DYNAMIC-ORIENTED RESOURCE MANAGEMENT FOR MOBILE WIRELESS SYSTEMS

By

Fan Qiu

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

May, 2014

Nashville, Tennessee

Approved:

Professor Yuan Xue

Professor Gabor Karsai

Professor Douglas C. Schmidt

Professor Xenofon D. Koutsoukos

Professor William H. Robinson

To my dear parents
and my lovely wife for their support and encouragement

ACKNOWLEDGEMENTS

Looking back at the end of my Ph.D journey, I feel so blessed to have my family and friends, who have warmly helped and supported me along the way.

Foremost, I would like to express my deepest appreciation and thanks to my advisor, Professor Yuan Xue, who has guided me as a passionate, inspirational, and supportive mentor throughout my graduate study. This work would not have been possible without her support and patience.

I want to express sincere gratitude to my dissertation committee: Professor Douglas C. Schmidt, Professor Gabor Karsai, Professor Xenofon D. Koutsoukos and Professor William H. Robinson. I appreciate their advice and guidance.

I would also like to thank my colleagues at VANETS group and the Institute for Software Integrated Systems. They are like families sharing frustration, joy and hope. I will never forget their help and encouragement. I am very thankful to my friends here, with whom I have had a great time at Vandy. I would especially like to thank Dr. Yi Cui, for inspiring me at the beginning of this journey, to follow my dream and pursue a Ph.D degree in Computer Science.

Last but not least, I am particularly grateful to my parents and my wife Danqing. I would not have come this far without their love and support. This dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Mobile wireless technologies have been fundamentally facilitating data communications by offering a rich set of features such as ubiquitous connectivity, mobility and scalability. To date, mobile wireless systems are employed in a vast variety of application areas. Notably, mobile smart computing platforms, like smartphones, tablets, wearable computing devices and mobile game consoles, have become widely popular. In the fourth quarter of fiscal year 2012 alone, 26.9 million *iPhone*s were sold in the world [1]. It is reported by *Google* that more than 750 million *Android* mobile devices have been activated globally by March, 2013 [2].

Although today's mobile wireless systems can offer impressive performance, they are confronted with a twofold challenge. First, the accelerated proliferation of mobile wireless devices is creating explosive demand beyond the capabilities of wireless networks. *Cisco* claims that global mobile data traffic grew 70 percent in 2012 and monthly global mobile data traffic will surpass 10 exabytes in 2017 [3]. Recently, huge efforts have been dedicated to expanding the capabilities of network infrastructures. For instance, WiFi has been considered to be a primary candidate of data traffic offloading from the cellular networks [4]. However, this is just a temporary remedy, as the limitation on wireless spectrum will ultimately emerge, and performance issues like network congestion and slow data speed keep arising. Second, the resources on mobile wireless devices, like computational power and battery energy, are highly constrained, which may severely limit the quality of service of mobile applications. Unlike conventional computers, currently smart mobile devices are barely powerful enough to drive

computation-intensive applications, hence many mobile applications need to rely on remote computing, such as *Apple*'s *Siri* and *Google*'s *Translate* [5]. Battery life is another long lasting issue, which is now challenged by faster processors, sharper displays and sophisticated sensors on wireless mobile platforms [6]. To sum up, the resource problem of wireless mobile systems is the bottleneck of the performance of mobile applications, and it is becoming increasingly severe while the mobile applications today are largely developed in a resource oblivious manner [7].

Problem Statement and Research Goal

Since the prevalence of Internet, significant research efforts have been contributed to the problem of efficient network resource usage among network components. One notable example is TCP, a resource management approach used for congestion control, which implicitly preserves properties like fairness and stability. Due to the inherent nature of mobile wireless systems, resource management is extremely important, which basically follows two directions:

- Optimizing the internal resource usage in network architecture.
- Augmenting the capabilities of mobile platforms by integrating external computational resources.

Compared with wireline networks, optimizing resource usage among wireless nodes is uniquely challenged by the inherent resource sharing feature of wireless medium: wireless links may interfere with each other if they are close enough in spatial vicinity. A large body of work in literature has investigated the resource management problem for mobile wireless systems. In Fig. I.1 we categorize the existing solutions into several classes and associate them with the corresponding layer(s) in the network architecture.

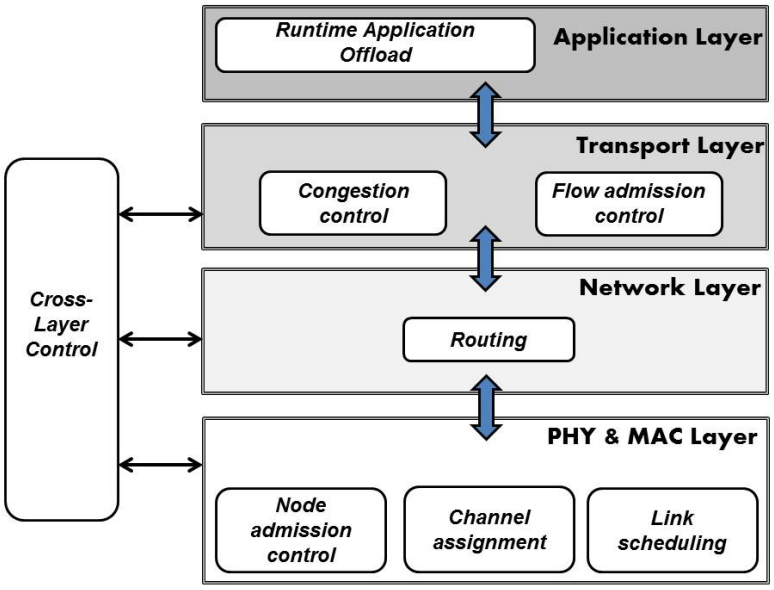


Figure I.1: Wireless resource management.

For example, the channel assignment and scheduling solutions [8, 9, 10] deal with resource allocation on the link layer and the medium access control (MAC) layer. The routing solutions [11, 12] addresses the route selection problem on the network layer. The congestion control and flow admission control algorithms manage the data flows on the transport layer. The mobile application offload techniques provide a therapy on the application layer [5, 6, 13]. Recently, to leverage the popular optimization framework [14] with spatial dependency issue and other dynamics in wireless networks, the joint design approaches are proposed to manage sources across multiple layers. The global optimization problem can be decomposed into sub-problems corresponding to different layers of the network architecture. The sub-problems are coordinated by variables that reflect the supply-demand conditions of network resources to achieve the global optimum. For instance, the joint congestion control and link scheduling approach is broadly employed in existing works [15, 16, 17, 18, 19].

In spite of the tremendous research efforts involved, one noteworthy problem with existing solutions is how to deal with dynamics, which stem from wireless environments, varying user demands, network topology changes, etc. If not deliberately handled, the dynamic factors may cause severe performance degradation.

The previous studies on wireless network resource management strongly rely on two assumptions: 1) feedback delay is negligible and 2) the wireless channels are considered to be static or time-varying following a simple model, such as a stable Markov chain [15]. In remarkable contrast to these simplified assumptions, feedback delays commonly exist in networked systems. Additionally, the real-world channel dynamics are highly complex. Delay effects, coupled with time-varying channels, significantly challenge the design of congestion control and scheduling in wireless networks. For example, delay effects incur stability problems. Some congestion control algorithms are sensitive to delay uncertainty. The sufficient conditions for local stability may vary a lot with the presence of delays. Moreover, in time-varying wireless networks, the rate adaptation at source nodes respond to the outdated and inaccurate congestion price signal, resulting in over-provisioning of sending rates and packet losses. This motivates the robust and optimized all-weather resource management solutions which not only efficiently utilize wireless resources, but consistently keep up the performance under all scenarios.

The problem also exists in the application layer solutions, for example, the offloading solutions which augment the capabilities of mobile wireless systems by offloading tasks from mobile terminals to external resource-rich devices. The resources on mobile terminals, like CPU power, storage and battery energy, are very limited. Research studies in literature have proposed solutions of offloading mobile applications to resource-rich systems, such as cloud systems. Then mobile applications are executed remotely in order to reduce the internal resource consumption on mobile terminals. However, designing an offload system is highly difficult because it usually needs to deal with *when*,

what and *how* to offload a mobile application. *When* to offload refers to the decision problem of whether a mobile task should be offloaded at a particular time instance. As network condition and workload are not always consistent, the decision of offloading should be adaptive to the change of environment, instead of following the static server-client paradigm. *What* to offload is the problem of partitioning a mobile application according to the resource usage profile. Simply put, the system needs to identify the resource intensive components of the application. *How* to offload refers to the implementation of the offload mechanism. The existing works either use a simplified model of dynamics [5, 20] or adopt offload mechanism with high overhead [6, 13], so designing an efficient and robust mobile application offload system remains an open problem.

Based on the discussions above, this dissertation makes research efforts towards dynamic-oriented resource management for mobile wireless systems. Our research goals are as follows:

- To design an optimization based cross-layer resource management solution for wireless networks, which is adaptive to time-varying channel and feedback delay.
- To design and implement a lightweight mobile application offload system, which offers optimal execution schemes for a mobile application, according to dynamic user demand and network condition.

Research Approach and Contributions

Our general research approach can be summarized to three steps.

- The first step is to *identify* and *characterize* the dynamic factors. For the two different resource management approaches introduced earlier, various dynamic

factors are considered. We explicitly model the dynamic factors that have significant impact on the performance of the mobile wireless systems. Time-varying channel capacity and feedback delay are considered in the first category of solution. User demand and network condition are considered in the second category of solution. We first profile the capacity vector, then we use time series analysis to identify the critical time scale of the capacity vector.

- The second step is to integrate the dynamic factors into the resource management problem formulation. For example, we have used the widely adopted *network utility maximization* framework, which contains an optimization objective and a group of constraints. By introducing dynamics into the framework, the solution space will be reshaped.

- The third step is to address the resource management problem and obtain the optimal solution. The problem can either be implicitly solved by distributed controllers (Chapter III and Chapter IV) or by a solver (Chapter V).

Following the above guideline, we present a two-tier solution to resource management for wireless mobile systems. The two tiers correspond to different layers. In Fig. I.2 we summarize the proposed techniques. The first tier of management scheme involves cross-layer techniques that improve resource allocation through interactions across multiple lower layers (the medium access layer, the network layer and the transport layer). It includes the following two techniques: *robust joint congestion control and scheduling* (Chapter III) and *time scale decomposition based cross-layer management* (Chapter IV). The second tier of management scheme focuses on the application layer and addresses mobile application related resource allocation problems. It includes the *dynamic-aware mobile application offload system* (Chapter V).

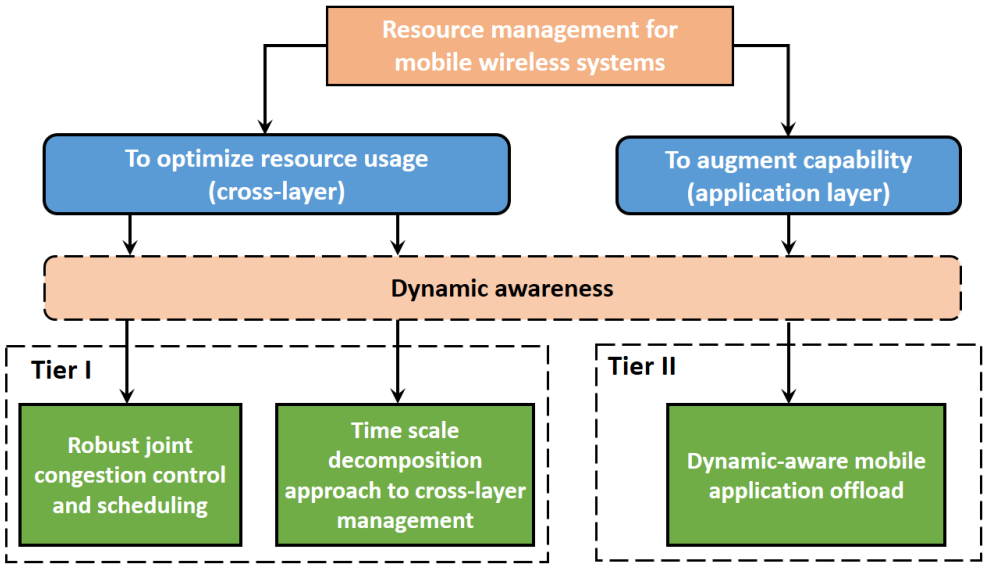


Figure I.2: Summary of the proposed techniques.

