inversed (1/s) so that increasing the value towards either the x-axis or the y-axis yields better performance with respect to the relevant metric.

(ranging from 0.1 to 0.3) on the resulted offloading policy and consequently on the performance of accuracy and delay. For instance, in order to double the delay efficiency (from 0.1 to 0.2), we would have to sacrifice roughly 10% accuracy, by offloading less frequently.

6.6 Conclusions

In this chapter we propose an algorithm for online data analytic task outsourcing from small end devices to a cloudlet capable of improving classification results. This is possible since more complex and resource heavy ML models can be deployed there, compared to the end devices. Moreover, the tasks are typically executed faster at the cloudlet, leveraging its high end hardware setup. Hence, in the following chapter, we build an MEC enabled object recognition system where mobile devices offload their object recognition tasks directly to a GPU-equipped edge server, and study accuracy/latency trade-offs that arise from tuning application specific system parameters.

Chapter 7

Trade-off Analysis of a MEC Object Recognition System

Edge-assistance will most likely be a key component of future latency-critical and computationally-demanding mobile applications such as video analytics and Tactile Internet services [161, 162]. Augmented Reality [80] and real time object recognition [79] are examples of such services that can benefit from the computational power of a nearby edge server, since mobile devices are too slow to timely perform the required computations. Nevertheless, the practical performance benefits of such edge architectures remain unexplored. On the one hand, data transmissions are added to the service delay. On the other hand, the quality and execution delay of analytics is affected by the volume of the transmitted data, as well as the complexity of the algorithm running on the edge server.

In this chapter, we investigate this issue experimentally by building the edge computing system illustrated in Fig. 7.1. We develop a real-time object recognition system, as a representative of the plethora of emerging visual-aided services, e.g. video stream analytics, mobile augmented reality, etc. A mobile handset (client) captures camera images and transmits them to an edge server for processing; the server uses a deep neural network (NN) to detect and classify objects in the images; and sends the output to the handset which overlays this information on the screen. We built the above system using an Android application and a state-of-the-art deep learning network running on GPU hardware for the server. We use a high performance 802.11ac wireless link for communication between the handset and the server, which features technology likely to

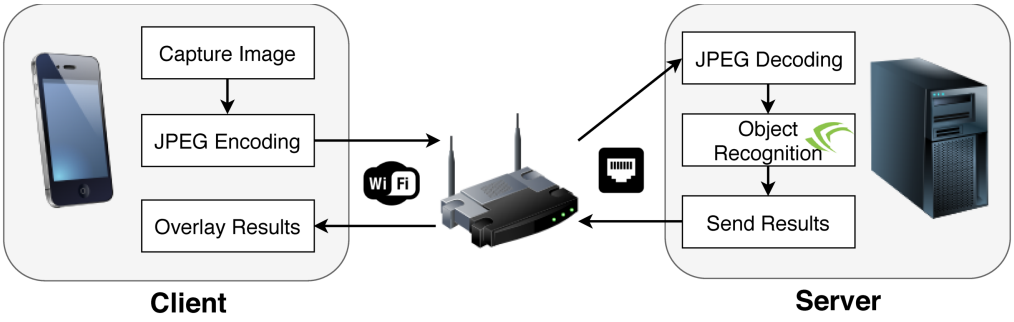


Figure 7.1: Schematic of edge-assisted object recognition system.

persist in future small cells[1], hence making our results relevant to a range of systems.

7.1 Motivation

Our goal is to understand the system-level trade-offs between end-to-end (E2E) latency and object recognition accuracy, and propose specific solutions that can improve the performance of the system. We firstly show that the degree of image compression and deep learning NN input size are key parameters affecting both performance metrics. In particular, the use of more aggressive image compression saves on communication latency between client and server (since the transmitted image file is smaller), but at the cost of reduced object recognition accuracy. While the impact of image degradation due to noise or blur on recognition accuracy has started to receive attention in the deep learning literature [163], the impact of compression on accuracy remains relatively poorly understood. Furthermore, a large NN size will improve recognition performance at the cost of higher execution delay at the server, hence increasing E2E latency. To the best of our knowledge, the trade-off between E2E latency and recognition accuracy for the above parameters, has not previously been explored.

We focus our effort in designing wireless transmission interventions that further improve the communication delay of the system. Such interventions have not yet received significant attention by the edge computing literature, as most efforts have been devoted to minimizing computation delays [18,77,91]. This delay source however, is of critical importance to low latency services, and hinders their ability to achieve real time performance, e.g. [79, 81]. We show that *transmit time can be reduced by up to 65%* by sending the images as short back-to-back bursts

[1]We use MU-MIMO/OFDM and channel aggregation at the PHY layer, and employ packet aggregation at the MAC layer to reduce framing/signaling overheads.

of UDP packets. We also find that the client Network Interface Controller (NIC) powersave can incur substantial transmit latency and, hence, *smarter sleep mode adaptation can further decrease latency by up to 60%.*

Finally, we model the different sources of delay in our system, and the obtained accuracy, as functions of the NN size and encoding rate using our measurements. We illustrate the use of the developed model to highlight optimal trade-offs between E2E latency and system object detection accuracy. Moreover, we show that the use of smart wireless transmission techniques employed, can nearly double the system performance along the Pareto-optimal curve of accuracy vs frame rate.

7.2 Preliminaries

7.2.1 Hardware & Software Setup

We developed an Android application that captures images through the handset's camera, carries out JPEG encoding and then transmits the compressed images to an edge server for processing. The server software (written in C/C++) decompresses and pre-processes the images, and submits them to the deep learning neural network (NN) which is implemented using a GPU-optimized framework. The results, i.e., the bounding boxes and labels, are then sent back to the client handset and overlaid on the displayed image.

Object recognition is performed by YOLO [164], a state-of-the-art deep learning detector implemented on darknet, an open source framework that supports GPU computations via cuda. It takes an $n \times n$ array of image pixels as input, with each pixel being a float value, and down-samples by 32 to give an $n/32$ grid. Then, each grid cell proposes bounding boxes and labels for any contained objects. These results are filtered to generate the output consisting of a set of bounding boxes of recognized objects with their labels and respective confidence. Since the NN only uses convolutional and pooling layers, the input size n can be rescaled without requiring NN retraining as long as N is a multiple of 32.

We use different mobile devices to measure the effect of the end user's hardware on the system's performance: *(i)* a Google Pixel 2 (default device), *(ii)* a Samsung Galaxy S8, and *(iii)* a Huawei P10 Lite. All devices are equipped with 802.11ac chipsets, and we will be using the Google phone unless stated otherwise. The edge server is connected via Ethernet to a WiFi

router that serves as an access point (802.11ac, 5GHz) for the handsets[2], see Fig. 7.1.

7.2.2 The Need for Edge Server Offload

We investigated first the viability of running YOLO on the handset by cross-compiling darknet, but found that the running times were excessive (on the order of minutes). Use of a cut-down version of YOLO, referred to as TinyYOLO [164], was also investigated. The running time was around 1s per image, substantially faster than with the full YOLO network but still very slow compared to the server. Note also that the speedup of TinyYOLO is obtained at the cost of significantly reduced object recognition accuracy, and supports only a small subset of object types. Our tests convey the same message as previous studies [61, 70], namely confirm the *necessity for offloading the object recognition task to a powerful server, if low latency operation is to be obtained.*

7.2.3 Evaluation Scenario

To evaluate the system performance we used the extensive COCO dataset [165] which covers a wide range of images and objects, and includes ground truth for each image (object locations and labels within each image). For quantifying performance, we used the Average Precision (AP) and Average Recall (AR) metrics[3] for a range of Intersection-over-Union (IoU) values. Detection is considered successful when the ratio of the overlapping area between the detected object and the ground truth, over their respective union area, is higher than an IoU value of 0.5. COCO further breaks precision and recall metrics down by whether objects are large, medium or small. YOLO is known to perform poorly on small objects and so we focus on large and medium objects.

To use the COCO images we connected the phone to a server via a USB cable and a Python script on the server sends commands to the phone using the Android Debug Bridge (adb). The server initiates the client application through adb and configures the system parameters for the experiment (e.g., the JPEG compression level). Then it iterates over 5000 images from the COCO validation set, sending them one-by-one to the phone through cable. The phone transmits each image to the server through the wireless interface, as if they were images captured

[2]The edge server is a 3.7 GHz Core i7 PC equipped with 32GB of RAM and a GeForce RTX 2080Ti GPU; and the router is the ASUS RT-AC86U.

[3]AP is the ratio $T_p/(T_p + F_p)$ while AR is the ratio $T_p/(T_p + F_n)$, with T_p being the true positive detections, F_p the false positive and F_n the false negative detections. The results are averaged over all objects classes.

by its camera, receives the server response over WiFi and passes this back over the USB cable for logging.

7.3 System End-to-End Latency

Our first goal is to measure each of the different delay components involved in the procedure, and investigate how they are affected by the encoding rate q and NN size n, but also by the network set up (from the transport, to data link and physical layer). Based on our findings we propose and evaluate network design choices that speedup the task completion.

7.3.1 Encoding Delay (T_{enc})

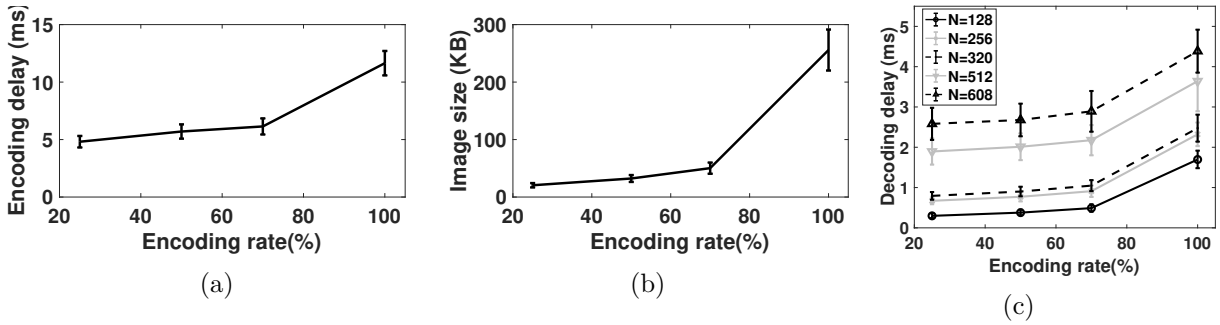
The handset application converts its camera images to JPEG format before transmission to the server. We use JPEG as it is widely adopted and supported by the Android API. While image encoding is a typical step in such systems, its impact on the performance of edge-assisted object recognition has not received attention, with only few exceptions [162]. JPEG is a lossy format and its compression is decided by the *encoding rate* q. Note that we rely on the terminology of the compression library we employed in our system[4] and define $q \in [10, 100]$ as the percentage ratio of compressed image size over its actual size, where $q = 100$ for an uncompressed image.

At higher encoding rates, the number of discrete cosine transform coefficients that represent the JPEG image is larger, leading to an expected increase in the encoding delay. Indeed, Fig. 7.2a (upper plot) shows the encoding delay T_{enc} vs. the encoding rate q. It can be seen that T_{enc} grows from 5ms to 11ms as q increases from 25% to 100%. This has also impact on the size of the compressed image, see Fig. 7.2a (lower plot).

7.3.2 Decoding and Pre-processing Delay (T_{dec})

Upon receiving an image, the server *(i)* decompresses it to obtain an RGB image; *(ii)* re-samples/pads the image to match the input size n of the deep learning network; *(iii)* rotates the image to compensate for the handset camera orientation; and *(iv)* converts the pixel values from 0-255 integers to 0-1.0 floats. Our profiling indicates that *most of this processing is limited by memory resources rather than CPU*. Hence, in our implementation we execute steps *(i)* and

[4]For jpeg compression (through quantization) we used the Android library: `https://developer.android.com/reference/android/graphics/YuvImage`.



(a) (b) (c)

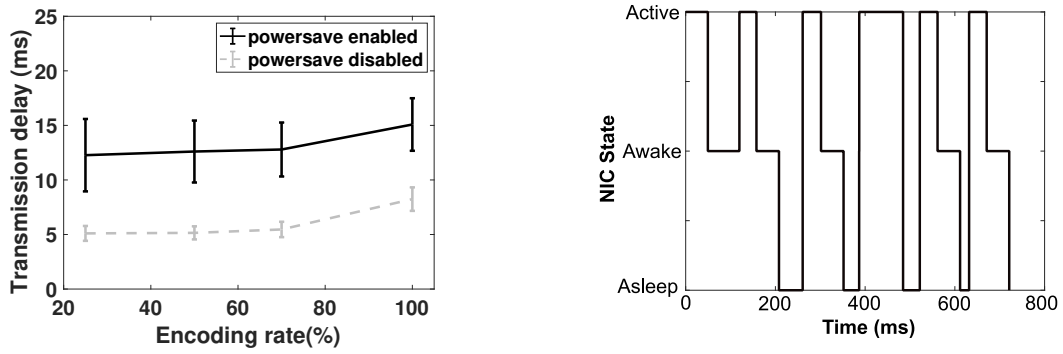
Figure 7.2: (a) Time used for JPEG encoding and (b) resulting image size. (c) Decoding and preprocessing delay, vs encoding rate q. Results are averaged for the entire COCO library (5000 images).

(ii) jointly so as to minimize memory movements and maximize scope for in-processor caching. And similarly we designed our implementation to execute simultaneously steps *(iii)* and *(iv)*. Contrary to encoding delay, this part of the processing depends both on the encoding rate and the NN size. Fig. 7.2c plots measurements of the processing time vs. q and n. Observe that when $q \leq 75$ the latency is largely insensitive to q, i.e., it is dominated by the preprocessing steps other than image decompression. Similarly, the NN size n affects significantly T_{dec} only when it is very large (notice the sudden increase when $n \geq 512$). As we will see later, these findings create opportunities for optimizing the overall system operation.

7.3.3 Transmission Delay (T_{tx})

Next, we investigate the network impact on the task delay, and propose specific solutions that can effectively halve this time. First, note that the size of the transmitted images vary between 20–250KB, corresponding to roughly 13–166 packets (each 1500B long). In contrast, the server response contains object bounding boxes and typically fits into a single packet. Hence, the network transmission delay is dominated by the time taken to transmit the image and we expect that this will increase with the encoding rate q.

The solid line in Fig. 7.3a plots the transmission delay vs. q. This delay includes the time needed to send the image to the server and the time for transmitting back the response. The measurements are when TCP is used with default Android and Linux settings, i.e., Cubic congestion control and dynamic socket buffer sizing. As expected, the delay tends to increase with the JPEG quality (for larger q). However, when $q < 80$ the delay is relatively insensitive to the encoding rate. *Further investigation reveals that this insensitivity is mainly caused by two*



(a) (b)

Figure 7.3: (a) Wireless transmission delay using TCP vs JPEG encoding rate, (b) example time history of the NIC state on the mobile handset when power saving is enabled.

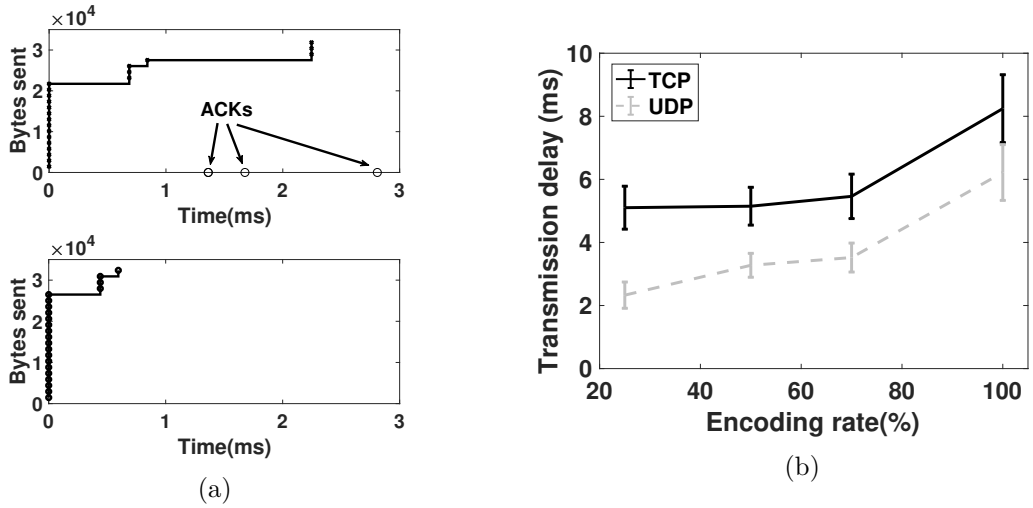
factors. Firstly, the handset's power management aggressively puts the NIC into sleep mode, and this induces a delay to wake the NIC when transmission or reception restarts. Secondly, the dynamics of TCP congestion control mean that it takes multiple round-trip times to transmit all image packets. Next, we propose solutions for these two issues.

7.3.3.1 Handset NIC Wake-from-Sleep Latency

When entering sleep mode, the handset's 802.11 NIC sends a special flagging frame to the AP which buffers any packets awaiting transmission until the handset signals it has woken up. Fig. 7.3b plots an example time history of the handset's NIC state derived by extracting these state transitions from tcpdump data[5]. Also indicated on Fig. 7.3b are "active" periods where the NIC is awake and exchanges data with the server. Note that the NIC regularly enters a sleep state, waking up when the handset starts to send an image. As indicated by our measurements above, the handset can roughly predict when the next image transmission will occur. Namely, a new captured image is transmitted approximately after 5-10ms (time for its encoding), and this could be used to preemptively wake up the NIC.

Solution: In order to investigate the potential latency gains of smart wake-up strategies, we adopted the cruder approach of using iperf to generate 1Mb/s of background UDP traffic from the server to the client, to keep the handset's wireless interface awake. The dashed line in Fig. 7.3a shows that the overall transmit delay is now decreased for all values of q, consistent

[5]In our experiment a delay is inserted between input of each image to the android app to make the power-save behavior easier to see.



(a) (b)

Figure 7.4: (a) Time histories showing transfer of a compressed image from client to server using TCP (upper plot) and UDP (lower plot). (b) Wireless transmission delay for TCP and UDP vs JPEG encoding rate q with mobile NIC power-save disabled.

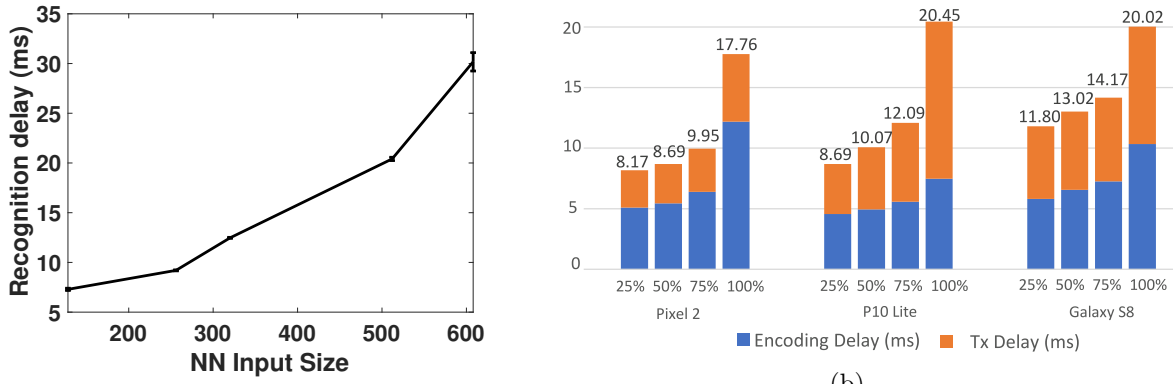
with the handset NIC no longer having to be woken up for transmitting the image. *The delay reduction is approximately 5ms for all encoding rates which corresponds to a reduction of 50% in the wireless transmission delay.*

7.3.3.2 Latency Caused By TCP Dynamics

The upper plot in Fig. 7.4a shows the time history when transferring an image using TCP. The connection is kept open and used for sending multiple images so that the overhead of the TCP handshake (SYN-SYNACK-ACK) is only incurred once (takes 4ms; not shown). The compressed image in this example is 31335B in size, and when the HTTP request header is added, it occupies 22 TCP packets[6]. Its transmission lasts 2.5ms and uses 4 MAC frames for data and 3 for TCP ACKs. On average, 5.5 TCP data packets are therefore sent in each MAC frame. Observe that the client needs to receive TCP ACKs before it can send the full image since the TCP congestion window (cwnd) limits the packets in flight to around 10 when starting a new transfer. Also, observe that there is contention between uplink and downlink due to the ACKs transmitted by the server.

Solution: We explore the gains from removing uplink/downlink contention and the impact of TCP cwnd, by modifying the Android client and server to use UDP. At the client side, an image is segmented and placed into a sequence of UDP packets which are then sent to the

[6]The payload of a 1500B TCP packet is 1448B including header overheads.



(a) (b)

Figure 7.5: (a) Server recognition delay vs NN size. (b) Edge device delay comparison.

socket back-to-back to facilitate aggregation by the NIC. The lower plot in Fig. 7.4a shows UDP measurements[7] for transmission of the same image. Despite that UDP packets are fit within a single MAC frame (our system can aggregate up to 128 packets in 1 frame), we see that the transfer used actually 3 frames. Presumably this is due to the scheduling delays between the kernel and NIC, and the relative timing of channel access opportunities and packet arrivals. Nevertheless, we find that the data transfer time is now 0.8ms, i.e., 3 times faster than with TCP. Finally, Fig. 7.4b plots measurements of the overall wireless transmission time (sending the image and receiving its response) for the full COCO data set when using TCP and UDP; and with mobile NIC power-save disabled. We find that using UDP packet bursting roughly halves the transmit time for all JPEG encoding rates.