The two tiers address the resource management problem under different scenarios. Nevertheless, they are still closely related, as the layers in the network architecture are not independent. The overall performance of a wireless mobile system is not only determined by each layer, but also by the interactions between different layers. Furthermore, both of the two tiers are centered on dynamic-awareness, which guarantees that the resource management schemes are robust with the presence of dynamic factors.

Our contributions have been summarized as follows.

- We present a robust joint congestion control and scheduling algorithm for time-varying multi-hop wireless networks with feedback delay. The algorithm is operated over a virtual capacity space: the dominant part of a virtual link capacity is the low-frequency components of the original link capacity, so it is robust to channel variations. The algorithm is adaptive to heterogeneous channel conditions, by adjusting the cut-off frequency during the capacity decomposition process. We

provide the sufficient conditions for the Lyapunov stability of the control algorithm, with the presence of feedback delay. The conditions are utilized to achieve runtime stability.

- We present a time-scale decomposition approach to joint congestion control and scheduling for wireless networks. We use the technique to investigate the risk imposed by unpredictable fast-time scale and perform risk-benefit analysis by introducing a penalty function into the optimization framework. This approach provides a solid theoretical foundation towards the understanding and the management of dynamics in wireless networks. We conduct extensive trace-driven packet-level simulation study and demonstrate that our approach outperforms the existing joint congestion control and scheduling solutions in the presence of delay under realistic channel conditions.

- We design and implement an energy-efficient mobile application offload system that supports seamless and dynamic-aware mobile application offloading. We establish energy cost models for different execution schemes, including the local execution energy model and the remote execution energy model. We introduce a novel algorithm for runtime decision-making of execution scheme to minimize energy consumption.

Dissertation Organization

The remainder of this dissertation is organized as follows. Chapter II overviews the existing work in literature on resource management for mobile wireless systems. In Chapter III, we detail a robust joint congestion control and scheduling approach for time-varying multi-hop wireless networks with feedback delay. In Chapter IV we discuss a time-scale decomposition approach to joint congestion control and penalty-aware

scheduling. Then Chapter V describes our design of a dynamic-aware mobile application offload system. Finally, Chapter VI summaries the completed work and outlines several future research directions.

CHAPTER II

RELATED WORK

Our work is largely related with two research areas. 1) Joint congestion control and scheduling for wireless networks, particularly the research studies focusing on time-varying channel condition and feedback delay. 2) Runtime mobile application offload and the related studies, such as dynamic profiling, application migration, and remote execution. This chapter overviews the related works and summarizes the open issues.

Optimization Based Wireless Network Resource Management

In this section, we review the optimization based wireless network resource management techniques and discuss the open issues.

Network Resource Usage Optimization

Optimization based approaches are widely utilized to improve the performance of network systems. *Network Utility Maximization* (*NUM*) has been one of the most notable techniques in network optimization, since it was first introduced in the 1960s. Kelly et al. [14] further extended *NUM* by introducing a pricing mechanism for resource management. This pricing approach is used in network congestion control optimization, such as [21, 22, 23]. *NUM* tries to capture the network dynamics by embracing user objectives (the utility function) and resource sharing constraints. Usually, *NUM* defines a utility function based on the types of optimality and fairness the algorithm wants to achieve. The utility function should be concave and second order differentiable. The optimization framework incorporates network dynamics like capacity, scheduling and

power constraints, by formulating them as constraint inequalities. Solution to the optimization problem can be obtained via primal, dual or primal-dual algorithms. Since its birth, *NUM* has gained tremendous success in resource management for both wire-line and wireless networks. Other optimization techniques, like multi-objective optimization involving the user objective and operator objective, and game theorem based optimization [10] are also effective techniques in optimizing resource management.

In wireless resource optimization problems, the major difference from wire-line networks is the resource sharing mechanism, which is constrained by the interdependencies among wireless links: when one link is used, the links in the same channel and within the interference range of this link cannot be active. Therefore the formulations of wireless optimization problems usually rely on some interference models, for instance, the *conflict graph* [24].

Channel Assignment and Scheduling

Optimization algorithms on channel assignment and scheduling focus on improving the quality of service by controlling media access, like wireless link access [10, 25, 26, 27, 28, 29], channel usage [8], etc. A notable scheduling policy is the well known throughput optimal scheduling [30]: *Maximal Matching Scheduling*, which is used by many existing works like [9, 15, 18] and [31].

The capacity of wireless networks can be greatly extended by using multi-channel multi-radio (*MC-MR*) interfaces. Each radio can switch between several orthogonal channels to avoid interference. The work of [8] proposes a robust resource provision channel assignment algorithm for *MR-MC* wireless networks. The solution provides guaranteed *QoS* under channel variability and external interference. The channel assignment problem is formulated to a *NUM* framework, where the utility is defined as

a function of interference margin, which represents the maximal degree of interference that can be tolerated given the targeted transmission rate of a link, so maximal network utility implies better robustness to interference. The idea of combining network uncertainty modeling and utility maximization is important for resource management with unpredicted network dynamics. However, the significance of this work is limited by the assumption that the link traffic demands are pre-determined.

Scheduling policies determine how to assign the time slots to different wireless links under the same channel, such that they are feasible for a given workload. For instance, the *largest debt first policies* [10] are proposed to satisfy the delivery ratio of a set of clients: the average proportion of transmitting time of a client is at least the implied workload. The *largest time-based debt first policy* tries to make every client get a share of time that is at least as large as the implied workload. The *largest weighted-delivery debt first policy* attempts to make every client obtain a delivery ratio at least equal to the desired delivery ratio. Here the concept of *debt* is similar to the notion of *backlog* used in some congestion control algorithms. The scheduling policies can satisfy the desired delivery ratio. The study makes a valuable step towards *QoS* related wireless scheduling design.

The *Max-Weight* algorithm [30] is a popular scheduling strategy, which determines the transmission priority based on the product of queue length and current channel rate. *Max-Weight* involves both user demands and channel variation, and was considered to provide throughput optimality. However, the work in [32] proves that *Max-Weight* is not always throughput optimal under certain flow-level dynamics. Therefore, the study of [26] proposes a novel scheduling policy that is adaptive to flow-level dynamics. A new metric, *system workload*, is used to decide the priorities of channel allocation to wireless links. The workload measures the backlog of a short-lived flow. By comparing

the backlogs of the short-lived flows to the queue length and channel conditions of long-lived flows, the algorithm generates the appropriate scheduling which is proved to be throughput optimal even under flow-level dynamics.

Routing

Generally, routing is a mechanism used to find the path(s) from a source to a destination in a network, which can also be formulated as optimization problems. The algorithms solving these problems usually claim analytical properties like resource utilization optimality and throughput fairness [12].

Optimal routing under conflict graph [24] is a throughput optimal routing strategy. It relies on a conflict graph model: the link dependencies in a wireless network can be modeled by a graph, in which the interference between two links is mapped to an edge of graph. A throughput optimal routing algorithm is derived from a multi-commodity flow problem, in which the graph model is used to formulate the constraint set.

The intuitive shortest path selection may lead to hot-spot regions with too many flows, so the wireless links within these regions are overloaded. To make the routing algorithms aware of congestion, the *congestion aware path selection* algorithm [33] defines a congestion degree for a path that reflects the availability (determined by the flows through it and the flows in its adjusted interference set) of links along the path. Then this metric is used to guide path selection to avoid congestion.

The classical routing algorithms usually consider routing of flows with fixed demands, however, the user demand on a flow rate could be dynamic. The *uncertainty-aware optimal routing* [34] investigates how to make wireless routing accommodate the flows with uncertain demand. It formulates the flow rates to a statistical model and solve the problem in a *NUM* framework to achieve throughput optimization.

Wireless networks are often constrained by the power of wireless clients. The *power-aware routing* [11] aims at finding the most energy-efficient route. It employs an energy-opportunistic weighted minimum energy routing strategy based on the energy consumption model to maximize the total revenue of a network.

Congestion Control

Congestion control algorithms adapt the sending rates at the source nodes to avoid traffic congestion in networks. Since the seminal work by Kelly [14], congestion control problems have been investigated under the optimal resource allocation framework. Algorithms in this class explore how to maximize network utility under the constraint of network capacity by allocating sending rates and meanwhile maintain fairness among the competing flows, such as the *max-min fairness* and the *proportional fairness*. To this end, many existing works [21, 23, 35] adopt an optimization variable called *price* that reflects the demand-supply relationship between the available network resource and bandwidth utilization. Then a flow source adjusts the sending rate according to the aggregate price charged along the flow route. Although Kelly's optimization framework has provided a very solid foundation for the resource allocation problem in wire-line networks, it barely captures the nature of wireless resource capacity, because wireless channel condition could be highly fluctuating and easily leads to capacity constraint violation and system oscillation.

Since the achievable capacity of wireless networks is not explicit, Xue et al. [23] use maximal clique to model the resource sharing of wireless nodes. In the contention graph of a wireless network, wireless links are mapped to the vertices and the interference between two links is mapped to an edge in the graph. In a maximal clique of the contention graph, any two vertices are connected, so only one link in the clique can be active at any

time, thus the contention graph can be mapped to a set of cliques. Then each clique is associated with a *shadow price*: the charge for a unit flow to use a link in the clique. Finally the resource management problem can be solved in the *NUM* framework.

Cross-Layer Optimization

The cross-layer control algorithms have been popular in wireless resource management. Examples include joint congestion control and scheduling, joint congestion control and routing, and joint congestion control and power control. The idea of cross-layer optimization is motivated by the inherent nature of wireless resource sharing: feasible rate region are not explicitly measurable due to wireless interference [15], therefore the optimal resource allocation could hardly be achieved from a single layer. To fully utilize the scarce wireless spectrum, the wireless resource management problem can be formulated under an optimization framework. Then the problem is decomposed into sub-problems corresponding to different layers of the network architecture. The sub-problems are coordinated by variables that reflect the supply-demand conditions of network resources to achieve the global optimum [36].

In this dissertation we focus on the joint congestion control and scheduling approaches [15, 19, 37, 38, 39, 40, 41, 42, 43, 44]. As we discussed earlier in this chapter, the objective of a congestion control algorithm is to achieve fairness and utility maximization, and the objective of scheduling is to achieve proper link utilization for interference avoidance. The two sub-problems are coordinated by optimization variables (feedback information), for example, link congestion price.

A dual decomposition based joint rate control and scheduling approach is proposed by Lin et al. [38] to solve the resource management problem for multi-hop wireless networks. This approach is analytically proved to ensure fairness and quality of service.

The study of [40] introduces a joint congestion control, scheduling and routing algorithm for utility maximization. Meanwhile, the system stability is related to queue overflow: the arrival data rate of any queue cannot be greater than the departure rate. The decomposition of the Lagrangian of the primal problem naturally leads to three subproblems: congestion control, scheduling, and routing. The congestion control computes the flow rate vector that maximizes the utility function. The scheduling finds a policy that stabilizes the system. The routing algorithm routes the flows following the selected links.

However, the optimal control scheme of the joint congestion control and scheduling is essentially a complex global optimization problem. Solving such a problem is computationally expensive. Therefore in the work of [18], an imperfect scheduling is used to achieve a sub-optimal solution to the *max-matching scheduling* problem. The study further provides discussion on the impact of the *imperfect scheduling* performance of the cross-layer congestion control. The imperfect scheduling based cross-layer control has properties such as system stability and utility sub-optimality. The results provide a significant step towards designing fully distributed cross-layer congestion control schemes for multi-hop wireless networks.

In our previous work [45], we have investigated the joint rate allocation and scheduling problem with end-to-end time delay constraints. In our model, the end-to-end delay of a flow can be adjusted by controlling the per-link delays via a novel parameter called *Virtual Link Capacity Margin* (VLCM), which is the measurement of the gap between the schedulable link capacity and the maximum allowable flow rate over a link. We solve the optimization problem via its dual decomposition through two price variables derived with regard to the constraints: the link congestion price that reflects the relationship between the traffic load and the capacity of a link, and the flow delay price that reflects the

margin between the average packet delay and the delay constraint of a flow. The proposed technique presents a new attempt toward optimal cross-layer resource allocation with quality of service constraints.

The cross-layer congestion control and scheduling approach is extended by [16] to time-varying networks, where the channel states are assumed to follow recursive Markov chain models. The resource management problem is formulated as a utility maximization framework with rate allocation constraints and scheduling constraints. Then three sub-problems: congestion control, routing and scheduling, are decoupled from the initial global optimization problem according to its dual decomposition.

Dynamic-Aware Network Resource Management

Resource management in wireless networks can be further complicated by external dynamics such as time-varying channel and feedback delay. Recently, providing dynamic-aware resource management has attracted a lot of research attention.

Time delay problem is notoriously intricate in the area of networking: the outdated feedback information could raise various problems such as inaccurate rate/scheduling adjustment, bandwidth under/over utilization and system instability. Therefore, investigating network performance under time delay has been a long lasting research focus, either within the context of wire-line networks [21, 46], or wireless networks [47, 48] and [49].

For instance, [48] studies the resource allocation problem in cellular networks with heterogeneous feedback delays and time-varying channels. The fair and utility-optimal resource allocation is achieved via a combined scheduler-congestion controller. The significance of the work is limited by the assumption that the channel states flow a given statistical model.

The work of [47] proposes an asynchronous congestion control algorithm and a distributed scheduling algorithm for multi-hop wireless networks with fixed channel capacity. They consider unbounded heterogeneous delays in congestion information exchange. The algorithm is stable and supports at least one third of throughput supported by the centralized algorithms. In spite of these elegant properties, this algorithm deals with static channel states, which is a relatively idealistic assumption.

In the study of [49], a primal-dual congestion control algorithm is presented for wireless network with time-varying channels. The study provides sufficient conditions for the algorithm to be locally stable with the presence of feedback delay. Although the proposed algorithm demonstrates certain robustness against time-varying channels, it does not explicitly contains an interference model, which is not negligible in wireless networks. In addition, the focus of the work is to analyze the stability and sensitivity of the congestion control algorithm, rather than design a robust algorithm to handle channel perturbations.

Summary of Open Issues

In spite of the efforts of recent research studies devoted to the the cross-layer resource management for wireless networks, there still lies a wide gap between the existing works and the real wireless environment. To date, the existing studies strongly rely on two assumptions: 1) feedback delay is negligible and 2) the wireless channels vary over time following a simple model, such as a stable Markov chain [16]. Unfortunately, neither of the assumptions holds in real world wireless networks.

Dynamics caused by delay and channel state variations make the scope of the problem significantly different. Wireless channels are usually highly volatile [50]. A real optimal congestion control algorithm needs to make rate allocation based on time-varying

channel states. Meanwhile, delay effects are ubiquitous in wireless networks, so instantaneous delivery of the feedback information (such as congestion price) can barely be guaranteed. Especially, delay effects, coupled with time-varying wireless channels, make the design of a stable congestion control algorithm non-trivial. The problem is even more challenging when combined with link scheduling. Based on the above discussion, we summarize the open issues as follows.

- Rate allocation with channel perturbation: ensuring the time-varying capacity can be adequately utilized by the rate allocation algorithm is difficult with channel perturbation.

- System stability with feedback delay: achieving system stability with the presence of feedback delay is still largely an open issue.

- Penalty-aware scheduling: the *max-weight* scheduling is a throughput-optimal policy, however, it is not aware of the status of resource provisioning with dynamic network conditions.

Dynamic-Aware Mobile Application Resource Management

In this section, we introduce the existing mobile application resource management techniques and discuss the open issues.

Dynamic Mobile Application Offloading

While mobile applications keep gaining popularity and putting increasing demand on hardware capabilities, the *resource shortage* problem of mobile devices becomes pressing. Recently, the *dynamic offload* approach has attracted tremendous research

attention [5, 6, 13, 20, 51, 52]: the resource intensive components of a mobile application are (partially) *migrated* to a remote machine at runtime, such that the resource consumption of the mobile device is reduced. The results of the remote execution will be returned the mobile device. *Dynamic offload* often employs the techniques such as *program partition*, *process migration*, *virtualization*, *dynamic analysis*, etc.

Dynamic offload is closely related to *remote execution* [53, 54, 55, 56, 57], which has been widely used in mobile and pervasive computing. The systems that support remote execution usually require pre-partition for remote execution of resource intensive tasks [13], which is similar to the server-client paradigm. But this approach can barely adapt to runtime dynamics.

More recently, Cloudlets [58] suggested that the remote execution techniques be applied to mobile device environments. This work proposes an infrastructure that enables mobile devices to offload the running applications to the cloudlets within close proximity. In the infrastructure, the device software can be rapidly deployed on a nearby cloudlet through virtual machine (VM) migration techniques. Mobile application offload is leveraged by the *dynamic VM synthesis* technique: a VM overlay is established above the mobile device and the cloudlet, the infrastructure derives the state of the launch VM (mobile application instance) and starts execution on the cloudlet from the suspended state.

MAUI [6] further extends *dynamic offload* by enabling fine-grained energy-aware offload of mobile code. The previous offload systems mostly relied on precise pre-partition or coarse-grained migration (full process or VM), while MAUI supports dynamic and method level code offload. The system provides a programming environment where developers can annotate the methods that are candidates for code offload. MAUI also profiles mobile applications and network condition. At runtime, an optimization

problem solver determines if an annotated method should be offloaded to the remote server.

Similarly, CloneCloud [13] is also a dynamic mobile application offload system, and it goes a step forward by reducing programmer involvement and supporting thread granularity offload. CloneCloud identifies the legal partition points of a mobile application according to its control flow graph, in an offline manner. For instance, the methods that create graphical user interface and the methods that share native states cannot be offloaded. The system uses a dynamic profiler to build a cost model for a mobile application. The offload decision is made by an optimization solver based on the current network state and the estimated offload cost.

COMET [5] is an even finer-grained mobile code offload system which reduces the restriction on the offloadable code. The system can complete transparent migration of application execution based on the *distributed shared memory* (DSM) mechanism. Making use of a VM-synchronization primitive, COMET can simultaneously migrate multiple threads from a mobile device to a remote server.

In contrast to the above approaches, the migration of mobile applications in the *Dynamic Mobile Software Deployment* system [20] relies on software modularization techniques. The system is designed in an OSGI [59] framework. Mobile applications are designed to contain pluggable modules (OSGI bundles), and these modules can be dynamically offloaded and deployed in cloud. One advantage of this design is that it avoids the overhead of virtual machine synchronization.

Summary of Open Issues

Although recently tremendous research progress has been made in dynamic mobile application offload, there are still several open issues in this area.

- *How to model the execution schemes* of a mobile application? Existing works mostly use intuitive (or oblivious) partitioning strategies, such as the resource intensive components based partitioning, or non-native method based partitioning.

- *How to establish the cost models* in dynamic environment? The energy consumption of an offload scheme may vary under different network conditions and different workloads. It remains unclear what variables should be incorporated into the cost model and how to formulate the problem.

- *How to make an offload decision* (local execution or remote execution) at runtime to achieve cost-efficient execution? The decision should be made by evaluating the costs of different schemes based on the cost models. The optimal scheme should have the minimal cost of execution.

CHAPTER III

ROBUST JOINT CONGESTION CONTROL AND SCHEDULING FOR TIME-VARYING WIRELESS NETWORKS

In this chapter we present ROCS: a RObust joint Congestion control and Scheduling algorithm for time-varying multi-hop wireless networks with feedback delay, to bridge the gap between the existing approaches and the reality of wireless networks. The fundamental idea behind ROCS is *Capacity Space Projection*, that combines the *slow time scale* part of the channel capacity and a margin estimated from the *fast time scale* part, to form a new capacity space. Here *slow* indicates the channel varies sufficiently slowly so the resource management algorithm can closely follow the changes, while *fast* implies the dynamic channel condition is hard to capture. In our design, both time scales are in the order of milliseconds.

The resource allocation problem is formulated into a utility maximization framework over the new capacity space. The problem is solved by a control algorithm consisting of link scheduling and congestion control. Link scheduling coordinates wireless link utilization base on maximum weight matching, and congestion control allocates flow rates according to congestion feedback information. Experiments conducted over simulated and real world traces demonstrate that ROCS substantially achieves robustness and efficiency.

The remainder of this chapter is organized as follows. We first give an overview of the proposed technique. Then we introduce the system model and give the problem description. In the following four sections we present our robust joint congestion control and scheduling algorithm and the experimental study. Finally we conclude this chapter by a brief discussion.

Overview

A widely adopted optimization based approach to solve the wireless resource allocation problem is to formulate it under a utility maximization framework. The optimization problem can be solved by a cross-layer approach. Recent studies in this area usually use static channel models and assume feedback delay is negligible. However, the assumptions do not hold under real wireless environments. Wireless channels are highly volatile. Additionally, feedback delay is ubiquitous in wireless networks, so instantaneous delivery of the feedback information can barely be guaranteed. This leads to severe problems like resource over-provisioning or system oscillation.

To address these challenges, we present a *RObust joint Congestion control and Scheduling* algorithm: *ROCS*. The algorithm focuses on the following unsolved critical issues in existing works: 1) *Rate allocation with channel perturbation*: ensuring the time-varying capacity can be adequately utilized by the rate allocation algorithm is difficult with channel perturbation. 2) *System stability with feedback delay*: achieving system stability with the presence of feedback delay is still largely an open issue.

Our solution is centered around the concept of *Virtual Capacity Space*, which is designed to protect a network system against channel perturbation. The original link capacity can be decomposed into slow time scale components and fast time scale components. The fast time scale capacity components are difficult to follow with the presence of feedback delay. We first decompose the original instantaneous channel capacity into a slow time scale part and a fast time scale part. Then our algorithm maps the original capacity space to a virtual capacity space by combining the slow time scale part and a margin estimated from the fast time scale part. The algorithm mainly keeps pace with the slow time scale signal so that it captures the critical changes of the channel. Moreover, it also incorporates the high frequency variations by using the margin.

The cross-layer resource allocation problem is formulated into a utility maximization framework over the newly generated capacity space. The problem can be solved by a joint control algorithm which contains two closely coupled components: link scheduling and congestion control. We establish the sufficient conditions of the stability of the algorithm according to the Lyapunov-Razumikhin theorem. The conditions reveal the relationship between the feedback delay and controller parameters. We further provide a distributed implementation of the joint control algorithm.

System Model

We consider a single channel multi-hop wireless network, consisting of V nodes, collectively denoted as $\mathcal{V}$. The nodes communicate with each other via directed wireless links, denoted as $\mathcal{L}$. The end-to-end flow set is represented by $\mathcal{S}$. Each flow $s \in \mathcal{S}$ with sending rate x_s is associated with a utility function $U_s(x_s)$, which is concave and twice differentiable. The link set the flow s traverses is denoted as $\mathcal{L}(s)$, and the flows incident to link $l \in \mathcal{L}$ are denoted as $\mathcal{S}(l)$.