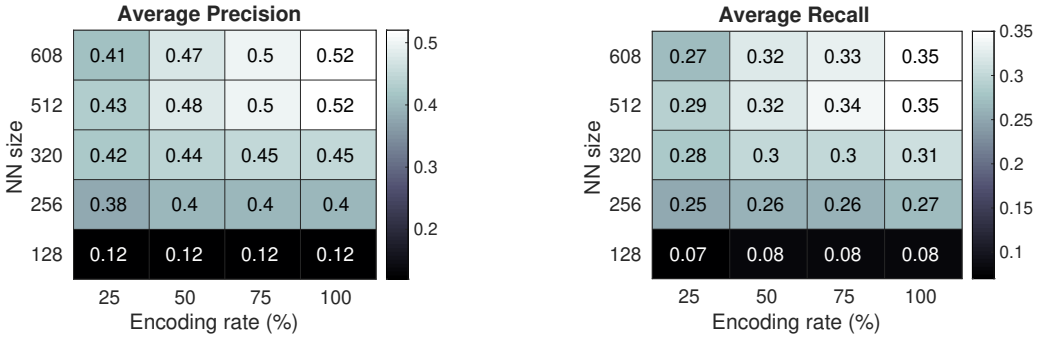
Concluding, in this subsection we showed that *tailored transmission strategies, such as smart NIC power-saving and using UDP with packet bursting, reduce the transmit time to around 5ms.* This improvement is hugely important given the targeted E2E latency budgets.[8]

7.3.4 Recognition Delay (T_{dl}) and Impact of Handheld

YOLO outputs the coordinates of the image's detected objects along with their labels. The recognition delay T_{dl} depends on the NN size, and our measurements in Fig. 7.5a show that it increases, roughly, quadratically with n. Other works have reported similar findings, e.g., see [18, 61], but the delays are quite higher than our results, presumably due to the usage of

[7]Including the time needed to segment the image into UDP packets, so the values are comparable with the TCP data.

[8]To achieve real time frame update rates, such as 30fps, the available total latency budget is only 33ms.



(a) (b)

Figure 7.6: Precision and recall (IoU=0.5) vs NN size and encoding rate.

older GPU hardware. Furthermore, DeepMon [91] proposes NN optimizations on the mobile devices that reduce the delay at about 1sec for YOLO, but it is still worse than our system's performance. These values may vary from system to system, but we expect qualitatively the trend to persist.

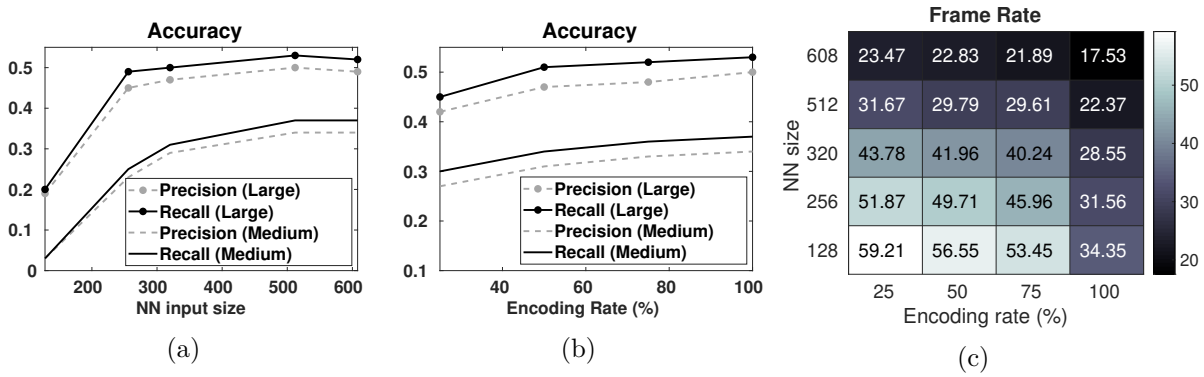
Similarly, we suspect that the handset hardware affects only slightly (i.e., quantitatively) the results. To verify this, we repeat our experiments with 2 additional mobile devices. The delays that are directly related to the handset device, and may vary due to the different hardware specifications, are the encoding and transmission delay. Fig. 7.5b plots the total encoding and transmission delay measured for the 3 devices (Pixel 2, P10 Lite, Galaxy S8) for each encoding rate q (averaging all dataset images). We find that compared to the Pixel 2, the other 2 devices are slightly faster in image encoding, but also slower in transmitting. Such differences might likely arise due to the different chipsets/firmware implementations. Observe however, that the roughly quadratic increase of both delay components persists across all devices as q increases. Hence, qualitatively the results hold for different hardware.

7.4 Performance Trade-offs

Using our measurements above we discuss here the interaction and trade-offs between the two performance metrics, i.e., the accuracy and E2E delay, under a range of different scenarios. We discover that *in several cases there are sharp trade-off curves which create opportunities for improving the system operation, by carefully tuning parameters q and n.*

Figures 7.6a-7.6b plot the object recognition[9] average precision and recall vs the encoding

[9]We have used the Python library CoCoApi for calculating these metrics, `https://github.com/cocodataset/`



(a) (b) (c)

Figure 7.7: (a-b) Precision and recall for medium and large objects for uncompressed images. (c) Frame rate vs NN size and encoding rate.

rate q and the NN size n. We see that both metrics generally increase with q and n, although there is a sharp improvement going from $n=128$ to $n=256$. Moreover, as n drops the precision and recall performance deteriorate and cannot be improved even if we use high q (e.g., see last row in each matrix). This finding differs from previous studies, e.g., [163], perhaps due to the COCO dataset which contains images with a large range of object sizes.

We further study the impact of the object sizes on performance, while we consider different detection thresholds (IoU values) [165]. In Fig. 7.7a we plot the precision and recall vs n and q for large and medium objects, averaged for a range of IoU values. We see that for large objects the accuracy increases rapidly with n but plateaus when $n>300$. For medium objects on the other hand, the benefits of larger input size (and so higher image resolution) are greater and accuracy only plateaus when $n>500$. Fig. 7.7b shows that the dependence on q, albeit not that strong, follows indeed a continuous increase. We note that the precision and recall values in these plots are relatively low because we use very high IoU thresholds (up to 0.95). Also, we do not consider larger NNs since for $n=608$ we already have satisfactory precision but also large delays.

Finally, we study the frame rate, i.e., the reciprocal of E2E latency, for different NN sizes and image encoding rates. Fig. 7.7c presents the average frame rate for each scenario. Notice that for small NNs ($n<320$) the encoding affects significantly the frame rate, but this effect is weaker for $n>320$. For example, when $n=608$ the rate falls below 30fps even for very small values of q. In other words, we find that in the low NN size regime, the accuracy gains from choosing a high encoding rate are not significant, while the frame rate gains of a low encoding rate are substantial.

`cocoapi/tree/master/PythonAPI/pycocotools`.

Hence, a low encoding rate is probably more suitable for a small NN. The opposite is true in the high NN size regime, where we can achieve substantial accuracy gains without compromising significantly the frame rate. *These findings underline the importance of selecting jointly the values of parameters n and q.* Next section provides a systematic methodology towards that end.

7.5 Data Models and Pareto Analysis

Using our observations in the previous section, we discuss here the trade-offs between accuracy and delay, fit our measurements to create statistical models for these performance metrics, and characterize the pertinent Pareto fronts based on two representative optimization problems.

7.5.1 Fitting the Measurements

Our measurements indicate that the latency components and accuracy can be approximated using quadratic functions of the decision variables n and q. Note that only the decoding delay T_{dec} and precision f (we omit recall for brevity) depend on both n and q. On the other hand, the encoding and transmission delays, T_{enc} and T_{tx}, depend only on q, and the deep learning delay T_{dl} on n. We therefore define:

$$T_{enc}(q) = \alpha_0 + \alpha_1 q + \alpha_2 q^2, \tag{7.1}$$

$$T_{dec}(n,q) = \beta_0 + \beta_1 n + \beta_2 q + \beta_3 nq + \beta_4 n^2 + \beta_5 q^2, \tag{7.2}$$

$$T_{tx}(q) = \gamma_0 + \gamma_1 q + \gamma_2 q^2, \tag{7.3}$$

$$T_{dl}(n) = \delta_0 + \delta_1 n + \delta_2 n^2, \tag{7.4}$$

$$f(n,q) = \epsilon_0 + \epsilon_1 n + \epsilon_2 q + \epsilon_3 nq + \epsilon_4 n^2 + \epsilon_5 q^2. \tag{7.5}$$

The model parameters are obtained by fitting our measurements to (7.1)-(7.5). Clearly, the exact values of these parameters can change if, for instance, we use a different access point or server. However, as our tests with the different handset devices have revealed, the changes are minimal and only quantitative.[10]

[10]The handsets affect only the values of parameters $\{\alpha_i\}_i$ and $\{\gamma_i\}_i$.

7.5.2 Pareto Analysis

We leverage the above models to explore the interaction of the decision variables:

$$n \in \mathcal{N} \triangleq \{[128, 608] \mid \text{mod}(n, 32) = 0\}, \quad q \in \mathcal{Q} \triangleq [10, 100],$$

i.e., study how they jointly affect the precision and the frame rate (E2E latency), while we also devise the Pareto fronts for these two performance criteria by following a detailed parameter-sensitivity analysis. We formulate two optimization problems; $\mathbb{P}_1$, where we maximize the precision subject to achieving a minimum frame rate; and $\mathbb{P}_2$ where we maximize the frame rate while not dropping the precision below a threshold value. Formally the 2 problems can be written:

$$\mathbb{P}_1: \quad \underset{n \in \mathcal{N}, q \in \mathcal{Q}}{\text{maximize}} \quad f(n, q) \tag{7.6}$$

$$\text{subject to:} \quad T_{total}(n, q) \leq T_{max} \tag{7.7}$$

$$\mathbb{P}_2: \quad \underset{n \in \mathcal{N}, q \in \mathcal{Q}}{\text{minimize}} \quad T_{total}(n, q) \tag{7.8}$$

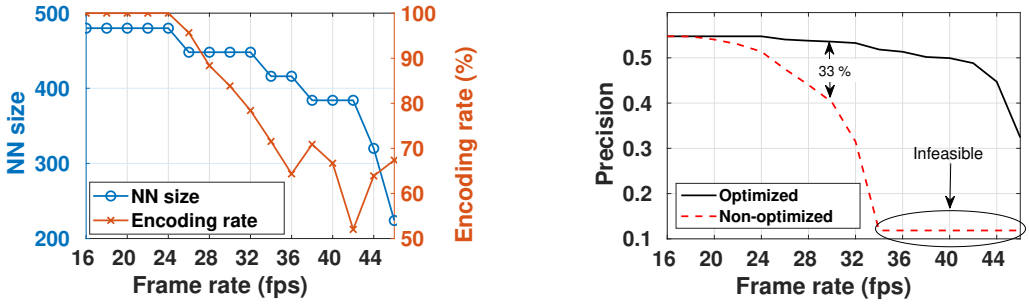
$$\text{subject to:} \quad f(n, q) \geq f_{min}. \tag{7.9}$$

where we have defined:

$$T_{total}(n, q) = T_{enc}(q) + T_{dec}(n, q) + T_{tx}(q) + T_{dl}(n),$$

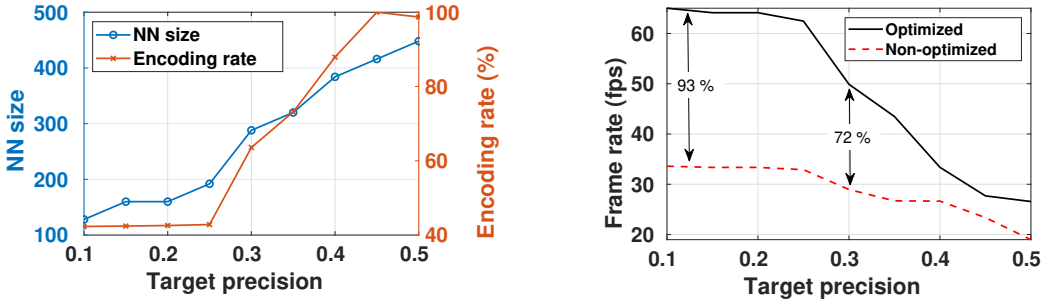
and T_{max} is the highest tolerable delay in order to achieve a frame rate of $1/T_{max}$ fps. Respectively, f_{min} is the target precision requested by the user. In essence, constraint (7.7) ensures that the total delay does not exceed T_{max}, and hence the frame rate $1/T_{total}$ will be greater or equal to the threshold $1/T_{max}$. Similarly in $\mathbb{P}_2$ we maximize the frame rate by minimizing T_{total}. Using both problem formulations we will be able to highlight the trade-offs between delay and precision.

Fig. 7.8a plots the values of n and q that maximize the precision while keeping the frame rate at or above the value indicated on the x-axis (recall that n is a multiple of 32). The achieved precision for each frame rate is displayed with a solid line in Fig. 7.8b. Observe how the increasing frame rate dictates the drop of NN size and encoding rate, which in turn result in decreasing precision performance. Moreover, we observe that the NN size continuously drops or



(a) (b)

Figure 7.8: (a) Optimal NN size and encoding rate for the desired frame rate. (b) Corresponding maximal precision values.



(a) (b)

Figure 7.9: (a) Optimal NN size and encoding rate for the target accuracy. (b) Corresponding maximal frame rate values.

stays level with the frame rate, while the encoding rate can increase in some cases. That occurs when the NN size has been reduced and hence the increase of the encoding rate can sustain a higher precision. Notice that for the largest range of frame rates, the NN size can be kept quite high (around and above 400), even when exceeding 30 fps. This yields a satisfactory precision of 0.5 at 40 fps[11]. However, after the 40 fps threshold, the NN size has to be very small to facilitate fast object recognition and the precision performance drops dramatically.

To highlight the impact of our optimized networking configuration, we compare the performance with the respective results of a non-optimized (*vanilla*) system, dashed line in Fig. 7.8b. Namely, these results were obtained by fitting the non-optimized (TCP, and enabled powersave) wireless transmission delay measurements to (7.3) and solving $\mathbb{P}_1$. Clearly, the increased transmission delays hamper the ability of the system to achieve high precision for acceptable frame rates (precision drops by 33% at 30 fps). Moreover, $\mathbb{P}_1$ becomes infeasible for a target frame rate

[11]Recall that we obtain low precision values because on purpose we used very high IoU values; for more typical thresholds the precision is much higher.

above 34 fps, indicating the greater range in which the system can operate after configuring the network. The respective results for $\mathbb{P}_2$ are displayed in Fig. 7.9a, 7.9b. The optimal frame rate can be kept very close to 30 fps, even for very high target precision. Also, we observe a huge gap between the optimized and non-optimized solution in this case, with the former achieving up to 93% higher frame rate than the latter when target precision is very low.

7.6 Conclusions

This chapter takes a measurement-driven approach to analyzing the trade-offs of an edge assisted object recognition application. We use our testbed to make an accuracy and latency evaluation of the possible service configurations, i.e. image encoding rate and NN input layer size. We further propose networking tweaks that greatly reduce the communication delay of the system, rendering on-device object recognition much less efficient than the edge computing alternative. Still though, there are other parameters to consider. The achieved latency and accuracy of such systems can vary with time and depend on the system software and hardware setup. In the following chapter, we utilize our testbed and extend our measurements to propose a solution for the online configuration of the system.

Chapter 8

Online Configuration of MEC Video Analytics

Existing MEC solutions manage computing or network resources to offload various tasks from user devices [86–88]. However, video analytics are heavily affected by several *new* parameters both at the user side, e.g., image resolution and encoding rate, and at servers, e.g., NN architecture. In particular, the key criteria of accuracy and latency are intertwined and shaped by the configuration parameters of the processing pipeline[1] and the wireless network that connects devices and servers. For instance, sending low-resolution or compressed frames reduces the transmission latency but also the object recognition accuracy; and increasing the frame rate improves a user's experience but strains the network and exacerbates the frame rate of others. Clearly, deciding jointly the resource allocation and pipeline configuration for multiple users is of utmost importance.

Pertinent studies focus on reducing the resource costs of video analytics [19, 96], and on maximizing their performance [18,98]. However, the dependence of performance metrics (accuracy, latency, and others) on the pipeline configuration that is distributed among several devices, and on the allocated resources, is unknown in practice, and might as well vary with time (e.g., due to wireless conditions). Importantly, as our experiments reveal, the performance depends also on the *platform*, i.e., the devices' and servers' hardware and software; and on the actual video data. Hence prior approaches that rely on statistical models, offline datasets or pre-training, are limited

[1]This term refers to the video processing stages, e.g., decoder, frame sampler and inference module (as, e.g., Yolo [164]), see [19] for details.

to specific systems and scenarios, and are ill suited for the heterogeneous edge environment. Here we take a fundamentally different approach and develop a *Bayesian learning framework towards automating the configuration of multi-user video edge analytic services*. This way we can tailor the configuration of each edge system to adapt to its users' requirements and resource availability.

8.1 Methodology and Contributions

We start with an experimental analysis using our prototype system presented in the previous chapter. We aim to maximize the recognition accuracy while satisfying user-defined minimum frame rate constraints; by deciding the image encoding rate, service time allocation and NN input layer size. We find that the system has stochastic accuracy and latency response (which shapes the achieved frame rate) even for fixed configurations. The former is due to differences in the images' objects, and the latter due to wireless channels and processing delay variations. We also find that similar configurations induce similar performance, which depends on the DNNs and devices, and even exhibits non-monotonic behavior. Our measurements extend prior works [18, 83, 166], and highlight the platform *and* data-dependent performance of these systems.

Motivated by these findings, we propose an optimization framework consisting of two components: a surrogate model builder for the unknown objective and constraint functions, and an acquisition rule that explores iteratively the system configurations. The former employs Gaussian Processes (GPs) and Bayesian updates [167] to construct the required models in real time using the collected data. The second component quantifies each configuration's performance and uncertainty regarding the existence of better configurations [168]. The result is a data-driven, platform-oblivious algorithm that is executed at system runtime. Its advantage is that, unlike other collaborative learning techniques, e.g. transfer learning, it is able to adapt to the inherent heterogeneity of the edge systems and learn the optimal configuration of each one. We prove that the algorithm finds a near-optimal solution and achieves average *sublinear pseudo regret* of $O(\sqrt{T}\gamma_T)$ where γ_T is a system-related parameter. Moreover, the algorithm performs *safe exploration* in the sense that it satisfies the users' minimum frame rate constraints while exploring the configuration space.

Our approach builds on the theory of Bayesian non-parametric learning, and falls into the

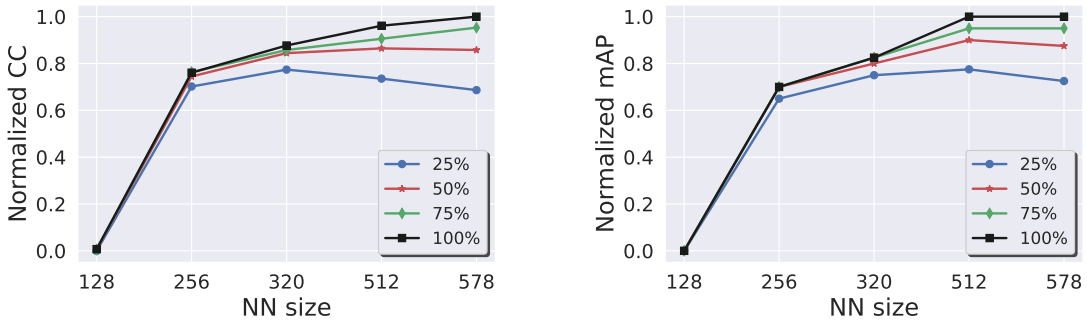
realm of Automated Machine Learning. *AutoML*, as it is known, has been used to automate the configuration of software packages[2], or the selection of various ML hyper-parameters [169, 170] which otherwise are set using heuristics [171]. We extend these ideas to automate the video pipeline and network configuration, while catering for frame rate constraints. This way, we tackle the main challenge of the service's dependency on system hardware and video data, and enable the users' devices to collectively configure the system in a way that satisfies their requirements. Being a powerful framework, it can be used to also allocate computing resources, select different networks, and so on (see details in Sec. 8.5).

Finally, we evaluate the system performance and find that our algorithm can get to within 5% from the optimal point in no more than 200 iterations. We also propose a set of practical steps to improve its performance, based on our experimental observations, e.g., the usage of stopping criteria for the different stages of the algorithm. Our technical contributions can be thus summarized as follows:

- Experimentally-motivated problem. We perform extensive experiments using different system equipment and datasets which reveal the volatile performance of video analytics and their dependency on said system and data. All our measurements are made available in an online fully-documented dataset [172].
- AutoML Framework. We propose a technique that finds a near-optimal configuration without violating the users' frame rate thresholds. This is achieved by combining a Bayesian GP technique with bandit learning and safe constraint exploration. To the best of our knowledge, this is the first time an AutoML framework is used to configure a video edge analytics service.
- Model Extensions. We extend our analysis to problems where additional video-related (e.g., frame resolution) or network parameters (e.g., user association to networks/servers) are decided. This manifests the framework's potential.
- Prototypes and Experiments. We evaluate the framework based on real data in our bespoke prototype, where we perform a thorough parameter sensitivity analysis, quantify its overheads, and verify its generality using a wealth of scenarios and system setups.

Our approach is different from related works (see Ch.2) since: *(i)* it uses Gaussian Processes

[2]E.g., the many parameters of mathematical solvers such as IBM CPLEX.



(a) Cumulative Confidence

(b) Mean Average Precision

Figure 8.1: Comparison between online (CC) and offline (mAP) accuracy metrics.

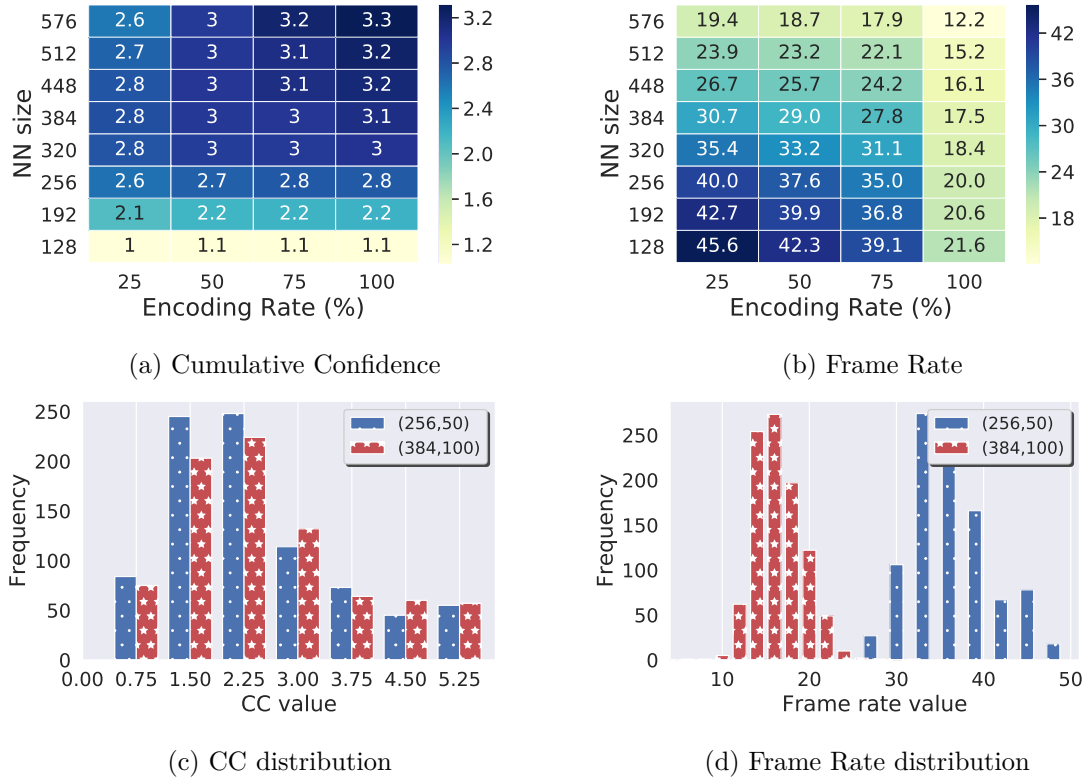
[167] to build models in real-time, thus does not require prior data; *(ii)* jointly configures server, device and network parameters; and *(iii)* employs data-efficient non-parametric learning, hence does not make assumptions about the system. We draw ideas from the area of Automated Machine Learning (AutoML) that streamlines the selection of ML hyper-parameters cf. [169,170,173], also using, lately, Bayesian optimization to improve the overall process [171]. Such techniques have been only recently used in systems, e.g., configuring cloud servers [174] or cellular networks [175]. To the best of our knowledge, this is the first work using this approach to build a platform/data-oblivious optimization framework for multi-user video edge analytics.

8.2 Preliminary Experiments

Previous works, e.g., [18,83,166], have studied similar trade-offs created by such system knobs in an offline setup, i.e., by pre-calculating the average Precision/Recall accuracy for large datasets of images. However, we aim to automate the system configuration at runtime, and hence cannot rely on offline evaluations; instead, we need *instantaneous* feedback for the performance. To that end, we use the Cumulative Confidence (CC) which is simply the sum of confidence values for all recognized objects that is output by YOLO, and is instantly available for each processed frame.

Next, we provide evidence that CC is a suitable *online* metric to other standard offline metrics such as the mean Average Precision (mAP). We used the COCO dataset of 40K images [165] to evaluate the CC and mAP for a series of NN size / encoding rate combinations [3] and depict the results in Fig. 8.1a-8.1b. Observe that the normalized performance of CC and mAP is almost

[3]See also our previous work [166] for details on the measurement methodology for the mAP.



(a) Cumulative Confidence

(b) Frame Rate

(c) CC distribution

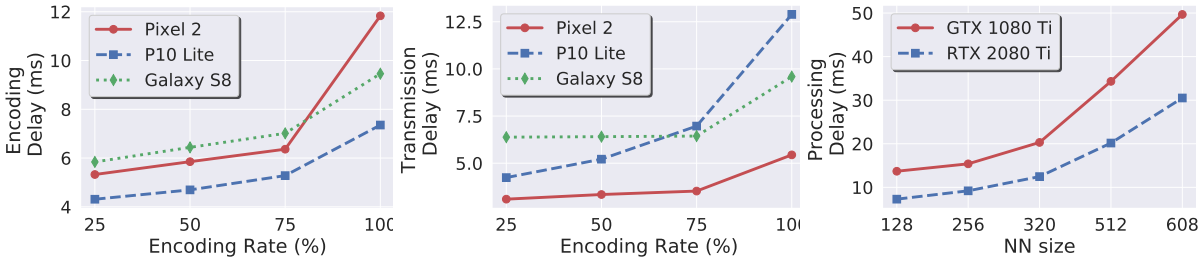
(d) Frame Rate distribution

Figure 8.2: (a)-(b): Cumulative Confidence and frame rate for various NN sizes and encoding rates. (c)-(d): Distributions of CC and frame rate for (NN size, encoding rate): $(256, 50\%)$, $(384, 100\%)$.

identical. By using the normalized values we can directly compare CC and mAP since the former is a sum of confidence values in the range $[0, 1]$ while the latter is a single value in the same range. Intuitively, the CC increases as we both get more objects recognized, and we do so with higher confidence.

We first quantify the trade-off between the CC and service frame rate using the COCO dataset. Figures 8.2a-8.2b depict the average CC and achieved frame rate, for different encoding rates and NN input sizes. It is evident that increasing the NN size and/or encoding rate, increases the CC and decreases the frame rate. Interestingly, we also found in Fig. 8.2a a case of non-increasing impact of the NN size on CC (for 25% encoding rate). Notice that the CC increases with the NN size before dropping for NN size > 448.

The main issue with those results is that they are averages of the system performance and can be obtained only after applying object recognition to thousands of images for each possible system configuration. Indeed, Figures 8.2c-8.2d depict the variations in CC and frame rate for 2 specific configurations. Moreover, the observed increase (decrease) in CC (frame rate)



(a) (b) (c)

Figure 8.3: (a) Encoding and (b) transmission delay for 3 devices. (c) Dependence of the NN processing delay on GPU.

is non-linear with either NN size or encoding rate, and surprisingly, not even monotonic as explained above. The measured performance can also vary depending on factors like the device environment and specifications, channel conditions or server capabilities, making the development of general accuracy and latency models highly cumbersome. We demonstrate the above in Fig. 8.3a-8.3b, where we measure the average encoding and transmission delay respectively, for 3 different mobile devices. Clearly, although all delays are increasing with the encoding rate, the fitted curves vary substantially across devices. The same trend persists when the server's hardware (GPU) changes. Fig. 8.3c depicts the difference in DNN processing delay between 2 GPUs as we increase the NN input size.

In summary, our experiments reveal a non-trivial multimodal impact of the encoding rate and NN size on CC and frame rate. These 2 key metrics are platform and dataset dependent, highly volatile, and there are unknown correlations among the configurations. Hence, it is both important and challenging to find the best system configuration at runtime.

8.3 System Model and Problem Statement

Network and edge service. Our system operates in time slots, each with fixed duration Δ secs. A set $\mathcal{N}$ of N mobile devices are connected to a MEC server that runs a video analytics service, e.g. object recognition, as in Fig. 8.4. Each device $n \in \mathcal{N}$ extract images from the captured video stream, where properties like the number and type of objects in each image vary over time. We denote those properties with $\{o_{nt}\}$ which follows an unknown random process $\{o_{nt}\}_{t=1}^{\infty}$. Each user applies an encoding rate to the captured images selected from finite set $\mathcal{X}$ and transmits them to the server for processing. The average signal to noise ratio (SNR) of

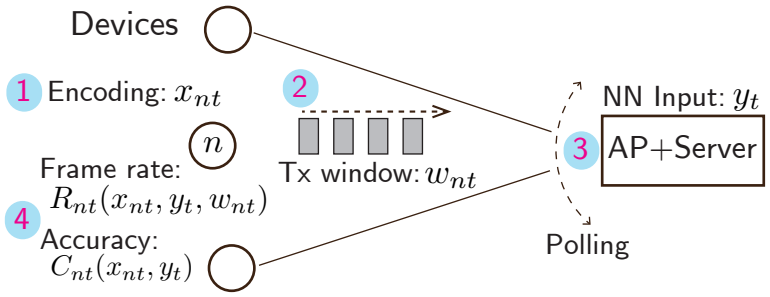


Figure 8.4: Multi-user system operation.

device n during slot t is denoted by h_{nt}, which is calculated by the AP and is given by a random process $\{h_{nt}\}_{t=1}^{\infty}$. Upon reception of an image, the server decodes and downsamples it to fit the input size of the NN that is loaded on its GPU. The possible NN size values are selected from a finite set $\mathcal{Y}$. Note that while each device can apply their own encoding rate, the NN input size is common for all devices as they share the server's GPU[4].

Decision variables. The encoding rate of each device n during time slot t is a system decision variable denoted by $x_{nt} \in \mathcal{X}$. The selection of x_{nt} will determine the resulting image size $s(x_{nt})$, which in turn will affect the transmission time to the server. We denote the fixed bandwidth of the wireless channel by W, and as a result, the Wi-Fi transmission delay of user n during t is:

$$\tau_{nt}(x_{nt}) = \frac{s(x_{nt}) + L}{W \log(1 + h_{nt})}, \tag{8.1}$$

where L is the TCP/UDP stack overhead added to the images.

To enable multi-user connectivity to the server and coordinate transmissions and GPU computations, we introduce the time allocation variable $w_{nt} \in \mathcal{W} \triangleq [0, \Delta]$ as a configurable parameter. By limiting the fraction of time w_{nt} that is allocated to each device n for continuous object recognition, we can guarantee that all devices have the opportunity to send a number of images during t. We compact all variables in $z_t = (x_{1t}, \ldots, x_{Nt}, y_t, w_{1t}, \ldots, w_{Nt}) \in \mathcal{Z} \triangleq \mathcal{X}^N \times \mathcal{Y} \times \mathcal{W}^N$. Note that our system is orthogonal to, and operates at a higher time scale than other underlying wireless mechanisms, e.g., contention control, which run in the scale of milliseconds.

Performance metrics. We define the function $C_n(z_t) : \mathcal{Z} \to \mathbb{R}_+$ to be the expectation of the CC experienced by user n when configuration z_t is applied to the system. In practice

[4]Our experiments showed that changing the NN input size for each user induces delay that impacts performance. In Sec. 8.5 we extend our model to allow different NN size per user whenever many GPUs are available.

however, we can only observe the noisy instantaneous CC achieved during each slot t, that follows a distribution like in Fig 8.2c. This noise is caused due to the varying content of the images expressed by o_{nt} that makes some objects easier/harder to classify than others. We denote the instantaneous CC as $C_{nt}(z_t; o_{nt}) : \mathcal{Z} \to \mathbb{R}_+,\ n \in \mathcal{N}$, and can write it:

$$C_{nt}(z_t; o_{nt}) = C_n(z_t) + \epsilon_1(o_{nt}), \text{ with } \epsilon_1 \sim \mathcal{N}(0, \sigma_1^2). \tag{8.2}$$

In a single user scenario, the frame rate is fully determined by the end-to-end latency of the system. With multiple users however, service is interrupted (see Fig. 8.5 for a 3-user scheduling example). The frame rate we refer to from now on, is the number of images processed for each user during a slot of length Δ, e.g. with respect to Fig. 8.5 we have 4 frames per slot for user 1, 3 frames for user 2 and 2 for user 3. Similar to CC, we define the average frame rate of device n by $R_n(z_t) : \mathcal{Z} \to \mathbb{R}_+$, that depends on all variables. Function $R_n(z_t)$ is also an average value that can vary over time due to varying channel conditions h_{nt} of each user, as shown in Fig 8.2d. We denote the noisy frame rate observed during slot t by $R_{nt}(z_t; h_{nt}) : \mathcal{Z} \to \mathbb{R}_+$ defined as:

$$R_{nt}(z_t; h_{nt}) = R_n(z_t) + \epsilon_2(h_{nt}), \text{ with } \epsilon_2 \sim \mathcal{N}(0, \sigma_2^2). \tag{8.3}$$

Note that the above frame rate is directly measured by the server since it is responsible for processing all user images. Hence, it keeps count of the number images processed for each user n during their allocated air time w_{nt}. Finally, each device n sets a minimum frame rate threshold λ_n based on their preferences or requirements. We consider the general case where these can differ across users.

User scheduling. If we allow the users to concurrently send sequences of images, we face the problem of interference and queuing at the server, since only one image can be processed at each time. In detail, if we consider a shared medium (previous versions of WiFi) we want to avoid the users to collide in their transmissions, i.e., avoid the *contention phase*, which will delay the service pipeline (encoding-transmission-decoding-processing) shown in Fig. 8.5. Second, if we consider the latest WiFi standard (802.11ax), which is based on OFDMA, several users can transmit using different sub-bands without colliding. In that case our aim is to prevent the server queue to grow infinitely. Such problems can deteriorate the analytics performance since the end-to-end latency of a single image can increase, and thus the information overlaid

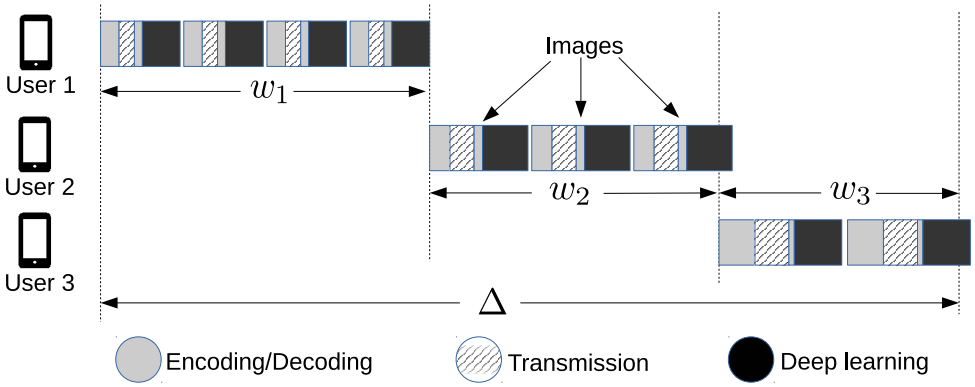


Figure 8.5: Task scheduling example for 3 users.

on the user's screen can be *substantially outdated*. To avoid that, we let the server apply a polling scheme, so that each user $n \in \mathcal{N}$ executes the entire processing pipeline (in both the mobile, the network and server) without interruptions, for the duration of its allocated time w_{nt}, using the selected configuration x_{nt}, y_t. Based on the values selected for $w_{nt}, n \in \mathcal{N}$, the server communicates to the users the timestamp at which they are supposed to start and end their transmissions in order to respect the scheduling scheme. This can occur any time the server sends object recognition results back to the users.

Problem formulation. Our aim is to maximize the CC of users while respecting their frame rate requirements. We define the observed CC across the users as $f_t(z_t) = \sum_{n\in\mathcal{N}} C_{nt}(z_t; o_{nt})$, and the constraints $g_{nt}(z_t) = \lambda_n - R_{nt}(z_t; h_{nt}),\ n \in \mathcal{N}$. Ideally, we would like to find the optimal solution $z^* = (x_1^*, \ldots, x_N^*, y^*, w_1^*, \ldots, w_N^*)$ to the following problem:

$$\mathbb{P}: \underset{z\in\mathcal{Z}}{\text{maximize}} \quad \mathbb{E}\{f_t(z)\} \tag{8.4a}$$

$$\text{subject to:} \quad \mathbb{E}\{g_{nt}(z)\} \leq 0,\ n \in \mathcal{N} \tag{8.4b}$$

$$\sum_{n\in\mathcal{N}} w_n \leq \Delta \tag{8.4c}$$

Observe that applying the expectations in (8.4a)-(8.4b), by using (8.2)-(8.3), yields the unknown average functions, i.e.,

$$\mathbb{E}\{f_t(z)\} = \sum_{n\in\mathcal{N}} C_n(z), \quad \mathbb{E}\{g_{nt}(z)\} = \lambda_n - R_n(z).$$

Also, constraints (8.4b), (8.4c) ensure that the frame rate thresholds are respected, and that the time allocation is valid.

Clearly, $\mathbb{P}$ cannot be solved directly since functions $C_n(\cdot)$ and $R_n(\cdot)$ are unknown. Therefore, we follow an *online learning approach* where we select configurations z_t at the beginning of each slot t and calculate the perturbed outputs $f_t(z_t), g_{nt}(z_t)$ using the noisy measurements $C_{nt}(z_t; o_{nt})$, $R_{nt}(z_t; h_{nt})$. Our goal then is to find a sequence of configurations $\{z_t\}_t$ that will drive the average performance close to $\mathbb{E}\{f_t(z^*)\}$, while satisfying (probabilistically) the constraints $g_{nt}(z_t), \forall n$ and (8.4c) at each slot. Formally, we define the *pseudo-regret*:

$$Reg_T = \sum_{t=1}^{T} \mathbb{E}\{f_t(z^*)\} - \sum_{t=1}^{T} \mathbb{E}\{f_t(z_t)\}, \tag{8.5}$$

and require that sequence $\{z_t\}_t$ achieves sublinear average regret, $\lim_{T\to\infty} Reg_T/T = 0$. This will ensure that our policy learns to perform as well as the hypothetical benchmark z^* which can only be designed in hindsight, i.e., with complete knowledge of the platform functions and data.

8.4 Gaussian Processes and Problem Solution

In this section, we start with the necessary background and then present in detail the online algorithm and its properties.

8.4.1 MAB formulation through GP modeling

Due to the online nature of our problem, we address it following a Multi-armed Bandit (MAB) approach, by which we sequentially select different configurations (arms) to tackle the exploration-exploitation dilemma. However, most of classic MAB algorithms such as UCB [176] and Thompson Sampling [177], do not consider that nearby arms can be correlated, i.e., they yield similar performance; or assume these correlations to be known in advance, or to have a specific (e.g., linear) structure [178,179]. Nevertheless, as the experiments in Sec. 8.2 showed, the system configurations exhibit unknown, varying and even non-monotonic performance correlations.

In fact, these correlations could be fully characterized by the objective and constraint functions in $\mathbb{P}$, provided they were known. To rectify this, we use Gaussian Processes which is a model-free (or, assumption-free) approach requiring only a certain level of function smoothness

[167], something we already validated with our measurements. A kernel function $\rho(z, z')$ is used to express the correlation between the objective/constraint function value of any pair of configurations (z, z') and enables predictions about the function evaluation at any vector $z \in \mathcal{Z}$.

Following this approach, the seminal GP-UCB algorithm [180] was applied to *unconstrained problems* where the objective function is iteratively approximated using noisy observations, much like in our setup with the difference of constraints. The benefit of this approach is that it estimates the mean value of $f_t(z)$ for any z by only using the rewards observed up to t, including configurations that have not been applied in the past. In specific, if $\mathcal{A}_t = \{z_1, \ldots, z_t\}$, $\mathcal{F}_t = \{f_1(z_1), \ldots, f_t(z_t)\}$ are the applied configurations and respective rewards up to slot t, the mean value and covariance of $f_t(z)$ for any configuration (or, action) z are given by:

$$\mu_{f,t}(z) = \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_1^2 \boldsymbol{I}_t)^{-1} \mathcal{F}_t, \tag{8.6}$$

$$k_{f,t}(z, z') = \rho(z, z') - \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_1^2 \boldsymbol{I}_t)^{-1} \boldsymbol{k}_t(z'), \tag{8.7}$$

where $\boldsymbol{k}_t(z) = (\rho(z_1, z), \ldots, \rho(z_t, z))^\top$, $\boldsymbol{K}_t = (\rho(z_t, z_{t'}))$ is the positive definite kernel matrix, and $\boldsymbol{I}_t$ the identity matrix with dimension t. GP-UCB selects the next action based on a weighted acquisition rule:

$$z_{t+1} = \arg\max_{z \in \mathcal{Z}} \mu_{f,t}(z) + \beta_t \sqrt{k_{f,t}(z, z)},$$

where β_t is a problem-related parameter, and it provably achieves sublinear expected (or, pseudo) regret [180], for this ensures an efficient sampling, or active-learning, trajectory in the configuration space.

8.4.2 Constrained GP-based MAB optimization

In order to find configurations that progressively increase system performance, *and* do so without violating the frame rate thresholds, we need a twofold extension of GP-UCB. There are only few works that proposed similar ideas for *safe* GP-UCB algorithms, e.g., [181–183]. Following a similar approach, we design a learning algorithm with 2 stages: the *expansion stage* (for T_0 slots) and the *optimization stage* (for $T - T_0$ slots). In the former, given an initial safe set of configurations S_0, i.e., actions guaranteed to satisfy the thresholds, we successively create enlarged safe sets S_t by adding configurations that conservatively (by means of upper bounds) also respect the constraints. After we reach a satisfactory approximation of the maximum

achievable safe set, we commence the optimization stage where we apply the upper confidence bound (UCB) rule on that set, much like in GP-UCB [180].