The communication in a wireless network is subject to location dependent interference. In this work we adopt the concept of conflict graph [24] to model wireless interference. Each vertex in the conflict graph represents a wireless link of the original network. An edge exists between two vertices if their corresponding wireless links interfere with each other. We employ a scheduling mechanism Sc to schedule the wireless links on a slotted time basis: in each time slot, one independent set I is selected from the conflict graph and only the links corresponding to the vertices in I can be active because there is no interference. Let c_l represent the amount of bits that can be transmitted per second along a link l once it is scheduled. We denote the capacity vector by $\boldsymbol{c}$, where $\boldsymbol{c} = (c_l, l \in \mathcal{L})$.

The scheduling Sc essentially determines the maximum feasible rate vector $\hat{\boldsymbol{c}} = (\hat{c}_l, l \in \mathcal{L})$ of the links, where $\hat{c}_l$ is the average capacity over time. $\hat{\boldsymbol{c}}$ is constrained by the feasible capacity space Λ as introduced in [15], which is a convex hull [60, 61] and is defined as $\Lambda := \sum_I \alpha_I r^I$, where $\sum_I \alpha_I = 1$ and $\alpha_I \geq 0$. The L-dimension column vector r^I represents the capacity vector of I, where $r_l^I = c_l$ if $l \in I$, and $r_l^I = 0$ otherwise. We collect the notations in Tab. IV.1.

Notation	**Definition**
$v \in \mathcal{V} = \{1, 2, ..., V\}$	Node set
$s \in \mathcal{S} = \{1, 2, ..., S\}$	End-to-end flow set
$l \in \mathcal{L} = \{1, 2, ..., L\}$	Wireless link set
$\mathcal{L}(s)$	Links the flow s traverses
$\mathcal{S}(l)$	Flows incident to link l
Λ	Original Capacity Space
$\bar{\Lambda}$	Virtual Capacity Space
$C_l^t = \{c_l(t)\}, l \in \mathcal{L}, t \in [1, T]$	Original link capacity series
$\tilde{C}_l^t = \{\tilde{c}_l(t)\}, l \in \mathcal{L}, t \in [1, T]$	Low-pass link capacity series
$\boldsymbol{c} = (c_l, l \in \mathcal{L})$	Original link capacity vector
$\boldsymbol{\upsilon} = (\upsilon_{l,t}, l \in \mathcal{L})$	Link capacity margin vector
$\bar{\boldsymbol{c}} = (\bar{c}_l, l \in \mathcal{L})$	Virtual link capacity vector
$\hat{\boldsymbol{c}} = (\hat{c}_l, l \in \mathcal{L}) \in \bar{\Lambda}$	Maximum feasible rate vector
$\boldsymbol{y} = (y_l, l \in \mathcal{L})$	Aggregate flow rate vector
$\boldsymbol{x} = (x_s, s \in \mathcal{S})$	Flow sending rate vector
$\boldsymbol{\omega} = (\omega_l, l \in \mathcal{L})$	Cutoff frequency vector
$\boldsymbol{\tau}_b = (\tau_{l,s}, l \in \mathcal{L}, s \in \mathcal{S})$	Backward delay
$\boldsymbol{\tau}_f = (\tau_{s,l}, l \in \mathcal{L}, s \in \mathcal{S})$	Forward delay
$\boldsymbol{p} = (p_l, l \in \mathcal{L})$	Link congestion price vector

Table III.1: Notations

Problem Description

Next we discuss the challenge of congestion control algorithm under time-varying wireless channel when feedback delay is considered.

Congestion Control with Static Channel

The objective of congestion control is to maximize the aggregate utility of all flows across the network. Each flow s is associated with a utility function, such as $U_s(x_s) = \beta_s log(x_s)$, then the problem is formulated into a utility maximization framework.

$$W: \quad \max \quad \sum_{s\in\mathcal{S}} U_s(x_s) \tag{III.1}$$

$$\text{s.t.} \quad \sum_{s\in\mathcal{S}(l)} x_s \leq \hat{c}_l, \forall l \in \mathcal{L} \tag{III.2}$$

$$\text{over} \quad \hat{\boldsymbol{c}} \in \Lambda \tag{III.3}$$

Here (III.2) is the link capacity constraint, and (III.3) is the schedulability constraint. Problem W is well studied in existing works on wireless network congestion control.

Usually the distributed solution is derived from its dual decomposition. Let $\boldsymbol{p} = \{p_l, l \in \mathcal{L}\}$ be the Lagrange multipliers with respect to constraint (III.2). The dual of W is

$$\bar{D}(\boldsymbol{p}) = \min_{\boldsymbol{p}\geq 0} D(\boldsymbol{p}) \tag{III.4}$$

where

$$D(\boldsymbol{p}) = \max_{\boldsymbol{x},\hat{\boldsymbol{c}}} L(\boldsymbol{x},\boldsymbol{p},\hat{\boldsymbol{c}}) \tag{III.5}$$

$$= \max_{\boldsymbol{x}} \left\{ \sum_{s\in\mathcal{S}} \left(U_s(x_s) - x_s \sum_{l\in\mathcal{L}(s)} p_l \right) \right\} + \max_{\hat{\boldsymbol{c}}} \left\{ \sum_{l\in\mathcal{L}} p_l \hat{c}_l \right\} \tag{III.6}$$

From the dual decomposition we obtain the optimal solution $(\boldsymbol{x}^*, \hat{\boldsymbol{c}}^*)$, which should satisfy Eq.(III.7) and Eq.(III.8).

$$x_s^* = \arg\max_{x_s} \left\{ \sum_{s\in\mathcal{S}} \left(U_s(x_s) - x_s \sum_{l\in\mathcal{L}(s)} p_l \right) \right\} \tag{III.7}$$

$$\hat{c}_l^* = \arg\max_{\hat{c}_l\in\Lambda} \left(\sum_{l\in\mathcal{L}} p_l \hat{c}_l \right) \tag{III.8}$$

Here the multiplier p_l can be understood as the implicit congestion price [21] of link l, which represents the cost of delivering a unit of data through link l.

Eq.(III.7) can be solved by either adopting a dual controller or a primal-dual controller [46]. For instance, the primal-dual controller algorithm is described as

$$\dot{x_s}(t) = \lambda_s \left(1 - \frac{q_s(t)}{U_s'(x_s(t))} \right) \tag{III.9}$$

$$\dot{p_l}(t) = [\gamma_l(y_l(t) - \hat{c}_l)]_{p_l}^+ \tag{III.10}$$

The function $[f(x)]_x^+$ is defined as

$$f(x) = \begin{cases} f(x) & \text{if } x > 0 \\ max(f(x), 0) & \text{otherwise} \end{cases}$$

$q_s(t)$ is the aggregate congestion price along the path of flow s, and $y_l(t)$ is the aggregate flow rate over link l:

$$q_s(t) = \sum_{l \in \mathcal{L}(s)} p_l(t) \quad \text{(III.11)}$$

$$y_l(t) = \sum_{s \in \mathcal{S}(l)} x_s(t) \quad \text{(III.12)}$$

Eq.(III.9) implies that the rate of a flow is adjusted towards the direction of maximizing the profit, which is the utility minus the cost. Eq.(III.10) describes the demand-supply interaction of network resources: if the traffic demand over a link exceeds its maximum allowable rate, the link congestion price p_l will increase. The solution can be implemented in a distributed manner: each source node updates its sending rate according to Eq.(III.9) and each link updates its congestion price according to Eq.(III.10). Eq.(III.8) is essentially a link scheduling problem. A maximum weighted matching based scheduling policy [15] can be used to assure that the aggregate link weight $\sum_{l \in \mathcal{L}} p_l \hat{c}_l$ is maximized.

Congestion Control with Dynamic Channel

The channel capacity in W is regarded as static and the problem can be solved by the algorithm described by Eq.(III.8) - Eq.(III.10). However, the channel capacity in a real wireless environment can be highly volatile due to signal propagation effects, wireless interference, external noise, etc. [50].

Under time-varying channel $\boldsymbol{c}(t)$, the schedulable capacity region Λ is dynamic, denoted as $\Lambda(t)$. Under this context, the optimal sending rate of a flow s becomes time-varying, which is denoted by $x_s(t)$. A straightforward approach is to track channel dynamics and find the optimal solution for each time instance t. The time-varying rate allocation problem W_T is

$$W_T: \quad \max \quad \sum_{s\in\mathcal{S}} U_s(x_s(t)), \forall t \in T \tag{III.13}$$

$$\text{s.t.} \quad \sum_{s\in\mathcal{S}(l)} x_s(t) \leq \hat{c}_l(t), \forall l \in \mathcal{L} \tag{III.14}$$

$$\text{over} \quad \hat{\boldsymbol{c}}(t) \in \Lambda(t) \tag{III.15}$$

In order to do so, the link congestion price can be updated according to the instantaneous link capacity

$$\dot{p_l}(t) = [\gamma_l(y_l(t) - \hat{c}_l(t))]^+_{p_l} \tag{III.16}$$

Generally, time-varying solutions to W_T require that the network dynamics can be tracked timely so that the congestion control algorithm can make corresponding adjustment to achieve a utility-optimal solution under time-varying channels. For instance, the work of [15] presents an algorithm with time-varying channel modeled by an irreducible finite-state Markov chain. In [18], an algorithm is presented to deal with channel variations where the channel state follows a stationary distribution. However, these studies assume there is no feedback delay and rely on simplified wireless channel models. in real wireless environments the practical significance of the simplified models, such as a stationary Markov chain, is limited.

Congestion Control with Feedback Delay

In wireless networks, the delivery of the feedback information, such as congestion degree and link load, cannot be instantaneous due to the propagation delay, processing time, etc. The delay problem, coupled with time-varying channel, could greatly degrade the performance of a congestion control algorithm. Consider a single-flow-single-wireless-link scenario: let the link capacity of link l at time t be $c(t)$. The congestion price $p_l(t)$ from l the source node s may experience a delay τ^b, called the *backward delay*, so the price will be received by the source node at $t + \tau^b$. Based on this price, the source node adjusts the sending rate to $x_s(t + \tau^b)$, which will be detected by the link after a delay of τ^f, called the *forward delay*. When l makes price calculation according to the current capacity, which already becomes $c_l(t + \tau^b + \tau^f)$, the load it detects is still corresponding to the previous link capacity $c_l(t)$. Therefore a *mismatch* occurs.

In Fig. III.1 we plot the sending rate of a one-hop flow (between two Wi-Fi access points), which is generated from a cross-layer congestion control algorithm based on Eq.(III.8) - Eq.(III.10). The algorithm adjusts to time-varying wireless channels. The link capacity is derived from the Roofnet (IEEE 802.11b mesh network) trace [62]. The plots show that wireless link capacities fluctuate drastically over time. When there is no feedback delay, the sending rate can closely match the link capacity. But with the presence of feedback delay, the rate cannot match the capacity.