In detail, we use GPs to model the constraints, as we do for the objective, and evaluate their posteriors using the past observations $\mathcal{G}_{nt} = \{g_{n\tau}(z_\tau)\}_{\tau=1}^{t}$ as:

$$\mu_{n,t}(z) = \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_2^2 \boldsymbol{I}_t)^{-1} \mathcal{G}_{nt}, \tag{8.8}$$

$$k_{n,t}(z, z') = \rho(z, z') - \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_2^2 \boldsymbol{I}_t)^{-1} \boldsymbol{k}_t(z'). \tag{8.9}$$

We also use the upper and lower confidence bounds (UCBs, LCBs) for the constraint and objective functions:

$$u_t^i(z) = \mu_{i,t}(z) + \beta_t \sqrt{k_{i,t}(z, z)}, \ i = f, 1, \ldots, N \tag{8.10}$$

$$l_t^i(z) = \mu_{i,t}(z) - \beta_t \sqrt{k_{i,t}(z, z)}, \ i = f, 1, \ldots, N \tag{8.11}$$

where β_t is an increasing with t scalar (discussed below).

Regarding the safe set expansion stage, if we knew the constraint functions it could be achieved by performing the operation:

$$V_\zeta(S_t) = S_t \cup \bigcap_{n \in \mathcal{N}} \Big\{ z \in \mathcal{Z} \ | \exists \ z' \in S_t : g_n(z') + \zeta + M_n \|z - z'\|_2 \leq 0 \Big\}, \ t = 0, 1, 2 \ldots$$

where M_n is the Lipschitz constant of g_n, and ζ a tunable tolerance parameter. Essentially, we would expand S_t by including points z that are close enough to previous safe points z' such that they also satisfy the constraints. We denote with $S_{max}^{\zeta} \triangleq \lim_{t \to \infty} V_\zeta(S_t)$ the *maximum reachable* set through this operation, and S_{max} the *maximum possible* safe set that we obtain for $\zeta = 0$. Yet, since we do not know the constraint functions we follow a different approach.

Namely, we use instead the UCBs and the expansion rule:

$$S_t = \bigcap_{n \in \mathcal{N}} \bigcup_{z \in S_{t-1}} \Big\{ z' \in \mathcal{Z} \ | \ u_t^n(z) + M_n \|z - z'\|_2 \leq 0 \Big\} \tag{8.12}$$

and employ the updated safe set S_t to create a second set $G_t \subseteq S_t$ that contains configurations

Algorithm 7 Automatic configuration of video analytics

Initialize: $S_0 \subset \mathcal{Z}, z_1 \in S_0, \mathcal{A}_0, \mathcal{F}_0, \mathcal{G}_{n0} = \emptyset, \rho(z, z'), M_n,\ \lambda_n > 0$ and β_t
for $t = 1, \ldots, T$ **do**
  Process images and obtain: $f_t(z_t), g_{nt}(z_t),\ n \in \mathcal{N}$
  $\mathcal{A}_t \leftarrow \mathcal{A}_{t-1} \cup \{z_t\}$
  $\mathcal{F}_t \leftarrow \mathcal{F}_{t-1} \cup \{f_t(z_t)\}$
  $\mathcal{G}_{nt} \leftarrow \mathcal{G}_{nt-1} \cup \{g_{nt}(z_t)\},\ n \in \mathcal{N}$
  Update posteriors of $z \in S_t$ using (8.6)-(8.9)
  if $t \leq T_0$ **then**
    $S_t \leftarrow \cap_n \cup_{z \in S_{t-1}} \{z' \in \mathcal{Z} | u_t^n(z) + M_n \|z - z'\|_2 \leq 0\}$
    $G_t \leftarrow \{z \in S_t \mid e_t(z) > 0\}$
    if $\max_{z \in G_t} \left(u_t^n(z) - l_t^n(z)\right) < \zeta, \forall n \in \mathcal{N}$ **then**
      $z_{t+1} \leftarrow \arg\max_{z \in S_t} u_t^f(z)$
    else
      $z_{t+1} \leftarrow \arg\max_{z \in G_t} \left(u_t^n(z) - l_t^n(z)\right), n \in \mathcal{N}$
    end if
  else
    $S_t \leftarrow S_{t-1}$
    $z_{t+1} \leftarrow \arg\max_{z \in S_t} u_t^f(z)$
  end if
end for

which not only are safe but can lead to further expansion. For that, we define:

$$e_t(z) = \Big| \bigcap_{n \in \mathcal{N}} \{z' \in \mathcal{Z} \backslash S_t \mid l_t^n(z) + M_n \|z - z'\|_2 \leq 0\} \Big|, \tag{8.13}$$

and then build $G_t = \{z \in S_t \mid e_t(z) > 0\}$. Finally, if the configurations in G_t are still uncertain enough in terms of their possible values, i.e. $\max_{z \in G_t}(u_t^n(z) - l_t^n(z)) \geq \zeta, \forall n$, we select the most uncertain $z_{t+1} = \arg\max_{z \in G_t}(u_t^n(z) - l_t^n(z))$. Otherwise, we select the configuration with the highest UCB, i.e., $z_{t+1} = \arg\max_{z \in S_t} u_t^f(z)$. In that case, we have found a good approximation for the safe set, i.e., close enough to the maximum reachable set S_{max}^{ζ}, and can continue in the optimization stage. All steps are shown in Algorithm 7.

8.4.3 Theoretical results

The effectiveness of Algorithm 7 relies on the accurate estimation of sets S_t and G_t. Specifically, we want to conservatively expand the safe set in order to guarantee the feasibility of its configurations. On the other hand, if the expansion is too conservative, we will need many iterations to reach the set of all safe configurations S_{max}^{ζ}. This trade off is controlled by parameter

β_t which is chosen as [182]:

$$\beta_t = B + \sigma_1\sqrt{2(1+\gamma_{t-1}+\log(1/\delta))}, \tag{8.14}$$

In the above, B is an upper bound on the Reproductive Kernel Hilbert Space (RKHS) norm of f and g_n, while δ is the allowed *constraint violation probability*. Parameter γ_t is the maximum mutual information gain that can be obtained about the prior of f, after t samples have been observed [180]:

$$\gamma_t = \max_{A\subset Z,|A|=t} = \frac{1}{2}\log|\boldsymbol{I}+\sigma_1^{-2}\boldsymbol{K}_A|,$$

where $\boldsymbol{K}_A = [\rho(z,z')],\ z,z'\in A$ is the covariance matrix of the samples collected after t slots. Evidently, γ_t is very difficult to obtain in practice, but a conservative bound is given in [181] for the case of finite $\mathcal{Z}$ as

$$\gamma_t \leq |\mathcal{Z}|\log\left(1+\sigma_1^{-2}t|\mathcal{Z}|\max_{z\in\mathcal{Z}}k_{f,t}(z,z)\right).$$

We employ the Matern kernel function with parameter $\nu = 3/2$, which implies that our functions are at least once differentiable [167]. The kernel is given by:

$$\rho(z,z') = \Big(1+\frac{\sqrt{3}}{l}\|z-z'\|_2\Big)\Big(\exp(-\frac{\sqrt{3}}{l}\|z-z'\|_2\Big),$$

where l is a length scale parameter.

Next, we formally present the convergence properties of the safe set (expansion stage) and the average observed reward (optimization stage), to S^{ζ}_{max} and $\mathbb{E}\{f_t(z^*)\}$, respectively. For the former, what we need to do is find the minimum T_0 in the problem's time horizon T that guarantees this convergence. This is described as follows:

Lemma 3 *Given an initial safe set $S_0 \neq \emptyset$ such that $g_n(z)\leq 0,\ \forall z\in S_0, n\in\mathcal{N}$, fix any $\zeta>0$ and $\delta\in(0,1)$, choose β_t as in (8.14), and $\gamma_t = |\mathcal{Z}|\log(|\mathcal{Z}|t)$. The safe set expansion stage of Algorithm 7 guarantees with probability $1-\delta$ that only safe actions are included to the safe set at any time. Moreover, the expanded set S_t will reach the maximum safe set S^{ζ}_{max} if we select T_0 to be the smallest integer for which:*

$$\frac{T_0}{\beta^2_{T_0}|\mathcal{Z}|\log(|\mathcal{Z}|T_0)} \geq \frac{8(|S_{max}|+1)}{\zeta^2\log(1+\sigma_1^2)}.$$

Proof The proof is based on Theorem 1 in [182] where we apply the bound on the information gain γ_t. This is possible since in our setup the action set $\mathcal{Z}$ is always finite.

The next theorem characterizes the algorithm's convergence, and how its regret depends on the system parameters.

Theorem 1: Regret of Algorithm 7

Given an initial safe set $S_0 \neq \emptyset$ such that $g_n(z) \leq 0 \ \forall z \in S_0$, $n \in \mathcal{N}$, fix $\delta \in (0,1)$, and choose β_t as in (8.14). Algorithm 7 yields sublinear regret of $\mathcal{O}(\sqrt{T}|\mathcal{Z}|\log(|\mathcal{Z}|T))$ with probability $1-\delta$. In specific:

$$Reg_T \leq 4B\sqrt{(T+2)\gamma_T} + \gamma_T\sqrt{(T+2)(\alpha/\gamma_t+1)},$$

where $\alpha = 1 + \log(1/\delta)$, and $\gamma_T = |\mathcal{Z}|\log(|\mathcal{Z}|T)$.

Proof By the definition of regret we have

$$\begin{aligned} Reg_T &= \sum_{t=1}^{T} \mathbb{E}\{f_t(z^*)\} - \mathbb{E}\{f_t(z_t)\} \\ &\leq \sum_{t=1}^{T} \mu_{f,t}(z_t) + \beta_t\sqrt{k_{f,t}(z_t,z_t)} - \mathbb{E}\{f_t(z_t)\} \leq 2\beta_T \sum_{t=1}^{T}\sqrt{k_{f,t}(z_t,z_t)} \end{aligned}$$

where we used the upper and lower bounds (8.10), (8.11) and the fact that β_t is an increasing parameter. From Lemma 4 in [184] we have that $\sum_{t=1}^{T}\sqrt{k_{f,t}(z_t,z_t)} \leq \sqrt{4(T+2)\gamma_T}$ hence we obtain

$$\begin{aligned} Reg_T \leq 2\beta_T\sqrt{4(T+2)\gamma_T} &\leq 4(B+\sqrt{2}\sigma_1\sqrt{\alpha+\gamma_T})\sqrt{(T+2)\gamma_T} \\ &= 4B\sqrt{(T+2)\gamma_T} + \gamma_T\sqrt{(T+2)(\alpha/\gamma_t+1)}. \end{aligned}$$

Observe that the largest (second) term of the bound yields a regret growth of $\mathcal{O}(\sqrt{T}\gamma_T)$ and by the selection of γ_T we have $\mathcal{O}(\sqrt{T}|\mathcal{Z}|\log(|\mathcal{Z}|T))$.

Discussion. The above result shows that the cumulative regret does not grow indefinitely and the algorithm selects configurations towards increasing the obtained rewards. The performance of the algorithm depends on parameters such as ζ which allow us to set the optimization accuracy

Algorithm 8 GP-UCB

Initialize: $z_1 \in \mathcal{Z}, \alpha, \rho(z, z'), \lambda > 0,\ s$, and β_t
for $t = 1, 2, \ldots$ **do**
  Process images and obtain: $f_t(z_t) \leftarrow C_t(z_t; o_t) - \alpha|R_t(z_t; h_t) - \lambda|$
  $\mathcal{A}_t \leftarrow \{z_1, \ldots, z_t\}$
  $\mathcal{F}_t \leftarrow \{f_1(z_1), \ldots, f_t(z_t)\}$
  for $z \in \mathcal{Z}$ **do**
    $\mu_t(z) \leftarrow \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I}) \mathcal{F}_t$
    $k_t(z, z) \leftarrow \rho(z, z) - \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(z)$
  end for
  $z_{t+1} \leftarrow \arg\max_{z \in \mathcal{Z}} \mu_t(z) + \sqrt{\beta_t k_t(z, z)}$
end for

– increasing it reduces the expansion time T_0 but shrinks the range of considered configurations (by the algorithm and the benchmark); while reducing parameter δ improves the violation and regret bound probabilities but deteriorates the regret bound. Finally, note that all bounds are probabilistic, hence the term *pseudo*-regret.

8.5 Extensions and Practical Considerations

Next, we present important extensions of our system model, and also describe implementation issues that allow the practical deployment of Algorithm 7.

Unconstrained Multi-objective version. We firstly study a simpler version of the problem, where we consider a single user, and the frame rate constraints are embedded into the objective [185]. In detail, we only have a tuple of configurable parameters $z_t = (x_t, y_t)$ [5] that denote the selected encoding rate and NN size and a unified objective defined as:

$$f_t(z_t) = C_t(z_t; o_t) - \alpha|R_t(z_t; h_t) - \lambda|,$$

where λ is the desired frame rate and α balances between maximizing the CC and achieving a frame rate close to the desired λ. In that case, constraints (8.4b), (8.4c) are not needed, and we only need one GP to estimate the objective. The algorithm used, to find the optimal solution in this simpler version is given in Algorithm 8, and reduces to the standard GP-UCB algorithm.

Transmission control and sequencing. Besides scheduling the users, in many scenarios it is crucial to guarantee a low maximum delay between consecutive scheduling sequences of each

[5] Note that the n indices are missing as we only consider one user.

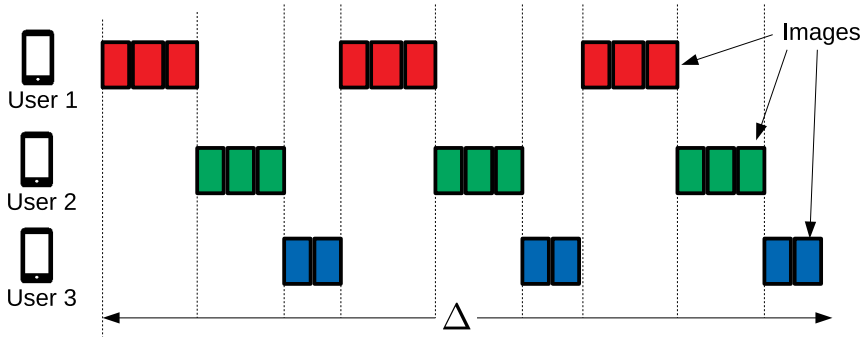


Figure 8.6: User scheduling with $k = 3$ subintervals for smoother user experience.

user. In Fig. 8.5 for example, this maximum delay for user 1 is equal to $w_2 + w_3 = \Delta - w_1$. A way to reduce this delay is to divide the slot into smaller subintervals, e.g. k sub-slots of duration Δ/k, which will effectively reduce the inter-arrival delay by a factor of k. This is demonstrated in Fig. 8.6, where the images of each user are highlighted with different color, effectively reducing the delay between non-consecutive images for the users and providing a smoother experience. However, if the number of users is too big the delay may not be brought down to desirable values even then. Alternatively, one might resort to interleaving transmissions for users with high performance requirements, and break the scheduling pattern. Such sequencing adjustments can be applied to our framework, since the server dictates the service sequence of the users, and those orthogonal to the parameters selected by our algorithm. Moreover, note that this framework operates at a higher time scale than typical wireless mechanisms, e.g. power allocation, which run in a much smaller time scale. These decisions are also orthogonal to the video pipeline configuration, and are essentially latent factors, the effect of which is incorporated through our Bayesian updates.

Additional configurations. In our prototype we experimented with the NN size, encoding rate and airtime. Nevertheless, other video processing pipelines involve parameters such as the frame resolution [19, 83], NN model and number of NN layers [18, 84, 94], or the maximum power the GPU can use [186]. These parameters eventually trade off frame rate for CC, just like the encoding rate and NN input size in our application, and our framework can be readily extended to account for these options. For example, consider the case where we can select the users' frame resolution p_{nt} from a finite set $\mathcal{P}$, on top of the encoding rate. Image size $s(x_{nt}, p_{nt})$ and

transmission delay $\tau_{nt}(x_{nt}, p_{nt})$ would be bi-variate functions and thus we would have:

$$\tau_{nt}(x_{nt}, p_{nt}) = \frac{s(x_{nt}, p_{nt}) + L}{W \log(1 + h_{nt})},$$

while the configuration vector would be $z_t{=}(x_{1:Nt}, y_t, w_{1:Nt}, p_{1:Nt})$, where we use shorthand notations $\alpha_{1:Nt}$ for vectors $(\alpha_{1t}, \ldots, \alpha_{Nt})$. Similarly, if we can select among $\mathcal{L}$ NN models that differ on, e.g., their training data, this vector becomes $z_t = (x_{1:Nt}, y_t, w_{1:Nt}, l_t), l_t \in \mathcal{L}$. Such extensions increase the configuration space and this can impact the convergence speed, which nevertheless is guaranteed. We evaluate this scenario in Sec. 8.6.

Controlling computing resources. On the other hand, some systems offer access to allocating their computing resources or have multiple GPUs (see Fig. **??**). Hence, we would be able to allocate a GPU, and as a result a distinct NN input size y_{nt} for each user n. The cost would be again an increased action space, namely $z_t = (x_{1:Nt}, y_{1:Nt}, w_{1:Nt})$, but the total user load would be distributed among a larger number of GPUs, allowing better system performance. Furthermore we can introduce assignment variables to allocate multiple users to multiple GPUs and/or Access Points (AP). In specific, consider that the server has K available GPUs and the users can connect to it through J APs, resulting in a joint GPU/AP assignment decision vector, i.e. $z_t = (x_{1:Nt}, y_{1:Kt}, w_{1:Nt}, v_{1:Nt})$, where v_{nt} is the association decision for user n, i.e. a tuple (j, k) that denotes n is served by AP j and GPU k. This way we can support higher frame rates for the users since the resource availability scales. However, as computation complexity increases with the action space, a single edge node might not be able to make the configuration decisions on time for the next slot. In such cases, a node with higher computing power (orchestrator) can receive the action and reward/constraint history $\mathcal{A}_t, \mathcal{F}_t, \mathcal{G}_{nt}$ and update some or all of the GPs, while the edge nodes are solely responsible for serving the users, reporting the measurements to the orchestrator, and applying the actions it receives back.

Implementation issues. In many settings some of the algorithm's parameters might be unknown. For instance, an upper bound for the norms of f, g_n is difficult to compute with no/incomplete data. The same is true regarding the Lipschitz constants M_n. In practice, we can compute the former during a small initialization period, or rely on historic data. For the latter

we can use a modified rule for the expansion stage [182], where we replace (8.12), (8.13) with:

$$S_t = \bigcap_{n\in\mathcal{N}} \{z \in \mathcal{Z} \mid u_t^n(z) \leq 0\}, \text{ and } e_t(z) = \Big| \bigcap_{n\in\mathcal{N}} \{z' \in \mathcal{Z}\backslash S_t \mid l_t^n(z) \leq 0\}\Big|,$$

where we simply use the upper/lower confidence bounds. The drawback is that we need to calculate the posteriors for all $z \in \mathcal{Z}$, not just the ones already in S_t, and essentially compute a new safe set at each iteration.

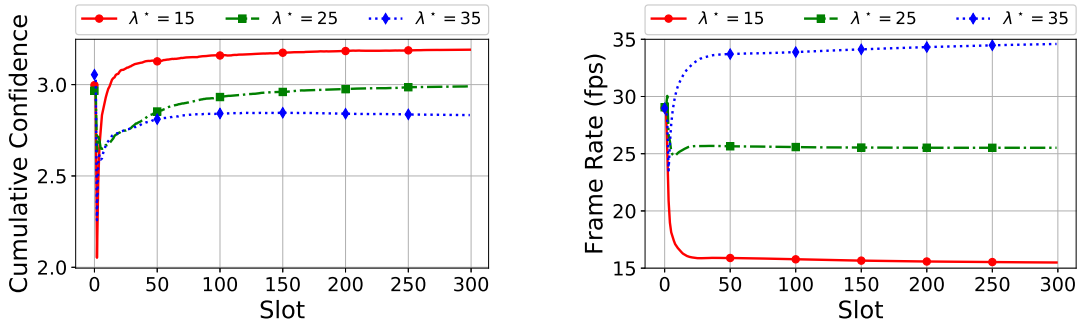
8.6 Performance Evaluation

We consider the sets $\mathcal{X}=\{25, 50, 75, 100\}$, and $\mathcal{Y} = \{128, 192, 256, 320, 384, 448, 512, 576\}$, and provide the respective measurements obtained from our testbed in [172]. We quantize the time allocation decisions w_{nt} so that our configuration space $\mathcal{Z}$ is finite. In specific, we define $\mathcal{W} = \{.1, .2, .3, .4, .5, .6, .7, .8, .9\}$ and $\Delta=5$ sec, so that $w_{nt}=0.5$ means that the time allocated to device n during t is 2.5 sec. For the construction of the initial safe set S_0, we only use configurations that include the lowest NN size and encoding rate, i.e. 128 and 25% respectively, since if the problem is feasible, these parameters will definitely satisfy the constraints. We used the measurements and the model of Sec. 8.3 to evaluate Algorithm 7 in finding the optimal configuration of a multi-user system with diverse frame rate constraints. The channel bandwidth is $W=40$ MHz and each user's mean SNR is selected from a uniform distribution in $[10, 35]$ dB. This mean is then used to sample the SNR h_{nt} at each slot from a Gaussian distribution.