Solution Overview

To solve the problem of joint congestion control and scheduling in time-varying wireless environments with the presence of feedback delay, we present a robust joint control algorithm: *ROCS*. Fig. III.2 provides a high level overview of this approach,

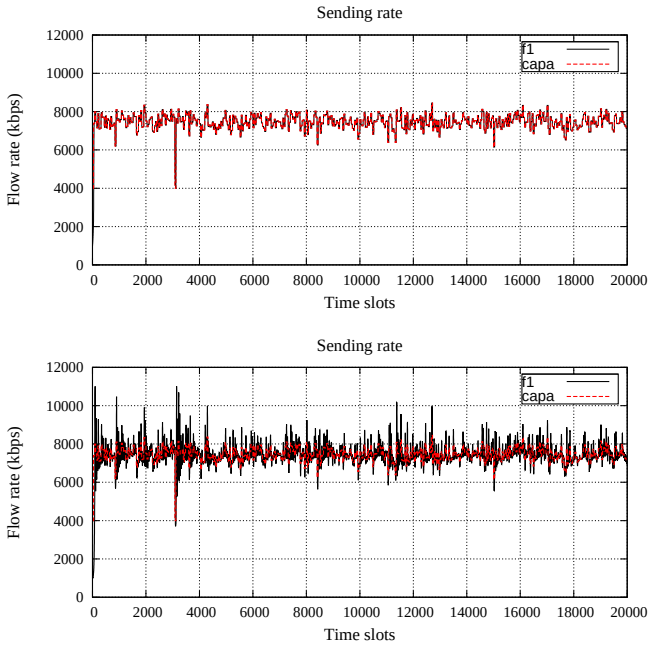


Figure III.1: Flow sending rate and link capacity of a single-hop network, with the presence and absence of delay (10ms, 10ms).

which consists of three key components: *Capacity Space Projection*, *Link Scheduling* and *Congestion Control*.

With the presence of feedback delay, the perturbation may lead to improper rate allocation, resulting in packet loss. Therefore, it is impractical to explicitly track the instantaneous signals. The central idea for addressing the problem of a dynamic channel with feedback delay is *Capacity Space Projection*. If we regard the capacity of a link as a signal, its perturbation is associated with its fast time scale component, which is hard to track and may lead to improper utilization with the presence of delay. Hence we employ a time decomposition approach: the slow time scale component $\tilde{c}_l(t)$ is directly applicable to the congestion control algorithm, while the fast time scale component is characterized by a *capacity margin* $\upsilon_l(t)$, which serves as a compensatory adjustment

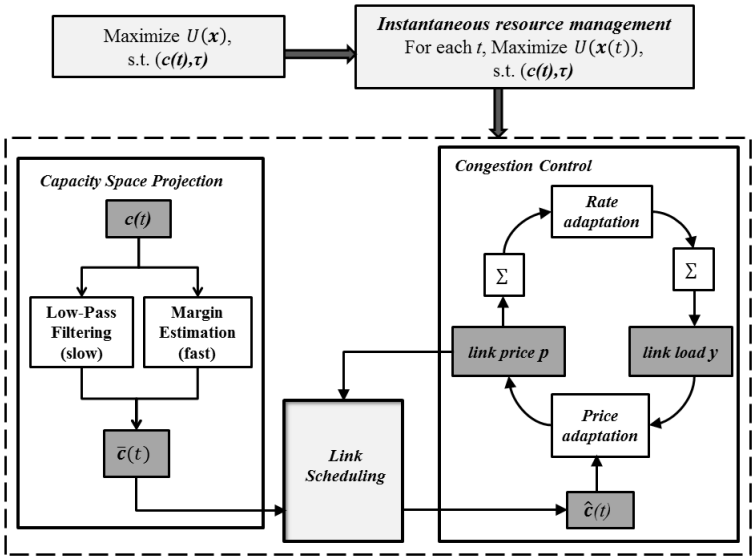


Figure III.2: Overview of *ROCS*.

to the slow time scale signal. The value of the margin variable needs to be deliberately selected to ensure the bandwidth resource can be fully exploited while suppressing the overshoot of sending rates. Based on the above discussion, the new link capacity variable $\bar{c}_l(t)$ is formulated as

$$\bar{c}_l(t) = \tilde{c}_l(t) + \upsilon_l(t) \tag{III.17}$$

Here $\tilde{c}_l(t)$ should vary sufficiently slowly. In other words, when a link adjusts its congestion price, the current capacity could be regarded as almost the same with the previous capacity. Therefore the delay effect can be reduced. Based on the new capacity space, now we can formulate a cross-layer congestion control problem W_T' over the new capacity space $\bar{\Lambda}(t)$ generated from *CSPA*, where $\bar{\Lambda}$ is more tractable than Λ.

$$\bar{W}_T^{'}: \quad \max \quad \sum_{s\in\mathcal{S}} U_s(x_s(t)) \tag{III.18}$$

$$\text{s.t.} \quad \sum_{s\in S(l)} x_s(t) \le \hat{c}_l(t), \forall l \in \mathcal{L} \tag{III.19}$$

$$\text{over} \quad \hat{\boldsymbol{c}}(t) \in \bar{\Lambda}(t), \tag{III.20}$$

$$\bar{\Lambda}(t) = CSPA(\boldsymbol{c}(t)) \tag{III.21}$$

$\bar{W}_T^{'}$ is solved by the joint *Link Scheduling* and *Congestion Control* algorithm which is very similar to the solution for Problem W.

Capacity Space Projection Algorithm

In this section, we present the *Capacity Space Projection Algorithm* (*CSPA*).

Capacity Space Projection

The *Capacity Space Projection Algorithm* (*CSPA*) is used to form a new convex capacity space as Eq.(III.17) shows, *CSPA* uses a time decomposition approach in which an original instantaneous link capacity signal is decomposed to a slow time scale signal and a margin value estimated from the fast time scale components. In this way the original capacity space $\Lambda(t)$ is projected to a new capacity space $\bar{\Lambda}(t)$, and the virtual channel capacity vector $\bar{\mathbf{c}}$ with this space is defined as $\bar{\mathbf{c}}(t) = \{\bar{c}_l(t), l \in \mathcal{L}\}, t \in T$.

Cutoff Frequency Search Algorithm

The central idea for identifying the slow time scale signal is low-pass filtering, so we need to determine the cut-off frequencies of the low-pass filters. Let the sampled link

capacity series be $C_l = \{c_l(0), c_l(1), c_l(2), ...\}$, and the capacity time series generated by *CSPA* be $\tilde{C}_l = \{\tilde{c}_l(0), \tilde{c}_l(1), \tilde{c}_l(2), ...\}$. Let vector ω_l denote the cut-off frequency of l. The low-pass filtering process is:

$$\tilde{C}_l = C_l * g_{\omega_l} \tag{III.22}$$

Here g_{ω_l} is the impulse response function of the low pass filter with cutoff frequency ω_l, and the symbol $*$ represents the convolution operation. The capacity signal after low pass filtering is usually less fluctuating than the original capacity. Fig. III.3 shows the autocorrelation function of the original link capacity series and the filtered link capacity series of a wireless link. Obviously when the time τ_l lag is below a certain value, the correlation between $\tilde{C}_l^t$ and $\tilde{C}_l^{t-\tau_l}$ of the low-pass filtered series is significantly higher than the original series. Hence even if a limited feedback delay is introduced to the network, the channel condition does not change sharply. The link capacity of l at the moment when the link senses the new rate allocation still resembles the link capacity when the source nodes adjust the sending rates. As a result, the delay effect is effectively reduced.

We propose a *Static Cutoff Frequency Search Algorithm* (*SCFS*) to find the optimal cutoff frequency of each link. The goal of this algorithm is: a) The capacity signal should be preserved as much as possible. Filtering a signal via a low-pass filter drops the high frequency components and entails information loss. b) The autocorrelation of the filtered link capacity series should be no less than a threshold. This implies the two time series $\tilde{C}_l^t$ and $\tilde{C}_l^{t-\tau_l}$ are highly correlated so that the delay effect can be mitigated.

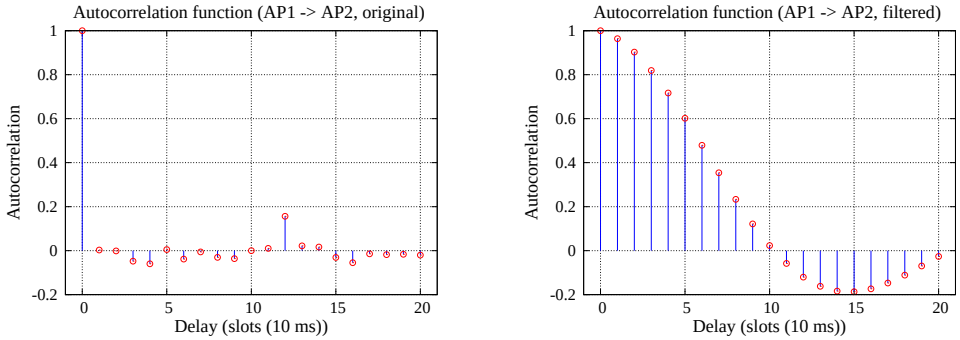


Figure III.3: Autocorrelation functions of the original link capacity series and low-pass filtered link capacity series.

To achieve a), we define a preservation degree threshold $\bar{\eta}_l$ to avoid unnecessary information loss, when searching the optimal cutoff frequency. To achieve b), we define a target correlation degree $\bar{r}_l$.

The algorithm is listed in Tab. III.2. Here $Fr(\cdot)$ is a function that returns the upper bound of the frequency spectrum of a capacity series, and $corr(\cdot)$ is a function that computes the autocorrelation of the filtered capacity series. *SCFS* requires a link capacity series C_l^t as input. The signal preservation degree is the ratio of the energy of the low-pass filtered series and the energy of the original series, which is an indicator of the information loss. We use a function $pv(\cdot)$ to compute the signal preservation degree, defined as

$$pv(X, Y) = \frac{E(X)}{E(Y)} \tag{III.23}$$

Here E represents the energy of a capacity series.

SCFS is an exhaustive search algorithm. It tries to preserve the information of the original capacity and meanwhile iteratively searches the optimal frequency. The algorithm starts with a tight preservation degree constraint. If no feasible solution exists with the current preservation degree, it gradually relaxes the preservation degree constraint

and repeating the search process. The algorithm satisfies requirement a) by guaranteeing the preservation degree will not fall below the threshold $\bar{\eta}_l$. It satisfies requirement b) by searching a cutoff frequency that can achieve the maximum autocorrelation degree.

***SCFS*: Static Cutoff Frequency Search Algorithm**

For each link l in $\mathcal{L}$:
Input: $C_l^t, \bar{r}_l, \bar{\eta}_l, \tau_l$
Output: ω_l^*.

1. $\omega_l^* \leftarrow F_r(C_l^t)$
2. $\omega^l \leftarrow \omega_l^*$
3. $\bar{\eta}_l \leftarrow 1$
4. **if** $(corr(\tilde{C}_{l,t}, \tilde{C}_l^{t-\tau_l}) \geq \bar{r}_l)$
5. **return** ω_l^*
6. **end if**
7. **while** $(\bar{\eta}_l > \bar{\eta})$
8. $\omega_l \leftarrow \omega_l - \epsilon$
9. $\tilde{C}_l^t \leftarrow C_l^t * g_{\omega_l}$
10. $\eta_l \leftarrow p(\tilde{C}_l^t, C_l^t)$
11. **if** $(corr(\tilde{C}_{l,t}, \tilde{C}_l^{t-\tau_l}) \geq \bar{r}_l)$
12. $\omega_l^* \leftarrow \omega_l$
13. **return** ω_l^*
14. **end if**
15. **end while**
16. **return** ω_l^*

Table III.2: Cutoff Frequency Search Algorithm (Static)

Capacity Margin Estimation

Now we present the fast time scale component modeling of via *capacity margin* estimation. We develop a model to estimate the fast time scale signal of the capacity. More specifically, we introduce a variable called *capacity margin* υ_l for link l to characterize the fast time scale signal.