8.6.1 Single User and Multi-objective Scenario

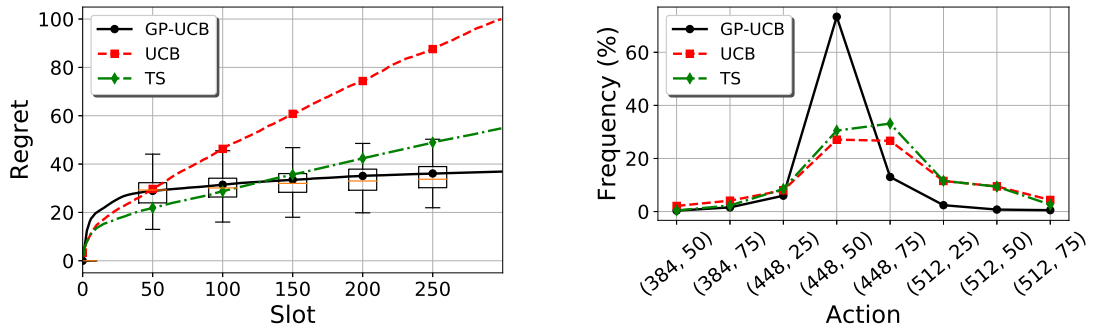
We evaluate the performance of Algorithm 8 with respect to the performance metrics, i.e. CC and frame rate, as well as the optimality measure, i.e. the regret. We compare the latter, with the regret achieved by standard algorithms used in the MAB literature like UCB [176] and Thompson Sampling (TS) [177]. In brief, UCB selects actions based on a combination of expected reward and selection frequency of each action. TS randomly samples a (Gaussian for our problem) distribution for each action, based on the observed rewards, and selects the one with the highest outcome.

Fig. 8.7a-8.7b display the running average of CC and frame rate (defined as $1/t\sum_{\tau=1}^{t} C_\tau(x_\tau)$ and $1/t\sum_{\tau=1}^{t} \lambda_\tau(x_\tau)$ respectively) achieved by Algorithm 8 for different values of the target



(a) (b)

Figure 8.7: Running average of Cumulative Confidence and frame rate of GP-UCB for different values of the desired rate $\lambda^{\star}$.

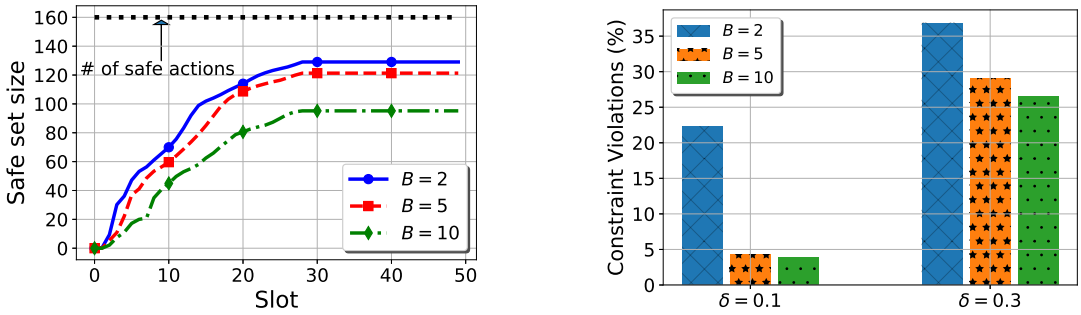


(a) (b)

Figure 8.8: (a) Regret comparison under $\lambda^{\star} = 25$ fps. (b) Action selection frequencies for the different algorithms.

frame rate $\lambda^{\star}$. The efficiency of the algorithm is increased as the CC increases and the frame rate is close to the rate $\lambda^{\star}$. As expected, for higher $\lambda^{\star}$, the CC gets smaller to allow a (higher) frame rate, closer to the desired $\lambda^{\star}$. Moreover, we observe that after the initial slots where the algorithm explores the correlation between actions, the CC increases until it reaches a stable average value, while the frame rate continues to approach $\lambda^{\star}$, as the algorithm keeps selecting high reward actions more frequently.

We depict the cumulative regret of GP-UCB, UCB and TS algorithms in Fig. 8.8a. Compared to UCB and TS, GP-UCB shows at first higher regret, since it needs to explore various areas of the action space to reduce the initially high uncertainty. However, it quickly manages to locate the high reward actions and after 150 slots its regret is lower than both UCB and TS. The latter, also outperforms UCB, since it takes advantage of the Gaussian distribution of the rewards, while UCB tends to explore low reward actions that have not been selected too often before.



(a) (b)

Figure 8.9: (a) Safe set expansion during the first stage of Algorithm 7. (b) Constraint violation percentage for different values of B, δ.

The frequency of selected actions is displayed in Fig. 8.8b, only for the actions selected for more than 1% of the time. This way we overlook purely exploratory actions, and focus on those that the algorithms deem worth exploiting. Note that actions (448,50) and (448,75) are the ones yielding the highest rewards. Observe that one of these actions is selected by GP-UCB for about 86% of the time while for UCB and TS this percentage is only 54% and 64% respectively. Instead they apply lower reward actions more frequently, e.g. (512,25), resulting in higher long term regret.

8.6.2 Parameter Analysis

We first study the impact of parameter β_t. There are several practical limitations when setting β_t, e.g., B is difficult to obtain a-priori and is application specific. Importantly, the value for B impacts the safe set expansion stage since it controls how conservative or slack we are in adding configurations to the safe set. In addition, parameter δ determines the constraint violation probability which is related to the correctness of S_t and how likely it is for relatively unsafe actions to be selected. Fig. 8.9a depicts the size evolution of the safe set over time versus B. We empirically select $T_0 = 30$ for this case since, even before $t = 30$, S_t is stabilized. We calculated (offline) that the number of configurations that satisfy $g_n(z) \leq 0,\ \forall n$ is 160. We observe that as we increase B the algorithm becomes more conservative in expanding the safe set. In specific, we have that $|S_t|$ for $B = 2$ is 80.6% of the actual safe set, while for $B = 10$ it is only 59.4%, meaning that many high reward actions will not be considered in the optimization stage.

Next, we evaluate the impact of B and δ on the constraints violation probability. Note that

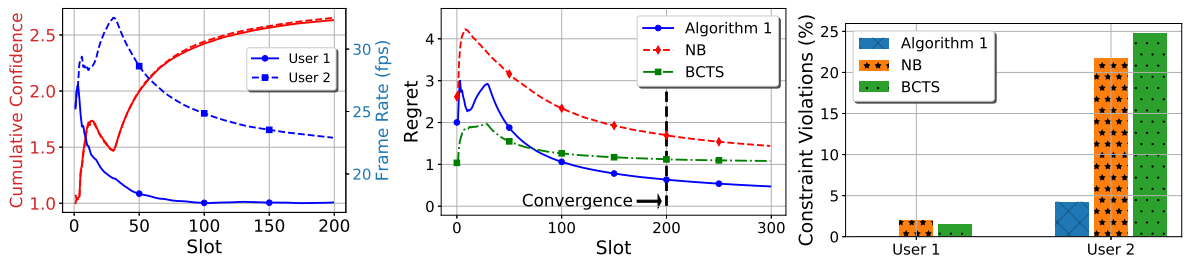
in order to actually achieve violation probability at most δ, parameter B has to be chosen such that $\|g_n\|_k^2 \leq B$. Fig. 8.9b displays the constraint violation probability over 200-slot simulations. We consider probabilities 0.1 and 0.3 for δ, and $B \in \{2, 5, 10\}$, as before. Notice that for $B = 2$ the violation probability increases beyond 10% and 30% respectively, indicating that the selection of B is too low to satisfy the desired probability. In the following we select $\delta = 0.1$ and $B = 5$ to enable low constraint violation probability and a relatively large safe set.

8.6.3 Results

In order to evaluate the performance of Algorithm 7, we use a for 2-user system where $\lambda_1 = 10, \lambda_2 = 20$ fps, and plot each user's achieved average CC and frame rate over time in Fig. 8.10a. The figure shows $1/t \sum_{k=1}^{t} C_{nk}(z_k; o_{nk})$ and $1/t \sum_{k=1}^{t} R_{nk}(z_k; h_{nk})$, $\forall t$, for the 2 users in each of the y-axes. Observe that during the expansion stage, i.e. $t \leq 30$, we have a rather random performance since the goal there is only to locate safe actions. For $t > 30$ however, the algorithm takes improved actions for both users, resulting in an increasing CC. These actions are at the edge of the safe set and hence they are "riskier" resulting in a controlled drop of the average frame rate, which is always above each user's threshold λ_n. Interestingly, we observe that the frame rate of user 1 is well above the target $\lambda_1 = 10$ fps. The optimal encoding rate for user 1 under this setup is 100%, an option that is not included in the safe set since it marginally satisfies the constraint. This results in much higher frame rates via the convergence to lower encoding rates, i.e. 50% or 75%. Additionally, the achieved CC is almost identical for both users, which indicates that the differentiation in time allocation rather than encoding rate is what differentiates the users' frame rates, since the former does not affect the CC.

We evaluate the performance of Algorithm 7 compared to other benchmarks using the average regret Reg_t/t in Fig. 8.10b. We select two state-of-the-art algorithms as competitors. The first one is based on Deep Neural Networks (DNN) and the second one is an online learning algorithm that handles constraints. The competitor algorithms are described as follows.

• *NeuralBandit (NB).* We have designed this algorithm based on the ideas in [187]. Our objective is to assess the performance of another function approximator instead of the GPs we use in Algorithm 7. For that purpose, we use a feedforward DNN to approximate the reward (output) as a function of an action (input). However, a single feedforward DNN cannot provide the uncertainty over the function estimation and therefore Algorithm 7 cannot be directly used



(a) (b) (c)

Figure 8.10: (a) Cumulative Confidence and frame rate of 2 users with $\lambda_1 = 10$, $\lambda_2 = 20$. (b) Average regret performance and (c) constraint violations for the competitor algorithms.

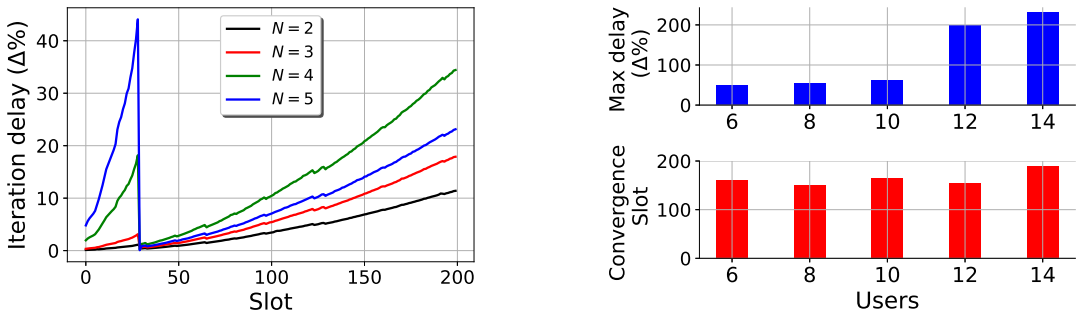
with this function approximator. For that reason, we adopt a common approach in which the reward function is redefined by including the constraints as penalty terms [188, 189]:

$$f_t(z_t) = \sum_{n \in \mathcal{N}} C_{nt}(z_t) - \alpha \max\left(0, \max_{n \in \mathcal{N}} \left(\lambda_n^* - R_{nt}(z_{nt})\right)\right), \tag{8.15}$$

where α is used to determine the constraint violation weight that is attributed to the reward. Note that if α is big enough the maximum average reward of (8.15) will be for the optimal action of problem $\mathbb{P}$. In each slot, the DNN is retrained with the information obtained up to the current slot and all inputs (actions) are evaluated to predict the respective rewards. We use ε-greedy as an acquisition function, i.e., at each slot, we select the action with the highest predicted reward with probability $1 - \gamma$, or a random action with probability γ. We set $\alpha = 0.5$ and $\gamma = 0.2$, that show in our simulations a good exploration-exploitation balance.

- *Behavioral Constrained Thompson Sampling (BCTS)* [190] is a state-of-the-art online learning algorithm that handles constraints. Since its original formulation is for contextual bandit problems, we customize it for our problem. It also works in 2 stages. In the first one, actions are sampled randomly in each slot to establish sufficient measurements for *(i)* their rewards, and *(ii)* a variable $r_z^e(t) \in \{0, 1\}$ that shows if an action z has violated any of the constraints at time slot t. After the first stage, the expected reward and constraint violation variables for each action are sampled from a normal distribution generated by the measurements of rewards and constraint violations. The next action z_{t+1} is based on balancing those 2 sampled values as follows:

$$z_{t+1} = \arg\max_{z \in \mathcal{Z}} \alpha \tilde{\mu}_z(t) + (1 - \alpha)\tilde{\mu}_z^e(t),$$



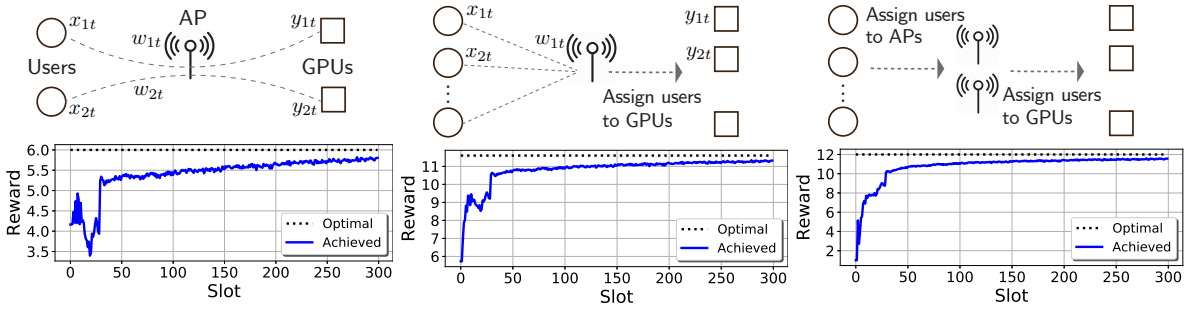
(a) (b)

Figure 8.11: (a) Average iteration delay of Algorithm 7. (b) Maximum iteration delay and convergence time in slots.

where α is the balancing parameter between the expected reward of action z, $\tilde{\mu}_z(t)$, and the respective expected constrained violation ($\tilde{\mu}^e_z(t)$ is higher for actions that satisfy the constraints). Since this approach does not guarantee the satisfaction of constraints with high probability, we have selected $\alpha = 0.1$ to fairly compare with our solution, i.e. minimize the frequency of selecting constraint violating actions.

Fig. 8.10b depicts the average regret for Algorithm 7 and its competitors for 100 simulation runs. Algorithm 7 makes high reward actions after the expansion stage, resulting in a continuous decrease of the regret. We impose the stopping criterion discussed before, and observe that convergence occurs at about 200. NB initially shows very high regret, especially because of the penalization of constraint violating actions. It recovers very quickly but still it has weaker performance than Algorithm 7. BCTS on the other hand shows very low regret, even in the first stage (30 slots), as constraint violations are not penalized and all actions are equally explored. In addition, convergence occurs very quickly, but it is to an action that is further away from the optimal compared to Algorithm 7, as the regret decreases in a much slower pace. The inability of NB and BCTS to safely explore the action space is highlighted in Fig. 8.10c where we see the percentage of slots where either of the user constraints were violated. Observe that for user 2 that has a stricter frame rate requirement, violations are up about 20% for both NB and BCTS compared to Algorithm 1. In conclusion, Algorithm 7 converges to a better solution than NB and BCTS, and guarantees much safer exploration of the action space.

Next, we evaluate the algorithm's scalability by measuring its average iteration delay, and in particular, the time required to execute steps 7-19 in our server. Fig. 8.11a depicts this delay as



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

Figure 8.12: Reward of (a): Many GPUs and preassigned users; (b): User-to-GPU assignment; (c): User-to-AP-to-GPU assignment.

a fraction of slot duration Δ for different number of users N. For the first 30 slots (expansion stage) we clearly see the delay increasing both with the slot t and users N. The former is because the updates (8.6)-(8.9) increase in complexity with the samples, since they involve matrix inversions of size t. The latter is because with more users, there are many more candidate configurations for the safe set expansion. After the first stage, we observe a drop of the delay since *(i)* the posteriors of the constraint functions no longer require updates, and *(ii)* the safe set has been fixed and $u_t^f(z)$ is only evaluated for $z \in S_t$. The iteration delay starts increasing again with t for the same reason as before, but is kept low until the algorithm converges to an acceptable solution. Interestingly, the delay for $N = 4$ is bigger than with $N = 5$, which is due to the smaller $|S_t|$ we get for the latter case, since fewer actions are feasible in that case. This typically results in smaller fractions of the action set ending up in the safe set, and as explained earlier, this is a main factor that affects the delay in the second stage.

We consider more users in Fig. 8.11b where we set a low frame rate requirement $\lambda_n = 2,\ \forall n \in \mathcal{N}$, so that problem $\mathbb{P}$ is feasible. In the top graph we show the maximum value of the iteration delay within a 200 slots evaluation. Notice that for $N \geq 12$ the delay gets much bigger than the slot duration, which suggests that we have to either increase Δ and admit longer convergence, or reduce the expansion stage duration. Alternatively, we can delegate parts of the GP computations to the orchestrator for deciding the next slot configuration on time. The lower graph in Fig. 8.11b shows the slot in which (on average) the stopping criterion discussed in Sec. 8.5 occurs for different values of N. We observe that the differences are insignificant and that we can always stop the algorithm in fewer than 200 slots.

Finally, we evaluate our framework for the settings where *(i)* multiple GPUs ($K = 2$) are

available to the server, and a NN size configuration y_{nt} is selected for each user n (Scenario 1); *(ii)* the number of users N is higher than the number of GPUs K (Scenario 2); and *(iii)* the users can be served by a number of J APs. In detail we set $N = 4, K = 2, J = 1$ for Scenario 2, and $N = 4, K = 2, J = 2$ for Scenario 3. The achieved and optimal reward of Algorithm 7 for Scenarios 1-3 is displayed in Fig. 8.12a-8.12c, along with a small graph depicting the differences in the setup and of each scenario. Remember that the achieved reward is simply the added observed CC for all users in each slot. We can see that in all scenarios the performance of the system keeps increasing and converging towards the optimal one. In specific, the observed performance is within only $6\%, 4\%$ and 5% from the optimal in each Scenario, after 200 slots.

Chapter 9

Conclusions

This thesis is the product of scientific effort to understand the operation and performance trade-offs of data analytic services deployed over edge computing networks. We studied IoT and mobile networks trying to answer questions like: Where should the service be deployed, or how should the parameters of the service be configured to offer optimal performance, while also complying to user and system constraints. The complexity of those problems lead to the adoption of various mathematical and programming tools in order to provide efficient and robust algorithms for their solution.

9.1 Summary and Findings

Starting with IoT networks capable of executing analytics, we studied how collaborative execution can increase performance. We formulated optimization problems towards minimizing (maximizing) the execution delay (accuracy) of tasks under various networking and power consumption constraints. The initial approach is a distributed algorithm that converges fast, even for large networks and increases the performance compared to the local-only task execution. A trace-driven evaluation showed that indeed there is need for such cooperative solutions, and that the designer needs to carefully tune the priority parameters, especially for networks where delay and accuracy are conflicting. Moving a step further, we designed a comprehensive optimization framework for cooperative task execution that does *not* assume willingness for cooperation, and provides the necessary incentives through a double-auction mechanism that finds the optimal task execution policy with minimal information requirements. This is a clean-slate algorithm of

theoretical interest in its own right, and it is backed by a real-world implementation. Namely, we fully implement and evaluate the performance of our proposal using a face recognition application and a RPi testbed. The results show that the algorithm outperforms various benchmark policies and increases the supportable rate of tasks.

Next, we assume more realistic assumptions for collaborative computing of analytics tasks in IoT networks. In Chapter 5 we combined the ADSM and FW algorithms to provide a robust scheduling policy that optimizes the system's task execution accuracy and keeps congestion in low levels compared to other standard approaches. Our algorithm makes fast schedule evaluations, since it solves a Linear Program in each iteration that is also a feasible schedule. Moreover, it is a solution tailored to analytic services where the parameters like traffic load, network capacities and even task execution accuracy vary over time in a possibly non-i.i.d way. We provide theoretical convergence bounds for the proposed solution and finally demonstrate the high performance level of our algorithm, comparing it to standard scheduling methods. We then demonstrate online scheduling for an edge-assisted network, where nodes can outsource certain tasks to a single more powerful cloudlet device, depending on resource availability. The key feature of our proposal is a dynamic and distributed algorithm that makes the outsourcing decisions based on the expected performance improvement, and the available resources at the devices and cloudlet. It was shown, theoretically and through experiments, that this *joint performance-costs* design outperforms other efforts that do not cater for the analytics accuracy or the resource availability. The proposed algorithm achieves near-optimal performance in a deterministic fashion, and under minimal assumptions about the system behavior. Namely, it suffices the perturbations to be bounded in each slot and have well-defined means. This makes it ideal for the problem at hand where, more often than not, the stochastic effects (e.g., expected accuracy gains) do not follow an i.i.d. or a Markov process, as required by other optimization approaches.

Finally, we built and studied a MEC-assisted object recognition system to highlight its unique performance trade-offs, and show that careful network transmit and powersave strategies can significantly reduce the wireless transmission delay. We find that the level of image compression, as well as the dimension of the deep learning network used, are key design parameters, affecting both end-to-end latency and object recognition accuracy. We demonstrate how our measurements can be used to choose these design parameters to optimally trade-off between execution delay

and accuracy. We then used our exemplar object recognition prototype, and demonstrated that MEC-assisted video analytics exhibit volatile and platform/data-dependent performance. This renders traditional optimization approaches inadequate for their control, especially for the challenging real-time high-accuracy analytics. To address this issue, we built a Bayesian online learning framework for configuring important service and networking parameters towards high accuracy object recognition with frame rate guarantees for multiple users. Putting theory into practice, we performed a thorough evaluation using our prototype, and verified the efficacy of the framework in a variety of scenarios, but also identified potential bottlenecks, such as increasing computational complexity, and proposed remedies for overcoming them. Our approach is inspired by ideas in the area of automated machine learning, and we believe that our work paves the way for building fully adaptable systems with performance guarantees, being also oblivious on the hardware and software setup of the system.

In summary, this work has identified the unique nature of edge analytics, and the intricate performance trade-offs that arise. Depending on the strictness of assumptions and the system setup of each scenario studied, we proposed optimization techniques tailored to their requirements and highlighted their advantages compared to other standard state-of-the-art approaches.

9.2 Future Work

The plethora of system configuration options that demonstrate varying levels of accuracy, delay and resource consumption complicate their efficient tuning. This problem becomes even more cumbersome as the increasing network density creates large-scale optimization problems that are difficult to track in reasonable amount of time, as discussed in Chapter 8. Moreover, many of the system parameters like, users' data rate, task size etc, and thus optimal solutions, constantly vary with time.

A potential solution to the scalability issues of these approaches would be the utilization of deep learning methods to making predictions about the system's parameter settings, like e.g., edge offloading and model selection variables [191]. In other words using e.g. a DNN for predicting the optimal configuration of a machine learning or data analytic service, based on the current system state, can lead to substantial decrease in computing delay of these solutions, and enable scalable deployment of the related services. This will also allow dynamic adaptation to

the system's varying parameters that sometimes render optimization techniques non-applicable in high user density systems.

There is evidence in the literature that support such ideas. Learning based approaches like [92] attempt to solve offloading problems. Deep learning solutions have also been studied for several resource allocation problems in edge systems. The works in [192–194] use a deep reinforcement learning technique (Deep Q-Networks) to tackle offloading problems. In [195], a double deep Q-Learning model is proposed to improve device energy consumption by learning the performance of the available DVFS algorithms. Such approaches are ideal for environments with feedback but they do not capture the distinct performance metrics of data analytics. More similar to our approach, the authors in [196] train a DNN to the inputs/outputs of a popular interference management algorithm in order to reduce its computational complexity and obtain prediction accuracy of $> 90\%$. The authors in [197] highlight the importance of using deep learning techniques for solving challenging integer optimization problems, and propose an algorithm for learning the branch-and-bound pruning policy, often used in such problems. Finally, the work in [198] uses a swarm optimization algorithm to obtain solutions for a Mixed Integer Non Linear Program (MINLP) that optimizes energy consumption, and uses the samples to train a DNN for making user association and resource allocation decisions. All the above works indicate the importance of developing deep learning solutions since they can provide an efficient alternative to solving complex optimization problems; serving as an interesting direction for future work.

Chapter 10

Appendices

10.1 Appendix to Chapter 5

As f is M-smooth, it is $f(y) \leq f(x) + \nabla_x f(x)^\top (y-x) + \frac{M}{2}\|y-x\|^2$. Let $y = x_{t+1} = (1-\beta)x_t + \beta s_t$, $x = x_t$, to obtain:

$$f(x_{t+1}) \leq f(x_t) + \beta \nabla_x f(x_t)^\top (s_t - x_t) + \beta^2 \frac{MR^2}{2}. \tag{10.1}$$

Proof of Proposition 2. The proposition will be true if any schedule that satisfies the constraints can be written as a convex combination of schedules in S. Consider schedule s for which (5.2) - (5.4) hold $\forall (i,j)$. Further, assume that for link (i,j) we have $\sum_c y_{ij}^{(c)} > 0$, and $\sum_c y_{kl}^{(c)} > 0, \forall (k,l) \in I(i,j)$, so $s \notin S$. Consider the following schedules that belong in S:

- s_{ij}: Same as s but it holds that $\sum_c y_{ij}^{(c)} = \mu_{ij}$, and $\sum_c y_{kl}^{(c)} = 0, \forall (k,l) \in I(i,j)$.
- s_{kl}: Same as s but $\sum_c y_{ij}^{(c)} = 0$, $\sum_c y_{kl}^{(c)} = \mu_{kl},\ (k,l) \in I(i,j)$, and $\sum_c y_{mn}^{(c)} = 0,\ (m,n) \in I(i,j) - \{(k,l)\}$.

The above schedules mean that either (i,j) or one of its interfering links (k,l) is active. W.l.o.g. we assume that s satisfies (5.4) strictly, i.e $\xi_{ij} + \sum_{(k,l)\in I(i,j)} \xi_{kl} = 1$, where $\xi_{ij} = \sum_c y_{ij}^{(c)}/\mu_{ij} \in [0,1]$, and $\xi_{kl} = \sum_c y_{kl}^{(c)}/\mu_{kl} \in [0,1],\ (k,l) \in I(i,j)$. Observe that we can express s as a convex combination of schedules that belong to S, i.e.,

$$s := \xi_{ij} s_{ij} + \sum_{(k,l)\in I(i,j)} \xi_{kl} s_{kl}.$$

This holds also when s has many active interfering links that respect (5.4), since we can express them as convex combinations of extreme point schedules from S. ■

The next two lemmas are used in the proof of Lemma 2.

Lemma 4 (Bounded level set) *Let the Slater condition hold, i.e. there exists a vector x_s such that $g_t(x_s) < 0, \forall t$. The superlevel set $\mathcal{Q}_{\hat{\lambda}} = \{\lambda \succeq 0 \mid h_t(\lambda) \geq h_t(\hat{\lambda})\}$ is bounded. That is, for any $\lambda \in \mathcal{Q}_{\hat{\lambda}}$ we have*

$$\|\lambda\| \leq (f_t(x_s) - h_t(\hat{\lambda}))/q_t, \text{where } q_t = \min_i\{-g_{t,(i)}(x_s)\}.$$

Proof It holds that $\forall \lambda \in \mathcal{Q}_{\hat{\lambda}}$ it is $h_t(\hat{\lambda}) \leq h_t(\lambda) = \min_x\{f_t(x) + \lambda^\top g_t(x)\} \leq f_t(x_s) + \lambda^\top g_t(x_s) = f_t(x_s) + \sum_{i=1}^m \lambda(i) g_{t,(i)}(x_s)$. Since $g_{t,(i)}(x_s) < 0$, and $\lambda(i) \geq 0$:

$$\min_i\{-g_{t,(i)}(x_s)\} \sum_{i=1}^m \lambda(i) \leq f_t(x_s) - h_t(\hat{\lambda}) \Rightarrow \|\lambda\| \leq \frac{1}{q_t}(f_t(x_s) - h_t(\hat{\lambda})),$$

which completes the proof.

Lemma 5 (Bounded Multipliers) *The multiplier sequence generated by Algorithm 4 is bounded, i.e.,*

$$\|\lambda_t\| \leq \Lambda_t \triangleq 2\frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \max\Big\{\|\lambda_1\|, \frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \frac{\gamma_t}{q_t} + \frac{\alpha\sigma_g^2}{2q_t} + \alpha\sigma_g\Big\}.$$

Proof We define the superlevel set $\mathcal{Q}_a = \{\lambda \succeq 0 \mid h_t(\lambda) \geq h_t(\lambda^\circ) - \gamma_t - \frac{\alpha\sigma_g^2}{2}\}$ and prove that

$$\|\lambda_t - \lambda^\circ\| \leq \max\Big\{\|\lambda_1 - \lambda^\circ\|, \frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \frac{\gamma_t}{q_t} + \frac{\alpha\sigma_g^2}{2q_t} + \alpha\sigma_g + \|\lambda^\circ\|\Big\}. \tag{10.2}$$

Observe that the above holds for $t = 1$. We are going to assume it holds for any t, and prove it holds for $t+1$. **Case** *(i)*: $h_t(\lambda_t) \geq h_t(\lambda^\circ) - \gamma_t - \frac{\alpha\sigma_g^2}{2}$. By the dual update rule:

$$\begin{aligned}\|\lambda_{t+1} - \lambda^\circ\| = \|[\lambda_t + \alpha g_t(x_{t+1})]^+ - \lambda^\circ\| &\leq \|\lambda_t\| + \|\lambda^\circ\| + \alpha\sigma_g \\ &\leq \frac{1}{q_t}(f_t(x_s) - h_t(\lambda^\circ) + \gamma_t + \frac{\alpha\sigma_g^2}{2}) + \|\lambda^\circ\| + \alpha\sigma_g\end{aligned}$$

where the last inequality follows from Lemma 4 for $\mathcal{Q}_a$, hence (10.2) holds. **Case** *(ii)*: $h_t(\lambda_t) < h_t(\lambda^\circ) - \gamma_t - \frac{\alpha\sigma_g^2}{2}$. We have:

$$\begin{aligned}\|\lambda_{t+1} - \lambda^\circ\|^2 \leq \|\lambda_t + \alpha g_t(x_{t+1}) - \lambda^\circ\|^2 \leq \|\lambda_t - \lambda^\circ\|^2 + 2\alpha g_t(x_{t+1})^\top(\lambda_t - \lambda^\circ) + \alpha^2\sigma_g^2 \\ \overset{(a)}{\leq} \|\lambda_t - \lambda^\circ\|^2 + 2\alpha(h_t(\lambda_t) - h_t(\lambda^\circ) + \gamma_t) + \alpha^2\sigma_g^2 \leq \|\lambda_t - \lambda^\circ\|^2,\end{aligned}$$

where (a) holds since

$$\begin{aligned}g_t(x_{t+1})^\top(\lambda_t - \lambda^\circ) = f_t(x_{t+1}) - f_t(x_{t+1}) + g_t(x_{t+1})^\top(\lambda_t - \lambda^\circ) \leq L_t(x_{t+1}, \lambda_t) - L_t(x_{t+1}, \lambda^\circ) \\ \leq h_t(\lambda_t) + \gamma_t - h_t(\lambda^\circ),\end{aligned}$$

which holds by the assumption of this case, making (10.2) true. Now we can rewrite (10.2) as:

$$\begin{aligned}\|\lambda_t - \lambda^\circ\| \leq \max\Big\{\|\lambda_1 - \lambda^\circ\|, \frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \frac{\gamma_t}{q_t} + \frac{\alpha\sigma_g^2}{2q_t} + \alpha\sigma_g + \|\lambda^\circ\|\Big\} \\ \leq \|\lambda^\circ\| + \max\Big\{\|\lambda_1\|, \frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \frac{\gamma_t}{q_t} + \frac{\alpha\sigma_g^2}{2q_t} + \alpha\sigma_g\Big\}.\end{aligned}$$

Since $\|\lambda_t\| \leq \|\lambda_t - \lambda^\circ\| + \|\lambda^\circ\|$ we obtain $\|\lambda_t\| \leq 2\|\lambda^\circ\| + \max\big\{\|\lambda_1\|, \frac{f_t(x_s) - h_t(\lambda^\circ)}{q_t} + \frac{\gamma_t}{q_t} + \frac{\alpha\sigma_g^2}{2q_t} + \alpha\sigma_g\big\}$, and by applying Lemma 4 for $\hat{\lambda} = \lambda^\circ$ we obtain the claimed bound.

Proof of Lemma 2. We first prove the upper bound on the objective function by linking the static and t-slot problems. We define $z_t \in \arg\min_{x\in X} f(x) + \lambda_t^\top g(x)$ (which is bounded as X is bounded) and also $y_t \in \arg\min_{x\in X} f_t(x) + \lambda_t^\top g_t(x)$. Now we can write:

$$h_t(\lambda_t) = f(y_t) + \lambda_t^\top g(y_t) + \epsilon_t^\top y_t + \lambda_t^\top \phi_t \leq f(z_t) + \lambda_t^\top g(z_t) + \epsilon_t^\top z_t + \lambda_t^\top \phi_t = h(\lambda_t) + \epsilon_t^\top z_t + \lambda_t^\top \phi_t.$$

Hence:

$$\begin{aligned}f(x^\circ) = h(\lambda^\circ) \geq \frac{1}{t}\sum_{\tau=1}^{t} h(\lambda_\tau) \geq \frac{1}{t}\sum_{\tau=1}^{t} h_\tau(\lambda_\tau) - \epsilon_\tau^\top z_\tau - \lambda_\tau^\top \phi_\tau \\ \overset{(b)}{\geq} \frac{1}{t}\sum_{\tau=1}^{t} L_\tau(x_\tau, \lambda_\tau) - \gamma_\tau - \epsilon_\tau^\top z_\tau - \lambda_\tau^\top \phi_\tau \geq \frac{1}{t}\sum_{\tau=1}^{t} f_\tau(x_\tau) + \lambda_\tau^\top g_\tau(x_\tau) - \gamma_\tau - \epsilon_\tau^\top z_\tau - \lambda_\tau^\top \phi_\tau.\end{aligned}$$

where (b) holds from the approximate minimization of the Lagrangian. Rearranging terms yields:

$$\frac{1}{t}\sum_{\tau=1}^{t} f_\tau(x_\tau) - f(x^\circ) \leq -\frac{1}{t}\sum_{\tau=1}^{t} \lambda_\tau^\top g_\tau(x_\tau) - \gamma_\tau - \epsilon_\tau^\top z_\tau - \lambda_\tau^\top \phi_\tau. \tag{10.3}$$

We proceed to upper bound the first term in the RHS of the last equation. Observe that for any $\theta \in \mathbf{R}_+^m$ we have $\|\lambda_{t+1} - \theta\|^2 = \|[\lambda_t + \alpha g_t(x_t)]^+ - \theta\|^2 \leq \|\lambda_t + \alpha g_t(x_t) - \theta\|^2 = \|\lambda_t - \theta\|^2 + \alpha^2\|g_t(x_t)\|^2 + 2\alpha(\lambda_t - \theta)^\top g_t(x_t)$. Rearranging yields:

$$\|\lambda_{t+1} - \theta\|^2 - \|\lambda_t - \theta\|^2 \leq \alpha^2\|g_t(x_t)\|^2 + 2\alpha(\lambda_t - \theta)^\top g_t(x_t).$$

Setting $\theta = 0$, and applying the telescopic summation:

$$-2\alpha\sum_{\tau=1}^{t} \lambda_\tau^\top g_\tau(x_\tau) \leq \alpha^2 \sum_{\tau=1}^{t} \|g_\tau(x_\tau)\|^2,$$

where $-\|\lambda_{t+1}\|^2$ and $\|\lambda_1\|^2$ were dropped in the above since the former is non-positive, and the latter can be zeroed out by setting $\lambda_1 = 0$. Using the fact that $\|g_\tau(x)\| \leq \sigma_g, \forall \tau$, for all $x \in X_\tau$, and diving across by $2\alpha t$ we obtain $-\frac{1}{t}\sum_{\tau=1}^{t} \lambda_\tau^\top g_\tau(x_\tau) \leq \frac{\alpha\sigma_g^2}{2}$. By using the above bound on (10.3) we prove the first claim.

From the dual update rule recursive expansion we have $\lambda_{t+1} \succeq \lambda_1 + \alpha\sum_{\tau=1}^{t} g_\tau(x_\tau)$. By dropping λ_1, dividing by αt, and taking the norms we obtain:

$$\|\frac{1}{t}\sum_{\tau=1}^{t} g_\tau(x_\tau)\| \leq \frac{\|\lambda_{t+1}\|}{\alpha t}, \tag{10.4}$$

which yields the claimed bound after applying Lemma 5.

We need the following Lemma for proving Theorem 1. Note also, that in order to differentiate between Algorithms 4 and 5 we use v_t for the primal variable of the latter.

Lemma 6 (FW for the Lagrangian) *Set $\eta > 0$ and select $\beta \in (0, 1]$ and $0 < \alpha < \frac{\beta\eta - (\beta^2 M_L R^2/2) - \sigma_f - \sigma_L - 2\Lambda\sigma_g}{\sigma_g^2}$, M_L is a constant independent of α, β. Algorithm 5 guarantees that if $L_t(v_t, \lambda_t) - h_t(\lambda_t) \geq \eta$:*

$$L_{t+1}(v_{t+1}, \lambda_{t+1}) < L_{t+1}(v_t, \lambda_{t+1}), \quad \textit{or}$$
$$L_{t+1}(v_{t+1}, \lambda_{t+1}) - h_{t+1}(\lambda_{t+1}) \leq 2\eta, \ \textit{otherwise.}$$

Proof Based on Lemma 5, the sequence of vectors $\{\lambda_t\}_t$ is bounded. Hence, $L_t(\cdot, \lambda_t)$ is an M_L-smooth convex function of v for any parameter λ. We consider two cases. **Case** *(i)*: $L_t(v_t, \lambda_t) - h_t(\lambda_t) > \eta$, where recall that $h_t(\lambda_t) \leq L_t(v, \lambda_t), \forall x$. Due to its smoothness and the FW rule applied on (10.1), we have

$$L_t(v_{t+1}, \lambda_t) - L_t(v_t, \lambda_t) \leq \beta \nabla_v L_t(v_t, \lambda_t)^\top (s_t - v_t) + \frac{\beta^2 M_L R^2}{2}, \tag{10.5}$$

and since s_t is the minimizer of $u^\top \nabla_v L_t(v_t, \lambda_t)$ we can replace the second term in the RHS with $\beta \nabla_v L_t(v_t, \lambda_t)^\top (v_t^* - v_t)$, where $v_t^* = \arg\min_v L_t(v, \lambda_t)$, i.e., $h_t(\lambda_t) = L_t(v_t^*, \lambda_t)$. Due to convexity of $L_t(\cdot, \lambda_t)$, it is $\nabla_v L_t(v_t, \lambda_t)^\top (v_t^* - v_t) \leq h_t(\lambda_t) - L_t(v_t, \lambda_t) < -\eta$, hence

$$L_t(v_{t+1}, \lambda_t) - L_t(v_t, \lambda_t) \leq -\beta\eta + \beta^2 \frac{M_L R^2}{2}. \tag{10.6}$$

Now, it holds that

$$\begin{aligned} L_{t+1}(v_{t+1}, \lambda_{t+1}) - L_t(v_{t+1}, \lambda_t) &= f_{t+1}(v_{t+1}) - f_t(v_{t+1}) + \lambda_{t+1}^\top g_{t+1}(v_{t+1}) - \lambda_t^\top g_t(v_{t+1}) \\ &= f_{t+1}(v_{t+1}) - f_t(v_{t+1}) + \lambda_t^\top (g_{t+1}(v_{t+1}) - g_t(v_{t+1})) \\ &+ \alpha g_t(v_t)^\top g_{t+1}(v_{t+1}) \leq \sigma_f + 2\|\lambda_t\|\sigma_g + \alpha\sigma_g^2, \end{aligned} \tag{10.7}$$

where we used Cauchy-Schwarz and triangle inequalities. Adding (10.6) and (10.7), we get:

$$L_{t+1}(v_{t+1}, \lambda_{t+1}) \leq L_t(v_t, \lambda_t) - \beta\eta + \beta^2 \frac{M_L R^2}{2} + \sigma_f + 2\Lambda\sigma_g + \alpha\sigma_g^2,$$

where we have defined $\Lambda = \max_t \|\lambda_t\|$. Subtracting $L_{t+1}(v_t, \lambda_{t+1})$in both sides, using $|L_{t+1}(v_t, \lambda_{t+1}) - L_t(v_t, \lambda_t)| \leq \sigma_L$, and by the stated choices of α, β we obtain that $L_{t+1}(v_{t+1}, \lambda_{t+1}) < L_{t+1}(v_t, \lambda_{t+1})$.

Case *(ii)*: $L_t(v_t, \lambda_t) - h_t(\lambda_t) \leq \eta$. Since s_t is the minimizer of $u^\top \nabla_v L_t(v_t, \lambda_t)$, the term $\nabla_v L_t(v_t, \lambda_t)^\top (s_t - v_t)$ is non-positive and hence can be dropped from (10.5):

$$L_t(v_{t+1}, \lambda_t) - L_t(v_t, \lambda_t) \leq (\beta^2 M_L R^2)/2. \tag{10.8}$$

Adding (10.7) to (10.8) and using $L_t(v_t, \lambda_t) - h_t(\lambda_t) \leq \eta$ we end up with

$$\begin{aligned} L_{t+1}(v_{t+1}, \lambda_{t+1}) - h_t(\lambda_t) &\leq L_t(v_t, \lambda_t) - h_t(\lambda_t) + \frac{\beta^2 M_L R^2}{2} + \sigma_f + 2\Lambda\sigma_g + \alpha\sigma_g^2 \\ &\leq \eta + \frac{\beta^2 M_L R^2}{2} + \sigma_f + 2\Lambda\sigma_g + \alpha\sigma_g^2, \end{aligned}$$

where we used the assumption of this case for the last inequality. Subtracting $h_{t+1}(\lambda_{t+1})$ in both sides and rearranging terms yields

$$L_{t+1}(v_{t+1}, \lambda_{t+1}) - h_{t+1}(\lambda_{t+1}) \leq \eta + \frac{\beta^2 M_L R^2}{2} + \sigma_f + \sigma_L + 2\Lambda\sigma_g + \alpha\sigma_g^2 \leq \eta + \beta\eta + \leq 2\eta,$$

where we used $|h_{t+1}(\lambda_{t+1}) - h_t(\lambda_t)| \leq \sigma_L$, and the stated α, β, completing the second case.