In order to find the value of υ_l, we look for compensation for packet loss: the loss incurred by overshooting will impose penalty on rate allocation. To this end, we define a penalty function $P_l(y_l^p)$ for link l, which is a convex function Here y_l^p is the amount that the virtual capacity of l exceeds the instantaneous capacity, i.e.

$$\begin{aligned} y_l^p(\upsilon_l, t) &= [\bar{c}_l(t) - c_l(t)]^+ \\ &= [\tilde{c}_l(t) + \upsilon_l - c_l(t)]^+ \end{aligned} \tag{III.24}$$

Now the objective of margin value selection is to maximize the profit across the network, which is the optimal network utility minus the aggregate penalty of all the links. The optimal network utility is obtained under the optimal rate allocation $x^*(t)$, which is the solution to W_T'. Note that in the formulation of W_T', the link capacity is defined by Eq. (III.17). Suppose the capacity margin vector is $\boldsymbol{\upsilon}$, The problem is formulated as

$$\begin{aligned} &\Phi(\boldsymbol{\upsilon}, t) \\ &= \max_{\boldsymbol{\upsilon}} \left(\sum_{s \in \mathcal{S}} U_s(x_s(\boldsymbol{\upsilon}, t)) - \sum_{l \in \mathcal{L}} P(y_l^p(\upsilon_l, t)) \right) \\ &= \max_{\boldsymbol{\upsilon}} \left(\sum_{s \in \mathcal{S}} U_s(x_s(\boldsymbol{\upsilon}, t)) - \sum_{l \in \mathcal{L}} P([\tilde{c}_l(t) + \upsilon_l - c_l(t)]^+) \right) \end{aligned} \tag{III.25}$$

where the rate allocation $\boldsymbol{x}(\boldsymbol{\upsilon}, t)$ is made by assuming that the capacity space $\Lambda(\boldsymbol{\upsilon}, t)$ is defined such that the usable link capacity $c_l^u(\upsilon_l, t)$ is

$$c_l^u(\upsilon_l, t) = \tilde{c}_l(t) + \upsilon_l \tag{III.26}$$

We propose a margin update algorithm find the margin vector $\boldsymbol{\upsilon}$. Suppose the upper bound of queue loss is $\bar{L}$. We introduce a margin update cycle T_υ The links keep monitoring their queue loss values and locally update their margin values at the beginning of a cycle. The algorithm is presented in Tab. III.3. If the queue loss of a link exceeds the upper bound, the margin will increase according to the gap between the average queue loss during the previous cycle and the upper bound. Otherwise, the margin will decrease if the queue loss is less than the bound. ϵ_l and ϕ_l are parameters for margin adjustment.

Margin Update Algorithm
At the beginning of T_υ, link l performs:
if $L_l \geq \bar{L}$
$\quad \upsilon_l \leftarrow \upsilon_l + \epsilon_l(L_l - \bar{L}_l)$
else if $L_l \leq \bar{L}$
$\quad \upsilon_l \leftarrow \upsilon_l - \phi_l(\bar{L}_l - L_l)$
end if

Table III.3: Margin Update Algorithm

CSPA Implementation

Based on the discussions of slow time scale capacity extraction and capacity margin estimation, we now present the implementation of *CSPA* in Tab. III.4. The input of *CSPA* includes $\boldsymbol{\omega}$ and T_υ. Here $\boldsymbol{\omega}$ is the optimal cut-off frequency vector generated by *SCFS* algorithm, and T_υ is the period of capacity margin update. Each link autonomously implements *CSPA* at run time and generates its virtual capacity. At the beginning of a time slot, a link l first monitors the channel condition to obtain the original capacity, then it updates its capacity time series $\bar{C}_{l,t}$ which is used to compute the slow time scales signal. Then the current virtual link capacity is computed using Eq.(III.22).

***CSPA*: Capacity Space Projection Algorithm**
Each active link l in $\mathcal{N}$ performs: **Input**: ω_l, T_h **Output**: $\bar{c}_l(t)$. 1. **if** $t \leftarrow T_h$ 2. Update υ_l 3. **end if** 4. Monitor current channel state and update $C_{l,t}$ 5. Filter $C_{l,t}$ and obtain the slow time scale signal $c_l^{'}(t)$ 6. Update virtual capacity series: 7. $\bar{c}_l(t) \leftarrow \tilde{c}_l(t) + \upsilon_l$ 8. return the current virtual link capacity $\bar{c}_l(t)$

Table III.4: *CSPA* implementation

Joint Congestion Control and Scheduling

In this section we introduce the joint congestion control and scheduling algorithm and its distributed implementation.

Robust Congestion Control Algorithm

Since *CSPA* can effectively mitigate the *forward delay* effect (it also reduces the *backward delay* effect) incurred by flow rate information delay and channel perturbation, we ignore the *forward delay* in this section and only consider the *backward delay*. Then the joint control algorithm J is listed as follows.

$$\dot{x}_s(t) = \lambda_s \left(1 - \frac{q_s(t)}{U_s'(x_s(t))}\right) \tag{III.27}$$

$$\dot{p}_l(t) = [\gamma_l(y_l(t) - \hat{c}_l(t))]_{p_l}^{+} \tag{III.28}$$

$$\hat{\boldsymbol{c}}(t) = \arg \max_{\hat{\mathbf{c}}(t) \in \Lambda(t)} \left(\sum_{l \in \mathcal{L}} p_l(t) \hat{c}_l(t)\right) \tag{III.29}$$

The algorithm jointly adjusts flow rate allocation and link utilization, in order to achieve a global optimum, where the aggregate network utility is maximized. The two components of the algorithm, congestion control (Eq.(III.27) and Eq.(III.28)) and scheduling (Eq.(III.29)), implicitly interact with each other: flow rate allocation determines link congestion prices and therefore affects the scheduling on link selection which is based on link weight $p_l(t)\bar{c}_l(t)$. Meanwhile, scheduling determines the effective link capacities, and hence affects flow rate allocation.

Stability Analysis

The stability of a delayed system can be studied either through Laplace transform or Lyapunov function based approaches. Because the dynamics caused by scheduling is not explicitly reflected in the frequency domain, we employ a time domain oriented approach: Razumikhin theorem [63], to investigate the sufficient conditions for the Lyapunov stability (local stability) of the control system (III.27) - (III.29). First, we assume the link capacity vector is fixed during an interval $[t, t + \Delta t]$, considering the capacity space varies at a sufficiently slow time-scale. We denote the current capacity allocation vector as $\hat{\boldsymbol{c}}$. Second, We assume the following condition holds for any flow congestion price $q_r^d(t)$. This implies the trajectories of the delayed signals are bounded.

$$\sup ||q_s^d(t) - q_s(t)||_2 = \zeta_s, \forall s \in \mathcal{S} \tag{III.30}$$

Let the equilibrium point corresponding to the channel state $\hat{\boldsymbol{c}}$ be $(\boldsymbol{x}^*, \boldsymbol{p}^*)$, where $\boldsymbol{x}^*$ and $\boldsymbol{p}^*$ are equilibrium rate vector and price vector respectively. We consider the following positive definite and radially unbounded Lyapunov-Razumikhin function [64]:

$$V(\boldsymbol{x},\boldsymbol{p};\boldsymbol{x}^*,\boldsymbol{p}^*)=\sum_{s\in\mathcal{S}}\frac{(x_s(t)-{x_s}^*)^2}{2\lambda_s}+\sum_{l\in\mathcal{L}}\frac{(p_l(t)-{p_l}^*)^2}{2\gamma_l} \tag{III.31}$$

The time derivative of V is

$$\begin{aligned}
\dot{V}(t) &= \sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)\left[U_s'(x_s)-q_s^d(t)\right] \\
&\quad +\sum_{l\in\mathcal{L}}(p_l(t)-p_l^*)\left[y_l(t)-\hat{c}_l(t)\right]^+ && \text{(III.32)} \\
&\leq \sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)\left[U_s'(x_s)-q_s^d(t)\right] \\
&\quad +\sum_{l\in\mathcal{L}}(p_l(t)-p_l^*)\left[y_l(t)-\hat{c}_l(t)\right] && \text{(III.33)} \\
&= \sum_{s\in\mathcal{S}}\left[(x_s(t)-x_s^*)(U_s'(x_s)-U_s'(x_s^*)+q_s^*-q_s^d(t))\right] \\
&\quad +\sum_{l\in\mathcal{L}}\left[(p_l(t)-p_l^*)\left(y_l(t)+y_l^*-y_l^*-\hat{c}_l(t)\right)\right] && \text{(III.34)} \\
&= \sum_{s\in\mathcal{S}}\underbrace{(x_s(t)-x_s^*)(U_s'(x_s)-U_s'(x_s^*))}_{(a)} \\
&\quad +\sum_{s\in\mathcal{S}}\underbrace{(x_s(t)-x_s^*)(q_s^*-q_s^d(t))}_{(b)} \\
&\quad +\sum_{l\in\mathcal{L}}\underbrace{(p_l(t)-p_l^*)\left(y_l(t)-y_l^*\right)}_{(c)} \\
&\quad +\underbrace{\sum_{l\in\mathcal{L}}(p_l(t)-p_l^*)\left(y_l^*-\hat{c}_l(t)\right)}_{(d)} && \text{(III.35)}
\end{aligned}$$

Note that (III.32) and (III.33) are equal if $\gamma(y_l(t)-\hat{c}_l(t))$ is non-negative. The second expression in (III.33) is positive because $(p_l(t)-p_l^*)$ is also negative when $\gamma(y_l(t)-\hat{c}_l(t))$ is negative. (III.34) follows that $U_s'(x_s^*)=q_s^*$. If there are no delays, term (b) and term (c) are canceled out. According to the concavity of the utility function, $(a)\leq 0$. In addition, by the scheduling policy (III.8), we have:

$$\sum_{l\in\mathcal{L}} p_l^* \hat{c}_l(t) \leq \sum_{l\in\mathcal{L}} p_l^* y_l^* \tag{III.36}$$

$$\sum_{l\in\mathcal{L}} p_l(t) y_l^* \leq \sum_{l\in\mathcal{L}} p_l(t) \hat{c}_l(t) \tag{III.37}$$

Now denote term (d) by S, then (III.36) and (III.37) lead to

$$\begin{aligned} S = & \sum_{l\in\mathcal{L}} p_l^* \hat{c}_l(t) - \sum_{l\in\mathcal{L}} p_l^* y_l^* \\ & + \sum_{l\in\mathcal{L}} p_l(t) y_l^* - \sum_{l\in\mathcal{L}} p_l(t) \hat{c}_l(t) \leq 0 \end{aligned} \tag{III.38}$$

If the delay is negligible, the time derivative of the Lyapunov function is not positive. Therefore the proof of Lyapunov stability without delay is completed. However, the above analysis is not sufficient for a system with delay.

Now we proceed to study the stability with delay. More specifically, we try to find the sufficient conditions for the Lyapunov stability of the system. Since $(d) \leq 0$, the time derivative of V satisfies the following inequality:

$$
\begin{aligned}
\dot{V}(t) \;\leq\; & \sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)\left[U_s'(x_s(t))-U_s'(x_s^*)\right] \\
& +\sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)(q_s^*-q_s^d(t)) \\
& +\sum_{l\in\mathcal{L}}(p_l(t)-p_l^*)\left(y_l(t)-y_l^*\right) \\
= \; & \sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)(U_s'(x_s(t))-q_s^d(t)) \\
& +\sum_{l\in\mathcal{L}}(p_l(t)-p_l^*)\left(y_l(t)-y_l^*\right) \\
= \; & \sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)(U_s'(x_s(t))-q_s^d(t)) \\
& +\sum_{s\in\mathcal{S}}(x_s(t)-x_s^*)\left(q_s(t)-q_s^*\right)
\end{aligned}
\tag{III.39}
$$

Hence we have

$$
\dot{V}(t)\leq Q(t) \tag{III.40}
$$

where

$$
Q(t)=\sum_{s\in\mathcal{S}}(x_s^*-x_s(t))[(q_s^d(t)-q_s(t))-(U_s'(x_s(t))-q_s^*)] \tag{III.41}
$$

Next we investigate the property of $Q(t)$. From the concavity of $U(x_s)$, the following inequality holds,

$$
(x_s^*-x_s(t))(U_s'(x_s)-q_s^*)\geq 0 \tag{III.42}
$$

We further consider the following two situations:

1. if $(x_s^* - x_s(t)) \geq 0$, then $(U_s'(x_s(t)) - q_s^*) \geq 0$,

   hence the sufficient condition for $Q(t) \leq 0$ is

$$|q_s^d(t) - q_s(t)| \leq |U_s'(x_s(t)) - q_s^*|, \forall s \in \mathcal{S} \tag{III.43}$$

2. if $(x_s^* - x_s(t)) < 0$, then $(U_s'(x_s(t)) - q_s^*) < 0$,

   hence the sufficient condition for $Q(t) \leq 0$ is

$$|q_s^d(t) - q_s(t)| \leq |q_s^* - U_s'(x_s(t))|, \forall s \in \mathcal{S} \tag{III.44}$$

According to condition (III.30), one sufficient condition for $\dot{V}(t) \leq 0$ is

$$\zeta_s \leq ||U_s'(x_s(t)) - q_s^*||_2, \forall s \in \mathcal{S} \tag{III.45}$$

Since J is a primal-dual controller, condition (III.45) can be guaranteed by manipulating the tuning parameters: λ_s and γ_l. Based on the discussions above we have the following theorems.