Proof of Theorem 1. By the stated selection of α and β, Lemma 6 applies and yields

$$L_{t+1}(v_{t+1}, \lambda_{t+1}) - h_{t+1}(\lambda_{t+1}) \leq \max\{\zeta_t, 2\eta\},$$

where $\zeta_t = L_{t+1}(v_t, \lambda_{t+1}) - h_{t+1}(\lambda_{t+1})$. By applying Lemma 2 with $\gamma_t = \max\{\zeta_t, 2\eta\}$ we obtain

$$1/t \sum_{\tau=1}^{t} f_\tau(v_\tau) - f(x^\circ) \leq \frac{\alpha\sigma_g^2}{2} + \frac{1}{t}\sum_{\tau=1}^{t} \max\{\zeta_\tau, 2\eta\} + \epsilon_\tau^\top \hat{x}_\tau + \lambda_\tau^\top \phi_\tau \tag{10.9}$$

We proceed by bounding the average performance of the selected schedules s_t. By convexity of $L_t(\cdot, \lambda_t)$, we have

$$L_t(v_t, \lambda_t) \geq L_t(s_t, \lambda_t) + \nabla L_t^\top(s_t, \lambda_t)(v_t - s_t) \geq L_t(s_t, \lambda_t).$$

So it is $f_t(s_t) \leq f_t(x_t) + \lambda_t^\top (g_t(x_t) - g_t(s_t))$ and by summing telescopically we have

$$\frac{1}{t}\sum_{\tau=1}^{t} f_\tau(s_\tau) \leq \frac{1}{t}\sum_{\tau=1}^{t} f_\tau(x_\tau) + 2\Lambda\sigma_g.$$

Combining the above with (10.9) we obtain the claimed bound (i). For the feasibility gap, we use the linearity of g_t to obtain $g_t(s_t) - g_t(x_t) = (1/\beta)g_t(x_{t+1} - x_t)$ which yields bound (ii) after summing over all t and dividing by t.

10.2 Appendix to Chapter 6

This section contains the proof of Theorem 1 in Chapter 6. We drop bold typeface notation here; use subscript $i = 1, \ldots, t$ to denote the ith slot; and n for the nth component of a vector. We will be using the following Lagrangians:

$$L(y_t, \mu_t, \delta_t, \epsilon_t) = f(y_t) + y_t^\top \epsilon_t + \mu_t^\top \left(g(y_t) + \delta_t(y_t)\right) \tag{10.10}$$

$$L(y_t, \mu_t) = f(y_t) + \mu_t^\top g(y_t), \tag{10.11}$$

and the respective dual functions $V(\mu, \delta_t, \epsilon_t)$ and $V(\mu)$. Eq. (10.10) is the Lagrangian used in the subgradient method; and unless stated otherwise, we will use below $y_t \in \arg\min_{y \in \mathcal{Y}} L(y, \mu_t, \delta_t, \epsilon_t)$. We first bound the distance of μ_{t+1} from vector $\theta \in \mathbb{R}^{N+1}$, i.e., $\|\mu_{t+1} - \theta\|^2 =$

$$\begin{aligned}
&\left\|[\mu_t + a(g(y_t) + \delta_t(y_t))]^+ - \theta\right\|^2 \leq \|\mu_t + a(g(y_t) + \delta_t(y_t)) - \theta\|^2 = \\
&\|\mu_t - \theta\|^2 + a^2 \|g(y_t) + \delta_t(y_t)\|^2 + 2a(\mu_t - \theta)^\top (g(y_t) + \delta_t(y_t)) \leq \\
&\|\mu_t - \theta\|^2 + a^2 \|g(y_t)\|^2 + a^2 \|\delta_t(y_t)\|^2 + 2a^2 \delta_t(y_t)^\top g(y_t) + 2a(\mu_t - \theta)^\top (g(y_t) + \delta_t(y_t))
\end{aligned} \tag{10.12}$$

Next, we bound the difference of $L(y_t, \mu_t, \delta_t, \epsilon_t)$ from $V(\mu_t)$. We define $\hat{y}_t \in \arg\min_{y \in \mathcal{Y}} L(y, \mu_t)$, which is different from y_t. Then we can write:

$$L(y_t, \mu_t, \delta_t, \epsilon_t) = L(\hat{y}_t, \mu_t) + L(y_t, \mu_t, \delta_t, \epsilon_t) - L(\hat{y}_t, \mu_t) \leq V(\mu_t) + A_t(\hat{y}_t, y_t) \tag{10.13}$$

where we defined $A_t(\hat{y}_t, y_t) = f(y_t) - f(\hat{y}_t) + y_t^\top \epsilon_t + \mu_t^\top (g(y_t) - g(\hat{y}_t) + \delta_t(y_t))$, and used $L(\hat{y}_t, \mu_t) = V(\mu_t)$. By Assumption 1, it holds that $\lim_{t\to\infty} L(\cdot, \mu_t, \delta_t, \epsilon_t) = L(\cdot, \mu_t)$, and given that these are continuous convex functions, this yields $\lim_{t\to\infty} A_t(\hat{y}_t, y_t) = 0$. Also, we can upper bound $A_t(\cdot)$ for every t, since the objective and constraint functions are upper bounded (all their components), and also the dual vector is bounded for any t (as we will prove in the sequel). Therefore, it also holds that $\lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} A_i(\hat{y}_i, y_i) = 0$.

(i) Optimality Gap. Using (10.13) we can write:

$$\begin{aligned} V(\mu^*) &\geq \frac{1}{t}\sum_{i=1}^{t} V(\mu_i) \geq \frac{1}{t}\sum_{i=1}^{t} L(y_i, \mu_i, \delta_i, \epsilon_i) - A_i(\hat{y}_i, y_i) \\ &= \frac{1}{t}\sum_{i=1}^{t} \Big(f(y_i) + y_i^\top \epsilon_i + \mu_i^\top \big(g(y_i) + \delta_i(y_i)\big) - A_i(\hat{y}_i, y_i) \Big) \\ &= \frac{1}{t}\sum_{i=1}^{t} f(y_i) + \frac{1}{t}\sum_{i=1}^{t} \Big(y_i^\top \epsilon_i + \mu_i^\top \big(g(y_i) + \delta_i(y_i)\big) - A_i(\hat{y}_i, y_i) \Big) \end{aligned} \tag{10.14}$$

Now, let $\theta = 0$ in (10.12), we get:

$$\|\mu_{t+1}\|^2 \leq \|\mu_t\|^2 + a^2 \|g(y_t)\|^2 + a^2 \|\delta_t(y_t)\|^2 + 2a^2 \delta_t(y_t)^\top g(y_t) + 2a\mu_t^\top \big(g(y_t) + \delta_t(y_t)\big), \tag{10.15}$$

and using (6.11) and the Cauchy-Swartz inequality:

$$\|\mu_{t+1}\|^2 \leq \|\mu_t\|^2 + a^2\sigma_g^2 + a^2\sigma_{\delta_t}^2 + 2a^2\sigma_g\sigma_{\delta_t} + 2a\mu_t^\top \big(g(y_t) + \delta_t(y_t)\big). \tag{10.16}$$

Applying it for all t and summing, we obtain:

$$\|\mu_{t+1}\|^2 \leq \|\mu_1\|^2 + a^2 t\sigma_g^2 + a^2 \sum_{i=1}^{t} \sigma_{\delta_t}^2 + 2a^2\sigma_g \sum_{i=1}^{t} \sigma_{\delta_t} + 2a \sum_{i=1}^{t} \mu_i^\top \big(g(y_i) + \delta_i(y_i)\big),$$

which, if we drop the non-negative term $\|\mu_{t+1}\|^2$, divide by $2at$, and rearrange terms, yields:

$$-\frac{1}{t}\sum_{i=1}^{t} \mu_i^\top \big(g(y_i) + \delta_i(y_i)\big) \leq \frac{\|\mu_1\|^2}{2at} + \frac{a\sigma_g^2}{2} + \frac{a}{2t}\sum_{i=1}^{t} \sigma_{\delta_i}^2 + \frac{a\sigma_g}{t}\sum_{i=1}^{t} \sigma_{\delta_i}. \tag{10.17}$$

Setting $\mu_1 = 0$, using the fact that $V(\mu^*) = f^*$, and combining (10.17) with (10.14), we obtain:

$$\frac{1}{t}\sum_{i=1}^{t} f(y_i) - f^* \leq \frac{a\sigma_g^2}{2} + \frac{a}{2t}\sum_{i=1}^{t} \sigma_{\delta_i}^2 + \frac{a\sigma_g}{t}\sum_{i=1}^{t} \sigma_{\delta_i} + \frac{1}{t}\sum_{i=1}^{t} y_i^\top \epsilon_i + \frac{1}{t}\sum_{i=1}^{t} A_i(\hat{y}_i, y_i).$$

By Assumption 1, all sums have diminishing terms and divided by t, hence converge to 0. Thus, we obtained the first part of the theorem.

(ii) Constraint Violation. If we apply recursively the following inequality:

$$\mu_{t+1} = \Big[\mu_t + a\big(g(y_t) + \delta_t(y_t)\big)\Big]^+ \succeq \mu_t + a\big(g(y_t) + \delta_t(y_t)\big), \quad \text{we obtain:}$$
$$\mu_{t+1} \succeq \mu_1 + a\sum_{i=1}^{t}\big(g(y_i) + \delta_i(y_i)\big). \tag{10.18}$$

Dropping $\mu_1 = 0$, and dividing by at, we get:

$$\frac{1}{t}\sum_{i=1}^{t} g(y_i) + \frac{1}{t}\sum_{i=1}^{t}\delta_i(y_i) \preceq \frac{\mu_{t+1}}{at}. \tag{10.19}$$

We wish to prove that $\lim_{t\to\infty} 1/t\sum_{i=1}^{t} g(y_i) \preceq 0$. We already know that the second term in the LHS of (10.19) converges to zero, hence it suffices to prove that all components of vector μ_{t+1} are bounded for every t. This will ensure that the RHS converges to 0. We will be using the following assumption.

Assumption 2 *There exists a Slater vector $y^s \in \mathcal{Y}$ such that $g(y^s) + \delta_t(y^s) \prec 0$, which holds component-wise, and for any time slot t.*

This means that there is a Slater vector that satisfies all perturbed instances of (P2). Since the constraints are linear and the perturbations bounded, it is easy to find such a vector (e.g., $y^s = 0$). Under this assumption, and given that f^* is bounded, we know that the set of the dual optimal values μ^* of (D) is bounded, [199]. Now, we define the set:

$$Q(\mu_0, \delta_t, \epsilon_t) = \{\mu \geq 0 \mid V(\mu, \delta_t, \epsilon_t) \geq V(\mu_0, \delta_t, \epsilon_t)\}, \tag{10.20}$$

which is parameterized by some dual vector μ_0, and the running averages of perturbations δ_t, ϵ_t. For every $\mu \in Q(\mu_0, \delta_t, \epsilon_t)$, it holds $V(\mu_0, \delta_t, \epsilon_t) \leq$

$$V(\mu, \delta_t, \epsilon_t) = \inf_{y\in\mathcal{Y}}\Big\{f(y) + y_t^\top\epsilon_t + \mu^\top\big(g(y) + \delta_t(y)\big)\Big\} \leq f(y^s) + \epsilon_t^\top y^s + \sum_{n=1}^{N+1}\mu_n\big(g_n(y^s) + \delta_{nt}(y^s)\big)$$
$$\Rightarrow -\sum_{n=1}^{N+1}\mu_n(g_n(y^s) + \delta_{nt}(y^s)) \leq f(y^s) - V(\mu_0, \delta_t, \epsilon_t) + \epsilon_t^\top y^s. \tag{10.21}$$

It is $g_n(y^s)+\delta_{nt}(y^s) < 0$, $\mu_n \geq 0$, $\forall n = 1, \ldots, N+1$ and $\forall t$, and setting $v = \min_{1\leq n\leq N+1}\{-g_n(y^s)-$

$\delta_{nt}(y^s)\}$, we can write:

$$v\sum_{n=1}^{N+1}\mu_n \leq -\sum_{n=1}^{N+1}\mu_n\big(g_n(y^s)+\delta_{nt}(y^s)\big) \leq f(y^s)-V(\mu_0,\delta_t,\epsilon_t)+\epsilon_t^\top y^s, \text{ hence}$$
$$\sum_{n=1}^{N+1}\mu_n \leq \frac{1}{v}\Big(f(y^s)-V(\mu_0,\delta_t,\epsilon_t)+\epsilon_t^\top y^s\Big), \tag{10.22}$$

and since $\mu \geq 0$ we have that $\|\mu\| \leq \sum_{n=1}^{N+1}\mu_n$. Therefore:

$$\max_{\mu\in Q(\mu_0,\delta_t,\epsilon_t)}\|\mu\| \leq \frac{1}{v}\Big(f(y^s)-V(\mu_0,\delta_t,\epsilon_t)+\epsilon_t^\top y^s\Big). \tag{10.23}$$

Now, we are going to prove that $\|\mu_{t+1}-\mu^*\| \leq$

$$\max\{\|\mu_1-\mu^*\|, \frac{1}{v}\Big(f(y^s)-V(\mu^*,\delta_t,\epsilon_t)-\epsilon_t^\top y^s\Big)+\frac{a(\sigma_g+\sigma_{\delta_t})^2}{2v}+\|\mu^*\|+a(\sigma_g+\sigma_{\delta_t})\} \tag{10.24}$$

This holds for $t=0$, and we use induction to prove it holds for any t. We start by expanding:

$$\|\mu_{t+1}-\theta\|^2 \leq \Big\|\big[\mu_t+a\big(g(y_t)+\delta_t(y_t)\big)\big]^+-\theta\Big\|^2 \leq \big\|\mu_t+a\big(g(y_t)+\delta_t(y_t)\big)-\theta\big\|^2 =$$
$$\|\mu_t-\theta\|^2+a^2\|g(y_t)+\delta_t(y_t)\|^2+2a\big(g(y_t)+\delta_t(y_t)\big)^\top(\mu_t-\theta). \tag{10.25}$$

Note that $g(y_t)+\delta_t(y_t)$ is a subgradient of $V(\mu_t,\delta_t,\epsilon_t)$. Hence, for any θ, we can write:

$$\big(g(y_t)+\delta_t(y_t)\big)^\top(\mu_t-\theta) \leq -\big(V(\theta,\delta_t,\epsilon_t)-V(\mu_t,\delta_t,\epsilon_t)\big). \tag{10.26}$$

Replacing this inequality in (10.25), and setting $\theta=\mu^*$, we have:

$$\|\mu_{t+1}-\mu^*\|^2 \leq \|\mu_t-\mu^*\|^2+a^2\|g(y_t)+\delta_t(y_t)\|^2-2a\big(V(\mu^*,\delta_t,\epsilon_t)-V(\mu_t,\delta_t,\epsilon_t)\big) \tag{10.27}$$

As a next step, we bound the distance:

$$\|\mu_{t+1}-\mu^*\| \leq \|\mu_t+ag(y_t)+a\delta_t(y_t)-\mu^*\| \leq \|\mu_t\|+\|\mu^*\|+a\sigma_g+a\sigma_{\delta_t}, \tag{10.28}$$

where we have used the triangle inequality, and we define the set:

$$Q(\mu^*, \delta_t, \epsilon_t, a) = \left\{ \mu \geq 0 \mid V(\mu, \delta_t, \epsilon_t) \geq V(\mu^*, \delta_t, \epsilon_t) - \frac{a(\sigma_g + \sigma_{\delta_t})^2}{2} \right\}, \tag{10.29}$$

which, by definition, is a superset of $Q(\mu^*, \delta_t, \epsilon_t)$. Now, we consider the following two cases.

First case: μ_t is such that it belongs to the set $Q(\mu^*, \delta_t, \epsilon_t, a)$. Then, by (10.23), we have:

$$\|\mu_t\| \leq \frac{1}{v}(f(y^s) - V(\mu^*, \delta_t, \epsilon_t) + \epsilon_t^\top y^s) + \frac{a(\sigma_g + \sigma_{\delta_t})^2}{2v}, \tag{10.30}$$

which, combined with (10.28), validates (10.24). **Second case**: μ_t is such that

$$V(\mu_t, \delta_t, \epsilon_t) < V(\mu^*, \delta_t, \epsilon_t) - \frac{a(\sigma_g + \sigma_{\delta_t})^2}{2}. \tag{10.31}$$

From (10.27) and (6.11):

$$\|\mu_{t+1} - \mu^*\|^2 \leq \|\mu_t - \mu^*\|^2 - 2a\Big(V(\mu^*, \delta_t, \epsilon_t) - V(\mu_t, \delta_t, \epsilon_t) - \frac{a(\sigma_g + \sigma_{\delta_t})^2}{2}\Big), \tag{10.32}$$

where by our assumption the last term is non-positive and can be dropped, thus $\|\mu_{t+1} - \mu^*\| \leq \|\mu_t - \mu^*\|$. Therefore, again we have shown that (10.24) holds by induction for this case as well.

Using (10.24) and $\|\mu_1 - \mu^*\| \leq \|\mu_1\| + \|\mu^*\|$, we can write $\|\mu_{t+1}\| \leq \|\mu_{t+1} - \mu^*\| + \|\mu^*\| \leq$

$$2\|\mu^*\| + \max\{\|\mu_1\|, \frac{1}{v}\Big(f(y^s) - V(\mu^*, \delta_t, \epsilon_t) - \epsilon_t^\top y^s\Big) + \frac{a(\sigma_g + \sigma_{\delta_t})^2}{2v} + a(\sigma_g + \sigma_{\delta_t})\}, \tag{10.33}$$

which states that all Lagrange multipliers are upper bounded by constants. To see this, note that: the optimal dual variables μ^* are bounded; the dual function is bounded for the bounded μ^* and any combination of perturbations, while we can further set $\mu_1 = 0$ to simplify the RHS. Since all dual variables are positive, and the norm of the dual vector is bounded, it is clear that the RHS of (10.19) converges to zero as $t \to \infty$. Similarly, the second term of the RHS (average value of perturbations) converges to 0, and hence we have that $\lim_{t\to\infty} 1/t \sum_{i=1}^{t} g(y_i) \preceq 0$. This proves that the actual system performance and constraint violation reach the optimal solution asymptotically.

Bibliography

[1] M. R. Palattella *et al.*, "Internet of things in the 5g era: Enablers, architecture, and business models," *IEEE JSAC*, vol. 34, no. 3, pp. 510–527, 2016.

[2] R. Gimenez *et al.*, "The safety transformation in the future internet domain," *The Future Internet*, pp. 190–200, 2012.

[3] O. Vermesan and J. Bacquet, *Cognitive Hyperconnected Digital Transformation: Internet of Things Intelligence Evolution*. Wharton, TX, USA: River Publishers, 2017.

[4] H. Awadalla *et al.*, "Achieving human parity on automatic chinese to english news translation," *CoRR*, vol. abs/1803.05567, 2018.

[5] E. Siow, T. Tiropanis, and W. Hall, "Analytics for the internet of things: A survey," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 74:1–74:36, 2018.

[6] GSMA, "The mobile economy 2019." [Online]: `https://www.gsma.com/r/mobileeconomy/`.

[7] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proc. of MCC*, 2012.

[8] Cisco, "Fog computing and the internet of things: Extend the cloud to where the things are," white paper, 2015.

[9] A. L. Stolyar, "Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm," *Queueing Systems*, vol. 50, pp. 401–457, Aug 2005.

[10] M. J. Neely, "Stability and Capacity Regions or Discrete Time Queueing Networks," *ArXiv e-prints*, Mar. 2010.

[11] M. J. Neely, "A Simple Convergence Time Analysis of Drift-Plus-Penalty for Stochastic Optimization and Convex Programs," *ArXiv e-prints*, Dec. 2014.

[12] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proc. of IEEE INFOCOM*, March 1998.

[13] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile Augmented Reality Survey: From Where We Are to Where We Go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[14] L. Jiang and J. Walrand, "Stable and utility-maximizing scheduling for stochastic processing networks," in *Conference on Communication, Control, and Computing (Allerton)*, 2009.

[15] L. Huang and M. J. Neely, "Utility optimal scheduling in processing networks," *Performance Evaluation*, vol. 68, no. 11, pp. 1002 – 1021, 2011.

[16] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[17] A. Nedic and A. Ozdaglar, "Subgradient methods in network resource allocation: Rate analysis," in *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on*, pp. 1189–1194, March 2008.

[18] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *Proc. of IEEE INFOCOM*, 2018.

[19] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable Adaptation of Video Analytics," in *Proc. of ACM SIGCOMM*, 2018.

[20] S. K. Sharma and X. Wang, "Live data analytics with collaborative edge and cloud processing in wireless iot networks," *IEEE Access*, vol. 5, pp. 4621–4635, 2017.

[21] M. Marjani *et al.*, "Big iot data analytics: Architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[22] P. Patel, M. I. Ali, and A. Sheth, "On using the intelligent edge for iot analytics," *IEEE Intelligent Systems*, vol. 32, pp. 64–69, September 2017.

[23] J. He *et al.*, "Multitier fog computing with large-scale iot data analytics for smart cities," *IEEE Internet of Things Journal*, vol. 5, pp. 677–686, April 2018.

[24] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 1628–1656, thirdquarter 2017.

[25] Y. Mao *et al.*, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 2322–2358, Fourthquarter 2017.