Theorem 1: The joint congestion control and scheduling algorithm J is Lyapunov stable without time delay.

Theorem 2: The joint congestion control and scheduling algorithm J is Lyapunov stable with time delay if (III.45) holds.

Note that (III.45) gives the sufficient conditions, and the algorithm J may also be stable if the conditions are not satisfied. Moreover, if the delay effect is dominant (ζ_s is sufficiently large), the condition cannot be satisfied by simply manipulating the tuning parameters, therefore, a feasibility check is required at the beginning of the algorithm.

Distributed Implementation of *ROCS*

The implementation of cross-layer congestion control has been studied in existing works, such as [15, 18]. Our cross-layer design differs from these works in that we employ a capacity space projection algorithm, such that the cross-layer control algorithm runs over a newly built capacity space.

Since a link capacity series usually possesses some unchanging statistical characteristics, these characteristics can be profiled from a sample sequence. At the beginning of *ROCS*, we profile the link capacities and obtain $\boldsymbol{\omega}$ by using *SCFS*. As we discussed in Sec. III, the cutoff frequency vector can also be obtained online: the time series of the sampled link capacity is periodically updated and fed into *SCFS*. When *ROCS* is running, the link states are monitored and the channel decomposition algorithm is applied to generate the new capacity space.

The scheduling policy should maximize the aggregate link weight $\sum_{l\in\mathcal{L}} p_l\hat{c}_l$ across the network. In our implementation, this is achieved by using a maximal matching scheduling, as introduced in [15]. However, one remarkable difference between our scheduling algorithm and the scheduling algorithm used in [15] is that we use a virtual capacity, consisting of the slow time scale component and an estimated component, instead of using the original capacity directly. As a result, our design is capable of not only adjusting rate allocation under time-varying channels, but also handling delay effects. The distributed implementation of the algorithm is presented in Tab. III.5.

Robust Joint Congestion Control and Scheduling (*ROCS*)

0) *Perform SCFS* to obtain $\boldsymbol{\omega}$

1) *Feasibility check*

if this is a feasible problem,

Start timer, $t \leftarrow 0$, repeat 2)-7) on a slotted time basis.

else exit.

2) Collect channel states at each time slot.

Perform *CSPA* to obtain $\bar{\mathbf{c}}$. $t \leftarrow t + 1$.

3) **if** $t == T$

stop timer and go to 4).

else go to 2).

4) *Link congestion price update*

Link l updates its link congestion price

$$\dot{p}_l(t) \leftarrow [\gamma_l(y_l(t) - \hat{c}_l(t))]_{p_l}^+$$

Add price $p_l(t)$ to the flow price q_s, if s travels through l.

5) *Rate adaption*

Each source node adjusts its sending data rate when it receives an ack packet from the destination node:

$$\dot{x}_s(t) \leftarrow \lambda_s \left(1 - \frac{q_s^d(t)}{U_s'(x_s)}\right)$$

6) *Scheduling*

For each node s, start the scheduling timer.

6.1) **if** s has been scheduled by any of its neighbor node,

send messages to notify all nodes in its interference set and stop scheduling procedure.

6.2) Each incident directional link l is assigned a weight

$$w_l \leftarrow p_l \bar{c}_l$$

6.3) Find a link with the maximum weight.

if the link is found,

Schedule this link and update the effective capacity $\hat{c}_l$.

Notify all neighbors and all links in the interference set.

else Stop scheduling procedure.

6.4) **if** timeout occurs,

stop scheduling.

else repeat 6.1) to 6.4).

7) Restart timer, $t \leftarrow 0$, and go to 2)

Table III.5: Distributed implementation of *ROCS*

Numerical Results

In this section we evaluate the performance of our robust joint congestion control and scheduling algorithm.

Experimental Setup

We simulate three wireless mesh network scenarios: a) a single-flow-single-link network; b) a multi-hop network with parallel flows and 3) a multi-hop network with overlapped flows, as shown in Fig. III.4. Data packets are delivered via wired connections from remote sources to the stationary local wireless routers, and then forwarded to the destination nodes in the wireless networks. We define utility function $U_s(x_s) = \beta_s log x_s$. The time-varying capacities of the links are modeled by two types of wireless network traces: one is a 4-state Markov chain trace with uniform inter-state transition probability. During each state, a link capacity can be one of the following values: $6Mbps, 7Mbps, 8Mbps$ and $9Mbps$. The other one is the Roofnet trace [62]. In our simulation, each link in the experimental networks is randomly assigned a link capacity series from the Markov trace the Roofnet trace. The capacity signal sampling interval is 40 milliseconds, more specifically, the capacity of a link is updated every 40 milliseconds. We do not simulate retransmissions. Note that hereinafter a delay value τ implies that the forward delay τ_f and the backward delay τ_b are both equal to τ.

To evaluate the performance of *ROCS*, we compare it with two baseline algorithms:

- *Baseline algorithm I* is a joint congestion control and scheduling algorithm, which is called *Alg.BASE*. It is based on the cross-layer congestion control algorithm introduced in [15],where we remove the routing component. It monitors the channel states and adjusts flow rate accordingly.

[1] In our simulation, the lost packets are not retransmitted. Sending rate = Receiving rate + Loss rate.

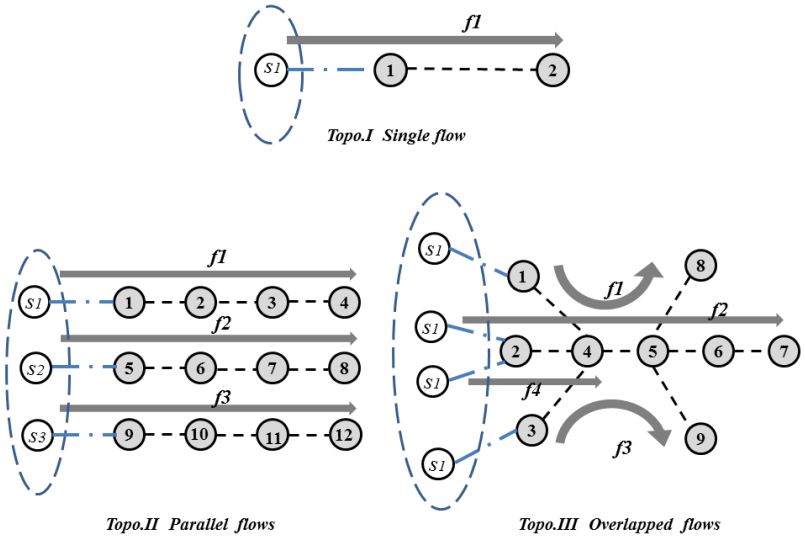


Figure III.4: Experimental multi-hop wireless networks with parallel data flows (left) and overlapped data flows (right).

- *Baseline algorithm II* is a joint congestion control and scheduling algorithm similar to Baseline algorithm I, called *Alg.AVER*. The major difference is that it uses the average link capacity evaluated from the the sampled capacity signal series, instead of the original capacity.

Performance metrics used in our experiments include: 1) flow rate (sending rate), 2) network queue length and 3) loss ratio.

Sending Rate of Single-Hop Networks

We start from investigating the rate allocation over a single-hop wireless network (Topo.I). The link capacity is modeled by the RoofNet trace, and the forward and backward delays are both set to be 10 ms. Fig. III.5 shows the sending rate generated by *ROCS*. The rates vary slowly compared with the rates in Fig. III.1.

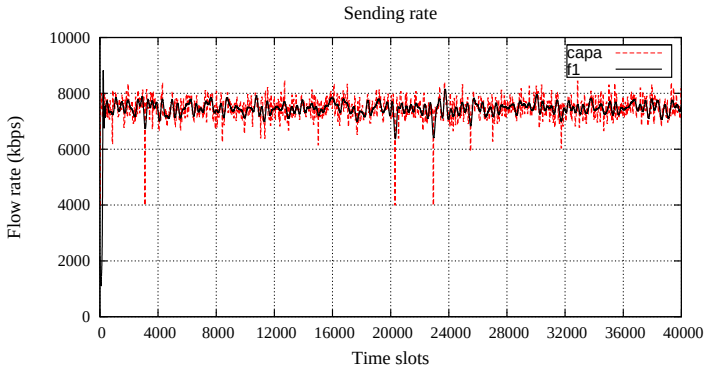


Figure III.5: Sending rate under ROCS of a single link (Topo.I) with delay (10ms, 10ms).

Sending Rates of Multi-Hop Networks

In this experiment we examine the sending rate allocation in the multi-hop networks. The forward delay and the backward delay are both 20 ms. Fig. III.6 shows the flow rates generated by *ROCS*. The instantaneous flow rates are fluctuating, because the link capacities are time-varying. According to our observation, the flow rates generated by *ROCS* can converge rapidly either under the simulated Markov channel or the RoofNet channel.

Queue Stability

To investigate the stability of our algorithm, we evaluate the sum of the aggregate queue length of the network (the network queue length) over time. A bounded network queue length implies the system is stable. Here we assume the queue size at each node is unbounded. Fig. III.7 shows the instantaneous network queue length of each scenario. From the plots we can see that the queue lengths are bounded. However, if the maximum queue size is bounded, packet loss may occur due to queue overflow. We will test the performance of *ROCS* under this context via a group of experiments in Subsection III.