[26] C. K. Tham and R. Chattopadhyay, "A load balancing scheme for sensing and analytics on a mobile edge computing network," in *Proc. of IEEE WoWMoM*, 2017.

[27] C. Wang *et al.*, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, pp. 4924–4938, Aug 2017.

[28] J. Ren *et al.*, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, pp. 5506–5519, Aug 2018.

[29] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, pp. 89–103, June 2015.

[30] C. You *et al.*, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, pp. 1397–1411, March 2017.

[31] N. Cao *et al.*, "Self-optimizing iot wireless video sensor node with in-situ data analytics and context-driven energy-aware real-time adaptation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, pp. 2470–2480, Sept 2017.

[32] A. Padmanabha Iyer *et al.*, "Mitigating the latency-accuracy trade-off in mobile data analytics systems," in *Proc. of ACM MobiCom*, 2018.

[33] Z. Wen *et al.*, "Approxiot: Approximate analytics for edge computing," in *Proc. of IEEE ICDCS*, 2018.

[34] I. Koutsopoulos, "Optimal incentive-driven design of participatory sensing systems," in *Proc. of IEEE INFOCOM*, 2013.

[35] "Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152 – 165, 2017.

[36] X. Gong and N. Shroff, "Incentivizing truthful data quality for quality-aware mobile data crowdsourcing," in *Proc. of ACM Mobihoc*, 2018.

[37] "Google nest cameras." [Online]: `https://nest.com/cameras`.

[38] Microsoft, "Azure iot edge software platform." [Online]: `https://azure.microsoft.com/en-us/services/iot-edge/`.

[39] Amazon, "Aws iot greengrass." [Online]: `https://aws.amazon.com/greengrass/`.

[40] IBM, "Project owl." [Online]: `https://developer.ibm.com/code-and-response/deployments/project-owl/`,.

[41] IBM, "Edgeware fabric." [Online]: `http://edgware-fabric.org/`.

[42] H. M. Raafat, M. S. Hossain, E. Essa, S. Elmougy, A. S. Tolba, G. Muhammad, and A. Ghoneim, "Fog intelligence for real-time iot sensor data analytics," *IEEE Access*, vol. 5, pp. 24062–24069, 2017.

[43] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[44] A. Dou *et al.*, "Misco: A mapreduce framework for mobile systems," in *Proc. of ACM PETRA*, 2010.

[45] M. Y. Arslan *et al.*, "Computing while charging: Building a distributed computing infrastructure using smartphones," in *Proc. of ACM CoNEXT*, 2012.

[46] H. Wang and L.-S. Peh, "Mobistreams: A reliable distributed stream processing system for mobile devices," in *Proc. of IEEE IPDPS*, 2014.

[47] S. Fan, T. Salonidis, and B. Lee, "A framework for collaborative sensing and processing of mobile data streams," in *Proc. of ACM MobiCom*, 2016.

[48] D. O'Keefe, T. Salonidis, and P. Pietzuch, "Network-aware stream query processing in mobile ad-hoc networks," in *Proc. of MILCOM*, 2015.

[49] D. Li *et al.*, "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *Proc. of IEEE/ACM SEC*, 2016.

[50] C. Long *et al.*, "Edge computing framework for cooperative video processing in multimedia iot systems," *IEEE Transactions on Multimedia*, vol. 20, pp. 1126–1139, May 2018.

[51] C. Shi *et al.*, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proc. of ACM MOBIHOC*, 2012.

[52] L. Pu *et al.*, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE JSAC*, vol. 34, no. 12, pp. 3887–3901, 2016.

[53] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive d2d collaboration for energy-efficient mobile edge computing," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 64–71, 2017.

[54] Y. Nan *et al.*, "Adaptive energy-aware computation offloading for cloud of things systems," *IEEE Access*, vol. 5, pp. 23947–23957, 2017.

[55] Z. Sheng *et al.*, "Energy efficient cooperative computing in mobile wireless sensor networks," *IEEE Transactions on Cloud Computing*, vol. 6, pp. 114–126, Jan 2018.

[56] S. Yang *et al.*, "Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing," in *Proc. of IEEE INFOCOM*, 2016.

[57] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. of ACM EuroSys*, 2011.

[58] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proc. of ACM MobiSys*, 2010.

[59] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The impact of mobile multimedia applications on data center consolidation," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2013.

[60] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, Oct. 2009.

[61] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering Deep Learning to Mobile Devices via Offloading," in *Workshop on Virtual Reality and Augmented Reality Network*, 2017.

[62] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proc. ACM VR/AR Network Workshop*.

[63] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2016.

[64] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[65] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. of IEEE ICDCS*, 2017.

[66] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *Proc. of IEEE INFOCOM*, 2019.

[67] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proc. of ACM/IEEE Symposium on Edge Computing (SEC)*, 2017.

[68] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. of ACM MobiSys*, 2016.

[69] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[70] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware Video Analytics on Edge Computing Platform," in *Proc ACM/IEEE SEC*, 2017.

[71] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends on Networking*, pp. 1–144, 2006.

[72] V. Chandrasekhar, J. Lin, Q. Liao, O. Morère, A. Veillard, L. Duan, and T. Poggio, "Compression of deep neural networks for image instance retrieval," in *Data Compression Conference (DCC)*, 2017.

[73] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[74] C. Lo, Y. Su, C. Lee, and S. Chang, "A Dynamic Deep Neural Network Design for Efficient Workload Allocation in Edge Computing," in *Proc. of IEEE ICCD*, 2017.

[75] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," *CoRR*, vol. abs/1701.00299, 2017.

[76] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices," in *Proc. of IEEE ICDCS*, 2017.

[77] M. Ali *et al.*, "Edge enhanced deep learning system for large-scale video stream analytics," in *Proc. of IEEE ICFEC*, 2018.

[78] W. Zhang, B. Han, and P. Hui, "Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking," in *Proc. of ACM Conference on Multimedia*, 2018.

[79] T. Y.-H. Chen, H. Balakrishnan, L. Ravindranath, and P. Bahl, "Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices," in *Proc. of ACM SenSys*, 2015.

[80] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-time Object Detection for Mobile Augmented Reality," in *Proc. of ACM MobiCom*, 2019.

[81] P. Jain, J. Manweiler, and R. Roy Choudhury, "OverLay: Practical Mobile Augmented Reality," in *Proc. of ACM MobiSys*, 2015.

[82] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *Proc. of USENIX OSDI*, 2018.

[83] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live Video Analytics at Scale with Approximation and Delay-Tolerance," in *Proc. of USENIX NSDI*, 2017.

[84] Q. Liu, S. Huang, J. Opadere, and T. Han, "An Edge Network Orchestrator for Mobile Augmented Reality," in *Proc. of IEEE INFOCOM*, 2018.

[85] T. Tan, and G. Cao, "FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile," in *Proc. of IEEE INFOCOM*, 2020.

[86] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet Load Balancing in Wireless Metropolitan Area Networks," in *Proc. of IEEE INFOCOM*, 2016.

[87] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online Job Dispatching and Scheduling in Edge-Clouds," in *Proc. of IEEE INFOCOM*, 2017.

[88] X. Chen, L. Jiao, W. Li, X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[89] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "JALAD: Joint Accuracy- and Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution," in *Proc. of IEEE ICPADS*, 2018.

[90] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "MobiQoR: Pushing the Envelope of Mobile Edge Computing Via Quality-of-Result Optimization," in *Proc. of IEEE ICDCS*, 2017.

[91] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications," in *Proc. of ACM MobiSys*, 2017.

[92] Y. Wu *et al.*, "A learning-based expected best offloading strategy in wireless edge networks," in *Proc. of IEEE GLOBECOM*, 2019.

[93] F. Wang *et al.*, "DeepCast: Towards Personalized QoE for Edge-Assisted Crowdcast With Deep Reinforcement Learning," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[94] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint Configuration Adaptation and Bandwidth Allocation for Edge-based Real-time Video Analytics," in *Proc. of IEEE INFOCOM*, 2020.

[95] P. Yang *et al.*, "Edge coordinated query configuration for low-latency and accurate video analytics," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4855–4864, 2020.

[96] C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "VideoEdge:Processing Camera Streams Using Hierarchical Clusters," in *Proc. of IEEE/ACM SEC*, 2018.

[97] Q. Liu, and T. Han, "DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing," in *Proc. of IEEE ICNP*, 2018.

[98] P. Yang, F. Lyu, W. Wu, N. Zhang, L. Yu, and X. Shen, "Edge coordinated query configuration for low-latency and accurate video analytics," *IEEE Trans. on Industrial Informatics*, vol. 16, no. 7, pp. 4855–4864, 2020.

[99] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach," in *Proc. of IEEE INFOCOM*, 2019.

[100] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing," in *Proc. of IEEE INFOCOM*, 2019.

[101] L. Chen and J. Xu, "Budget-constrained edge service provisioning with demand estimation via bandit learning," *IEEE JSAC*, vol. 37, no. 10, pp. 2364–2376, 2019.

[102] T. Salonidis *et al.*, "Online optimization of 802.11 mesh networks," in *Proc. of ACM CoNEXT*, 2009.

[103] S. K. Saha *et al.*, "Power-throughput tradeoffs of 802.11n/ac in smartphones," in *Proc. of IEEE INFOCOM*, 2015.

[104] M. A. Bagheri and Q. Gao, "An efficient ensemble classification method based on novel classifier selection technique," in *Proc of ACM WIMS*, 2012.

[105] T. Lan *et al.*, "An axiomatic theory of fairness in network resource allocation," in *Proc. of IEEE INFOCOM*, 2010.

[106] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[107] M. Kodialam and T. Nandagopal, "Characterizing achievable rates in multi-hop wireless mesh networks with orthogonal channels," *IEEE/ACM Transactions on Networking*, vol. 13, Aug 2005.

[108] M. Kodialam and T. Nandagopal, "Characterizing the capacity region in multi-radio multi-channel wireless mesh networks," in *Proc. of ACM MobiCom*, 2005.

[109] Z.-q. Luo and P. Tseng, "On the convergence rate of dual ascent methods for linearly constrained convex minimization," *Math. Oper. Res.*, vol. 18, pp. 846–867, Nov. 1993.

[110] F. Kaup, P. Gottschling, and D. Hausheer, "Powerpi: Measuring and modeling the power consumption of the raspberry pi," in *Proc. of IEEE LCN*, 2014.

[111] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[112] G. Iosifidis *et al.*, "Incentive mechanisms for user-provided networks," *IEEE Comm. Magazine*, vol. 52, no. 9, 2014.

[113] D. Yang, X. Fang, and G. Xue, "Truthful auction for cooperative communications," in *Proc. of ACM Mobihoc*, 2011.

[114] D. Levin *et al.*, "Bittorrent is an auction: Analyzing and improving bittorrent's incentives," in *Proc. of ACM Sigcomm*, 2008.

[115] A. Galanopoulos, G. Iosifidis, and T. Salonidis, "Optimizing data analytics in energy constrained iot networks," in *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2018.

[116] J. Huang, Z. Han, M. Chiang, and H. V. Poor, "Auction-based distributed resource allocation for cooperation transmission in wireless networks," *Proc. of IEEE Globecom*, 2007.

[117] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Econ. Theory*, vol. 29, no. 2, pp. 265 – 281, 1983.

[118] P. Maille and B. Tuffin, "Why vcg auctions can hardly be applied to the pricing of inter-domain and ad hoc networks," *Proc. of NGI*, 2007.

[119] R. P. McAfee, "A dominant strategy double auction," *Journal of Economic Theory*, vol. 56, no. 2, pp. 434 – 450, 1992.

[120] H. Xu *et al.*, "A secondary market for spectrum," in *Proc. of IEEE INFOCOM*, 2010.

[121] X. Chen, Z. Zhou, W. Wu, D. Wu, and J. Zhang, "Socially-motivated cooperative mobile edge computing," *IEEE Network*, vol. 32, no. 6, pp. 177–183, 2018.

[122] R. Johari and J. Tsitsiklis, "Efficiency of scalar-parameterized mechanisms," *Operations Research*, vol. 57, no. 4, pp. 823–839, 2009.

[123] F. Kelly *et al.*, "Rate control for communication networks: Shadow prices, proportional fairness, stability," *J. of Oper. Res. Soc. 29(3)*, 1998.

[124] G. Iosifidis *et al.*, "A double auction mechanism for mobile data offloading markets," *IEEE/ACM Trans. on Networking*, vol. 23, no. 5, 2015.

[125] D. Zhang *et al.*, "Double auction based multi-flow transmission in software-defined and virtualized wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 12, 2017.

[126] L. Johansen, "Price-taking behavior," *Econometrica*, vol. 4, no. 7, 1977.

[127] S. Shakkottai and R. Srikant, "Network optimization and control," *Foundations and Trends in Networking*, vol. 2, no. 3, 2007.

[128] A. V. Nefian, "Georgia tech face db." `www.anefian.com/research/face_reco.htm`.

[129] A. Aziz, D. Starobinski, and P. Thiran, "Understanding and tackling the root causes of instability in wireless mesh networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 4, pp. 1178–1193, 2011.

[130] B. Radunović *et al.*, "Horizon: Balancing tcp over multiple paths in wireless mesh network," in *Proc. of ACM MobiCom*, pp. 247–258, 2008.

[131] A. Warrier *et al.*, "Diffq: Practical differential backlog congestion control for wireless networks," in *Proc. of IEEE INFOCOM*, 2009.

[132] M. Haenggi *et al.*, "Stochastic geometry & random graphs for analysis & design of wireless networks," *IEEE JSAC*, vol. 27, no. 7, 2009.

[133] H. Yu and M. J. Neely, "On the convergence time of the drift-plus-penalty algorithm for strongly convex programs," in *IEEE CDC*, 2015.

[134] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends in Optimization*, vol. 1, no. 1, pp. 1–144, 2006.

[135] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[136] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 466–478, 1993.

[137] X. Lin, N. B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE JSAC*, vol. 24, pp. 1452–1463, Aug 2006.

[138] Y. Yi, A. Proutière, and M. Chiang, "Complexity in wireless scheduling: Impact and tradeoffs," in *ACM Mobihoc*, 2008.

[139] L. X. Bui, S. Sanghavi, and R. Srikant, "Distributed link scheduling with constant overhead," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 1467–1480, Oct 2009.

[140] A. Nedić and A. Ozdaglar, "Approximate primal solutions and rate analysis for dual subgradient methods," *SIAM Journal on Optimization*, vol. 19, no. 4, pp. 1757–1780, 2009.

[141] V. Valls and D. J. Leith, "Max-weight revisited: Sequences of nonconvex optimizations solving convex optimizations," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 2676–2689, Oct 2016.

[142] X. Wei and M. J. Neely, "Primal-dual frank-wolfe for constrained stochastic programs with convex and non-convex objectives," *arXiv preprint arXiv:1806.00709*, 2018.

[143] J. Fan *et al.*, "Deadline-aware task scheduling in a tiered iot infrastructure," in *IEEE GLOBECOM*, 2017.

[144] A. Destounis, G. Paschos, and I. Koutsopoulos, "Streaming big data meets backpressure in distributed network computation," in *Proc. of IEEE INFOCOM*, 2016.

[145] J. Zhang *et al.*, "Optimal control of distributed computing networks with mixed-cast traffic flows," in *Proc. of IEEE INFOCOM*, 2018.

[146] V. Valls, G. Iosifidis, and T. Salonidis, "Maximum lifetime analytics in iot networks," in *Proc. of IEEE INFOCOM*, 2019.

[147] A. Gupta, X. Lin, and R. Srikant, "Low-complexity distributed scheduling algorithms for wireless networks," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 1846–1859, Dec 2009.

[148] M. Kodialam and T. Nandagopal, "Characterizing the capacity region in multi-radio multi-channel wireless mesh networks," in *ACM MobiCom*, 2005.

[149] D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Athena Scientific, 2003.

[150] M. Jaggi, "Revisiting Frank-Wolfe: Projection-free sparse convex optimization," in *ICML*, 2013.

[151] K. Kiwiel *et al.*, "Lagrangian relaxation via ballstep subgradient methods," *Math. Oper. Res.*, vol. 32, 2007.

[152] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[153] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[154] A. Gelman and J. Hill, *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Analytical Methods for Social Research, Cambridge University Press, 2006.

[155] M. J. Neely, "Energy optimal control for time-varying wireless networks," *IEEE Transactions on Information Theory*, vol. 52, pp. 2915–2934, July 2006.

[156] L. Li, M. Pal, and Y. R. Yang, "Proportional fairness in multi-rate wireless lans," in *Proc. of IEEE INFOCOM*, 2008.

[157] Z. Chen *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proc. of ACM/IEEE Symposium on Edge Computing*, 2017.

[158] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 6, no. 4, pp. 325–327, 1976.

[159] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of USENIX OSDI*, pp. 265–283, 2016.

[160] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "An online algorithm for task offloading in heterogeneous mobile clouds," *ACM Trans. Internet Technol.*, vol. 18, no. 2, 2018.

[161] G. Ananthanarayanan *et al.*, "Real-Time Video Analytics: The Killer App for Edge Computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[162] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and Efficient Object Detection in Edge Computing: Challenges and Solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[163] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *Proc. of QoMEX*, 2016.

[164] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[165] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *arXiv*, vol. abs/1405.0312, 2014.

[166] A. Galanopoulos, V. Valls, G. Iosifidis, and D. J. Leith, "Measurement-driven Analysis of an Edge-Assisted Object Recognition System," in *Proc. of IEEE ICC*, 2020.

[167] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. MIT press Cambridge, MA, 2006.

[168] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. of ICML*, 2010.

[169] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, "Algorithms for Hyper-Parameter Optimization," in *Proc. of NIPS*, 2011.

[170] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," in *Proc. of ACM SIGKDD KDD*, 2013.

[171] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. d. Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[172] A. Galanopoulos, J. Ayala-Romero, D. J. Leith, and G. Iosifidis, "Edge-Dataset Description." `https://github.com/apgalano/Edge-Dataset`, 2020.

[173] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Proc. of NIPS*, 2015.

[174] O. Alipourfard et al., "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in *Proc. of USENIX NSDI*, 2017.

[175] Q. Liu, and T. Han, "VirtualEdge: Multi-Domain Resource Orchestration and Virtualization in Cellular Edge Computing," in *Proc. of IEEE ICDCS*, 2019.

[176] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, 2002.

[177] H. Junya and T. Akimichi, "Optimality of Thompson Sampling for Gaussian Bandits Depends on Priors," *arXiv preprint arXiv:1311.1894*, 2013.

[178] S. Filippi, O. Cappe, A. Garivier, and C. Szepesvári, "Parametric Bandits: The Generalized Linear Case," in *Proc. of NIPS*, 2010.

[179] Y. Russac, O. Cappé, and A. Garivier, "Algorithms for Non-Stationary Generalized Linear Bandits," *arXiv preprint arXiv:2003.10113*, 2020.

[180] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting," *IEEE Trans. on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

[181] Y. Sui, A. Gotovos, J. W. Burdick, and A. Krause, "Safe Exploration for Optimization with Gaussian Processes," in *Proc. of ICML*, 2015.

[182] Y. Sui, V. Zhuang, J. W. Burdick, and Y. Yue, "Stagewise Safe Bayesian Optimization with Gaussian Processes," in *Proc. of ICML*, 2018.

[183] S. Amani, M. Alizadeh, and C. Thrampoulidis, "Regret Bounds for Safe Gaussian Process Bandit Optimization," *arXiv preprint arXiv:2005.01936*, 2020.

[184] S. R. Chowdhury and A. Gopalan, "On Kernelized Multi-Armed Bandits," in *Proc. of ICML*, 2017.

[185] A. Galanopoulos, J. A. Ayala-Romero, G. Iosifidis, and D. J. Leith, "Bayesian Online Learning for MEC Object Recognition Systems," in *Proc. of IEEE GLOBECOM*, 2020.

[186] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Edgebol: Automating energy-savings for mobile edge ai," in *Submitted to ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2021.

[187] R. Allesiardo, R. Féraud, and D. Bouneffouf, "A neural networks committee for the contextual bandit problem," in *International Conference on Neural Information Processing*, 2014.

[188] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella, "D-dash: A deep q-learning framework for dash video streaming," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 703–718, 2017.

[189] J. A. Ayala-Romero, J. J. Alcaraz, A. Zanella, and M. Zorzi, "Online learning for energy saving and interference coordination in hetnets," *IEEE JSAC*, vol. 37, no. 6, pp. 1374–1388, 2019.

[190] A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi, "Incorporating behavioral constraints in online ai systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[191] Y. Han *et al.*, "Convergence of edge computing and deep learning: A comprehensive survey," *CoRR*, vol. abs/1907.08349, 2019.

[192] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, 2019.

[193] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[194] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. of IEEE WCNC*, 2018.

[195] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep Q-learning model for energy-efficient edge scheduling," *IEEE Trans. on Services Computing*, vol. 12, no. 5, pp. 739–749, 2019.

[196] H. Sun *et al.*, "Learning to optimize: Training deep neural networks for interference management," *IEEE Transactions on Signal Processing*, vol. 66, no. 20, pp. 5438–5453, 2018.

[197] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "Lorm: Learning to optimize for resource management in wireless networks with few training samples," *IEEE Trans. on Wireless Communications*, vol. 19, no. 1, pp. 665–679, 2020.

[198] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid mec networks," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6252–6265, 2020.

[199] H. Uzawa, "Iterative methods in concave programming," *Studies in Linear and Nonlinear Programming, K. Arrow, L. Hurwicz, and H. Uzawa*, no. 1, pp. 154–165, 1958.