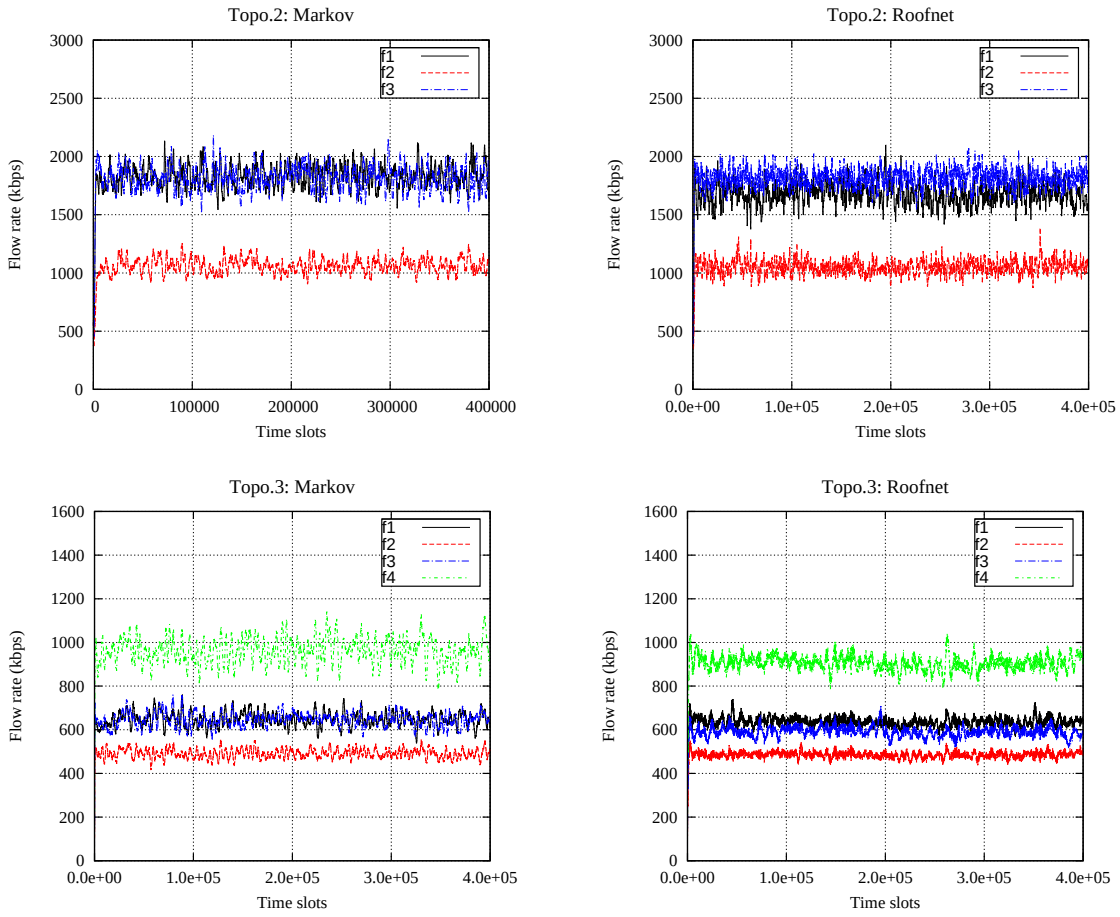


Figure III.6: Sending rates of *ROCS*.

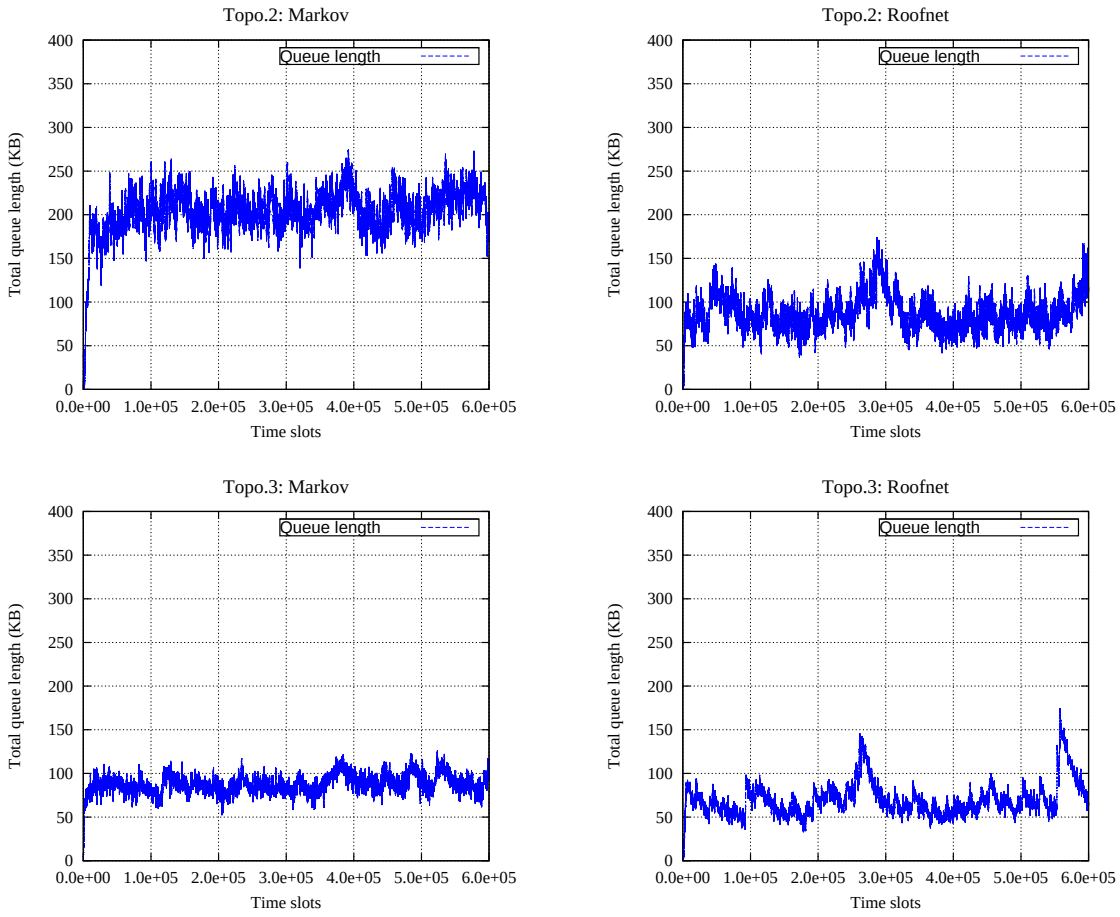


Figure III.7: The total instantaneous queue length.

Loss Ratio

In order to evaluate the efficiency of *ROCS* in packet delivery, we introduce *loss ratio*, which is defined as *the number of packets lost* divided by *the number of packets delivered*. This metric can be interpreted as the quantity of packets lost for each successfully delivered packet. We conduct two experiments: a) we change the delay values and plot the corresponding loss ratios; b) we change the maximum queue size and plot the corresponding loss ratios. For each network scenario, each simulation instance (corresponding to either a particular delay value or a particular maximum queue length) runs for 320 seconds.

We first fix the queue length bound to be 1000 and observe the loss ratios. Fig. III.8 shows the results across different delay values. Obviously the loss ratio of *ROCS* is the lowest among all the three algorithms. We observe an increasing loss ratio when the delay value increases under *Alg.BASE*. This is probably because a larger delay within this range imposes a more severe impact on the congestion control algorithm. On the other hand, the loss ratio under *Alg.AVER* is relatively stable. This is because the capacity used by *Alg.AVER* is an estimated average value. Therefore the impact of channel variation is limited. We plot the queue losses in Fig. III.9. The results indicate that the loss under *ROCS* is less than the other two algorithm.

Next we observe the loss ratios across various queue length bounds, which are shown in Fig. III.10. Like we expect, the loss ratios decrease while the maximum queue length increases. Again the loss ratio under *ROCS* is the lowest among all the algorithms.

To sum up, the robust congestion control algorithm *ROCS* conservatively exploits feasible capacity under channel perturbation, while avoiding queue overflows. It can effectively reduce data loss without jeopardizing network throughput. In real world

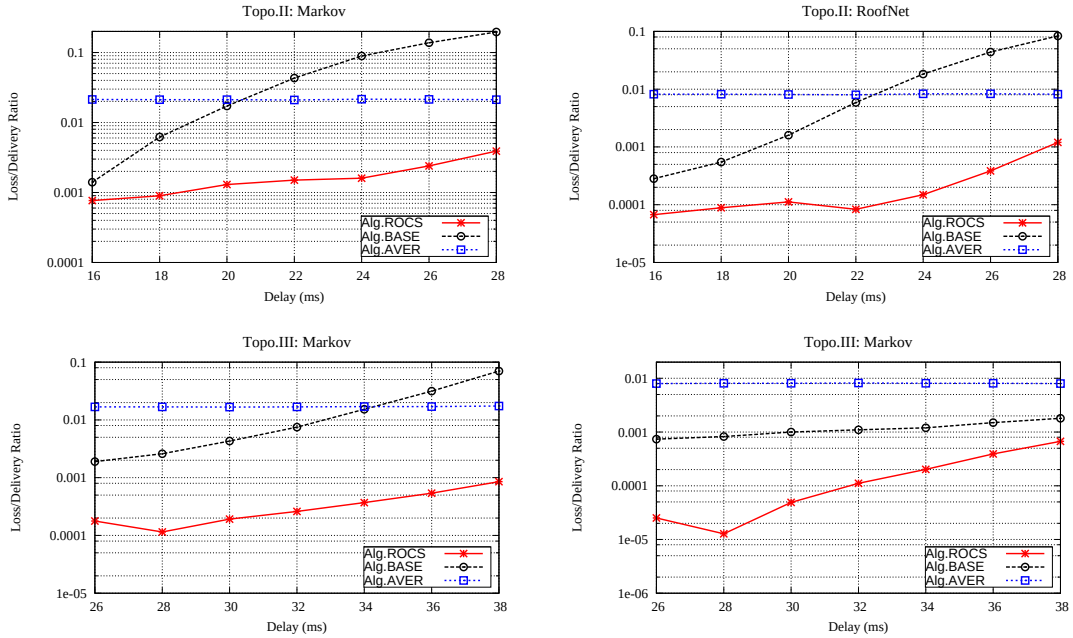


Figure III.8: Loss ratio (under different delays).

applications, queue overflows lead to packet losses, and packet losses incur retransmissions, which increase communication overhead. Therefore reducing occurrence of queue overflows carries significance in wireless communications.

Discussion

In time-varying multi-hop wireless networks, the performance of joint congestion control and link scheduling algorithms usually suffer from time delay and channel perturbations. The solution proposed in this chapter reshapes the wireless link capacity space, so the virtual capacity used by the congestion control algorithm mainly follows the slow time scale of the real capacity. By using this technique, the congestion control algorithm is more robust with the presence of high frequency channel capacity variations. At the same time, the delay effect is remarkably reduced. We implement the algorithm in a fully distributed way, so the nodes in a wireless network can autonomously

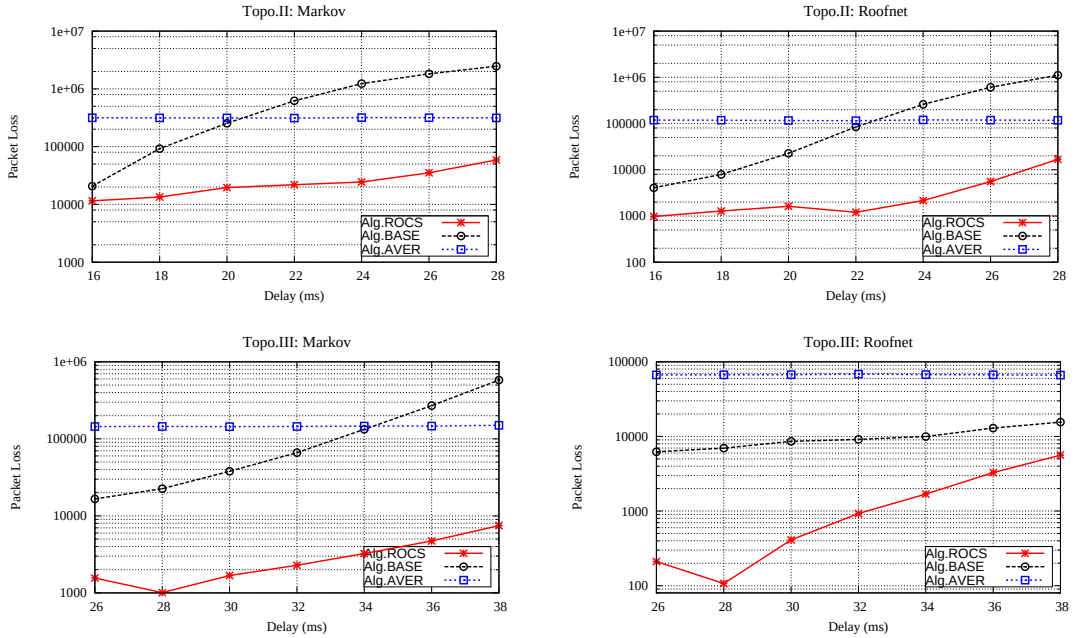


Figure III.9: Data loss due to queue overflow.

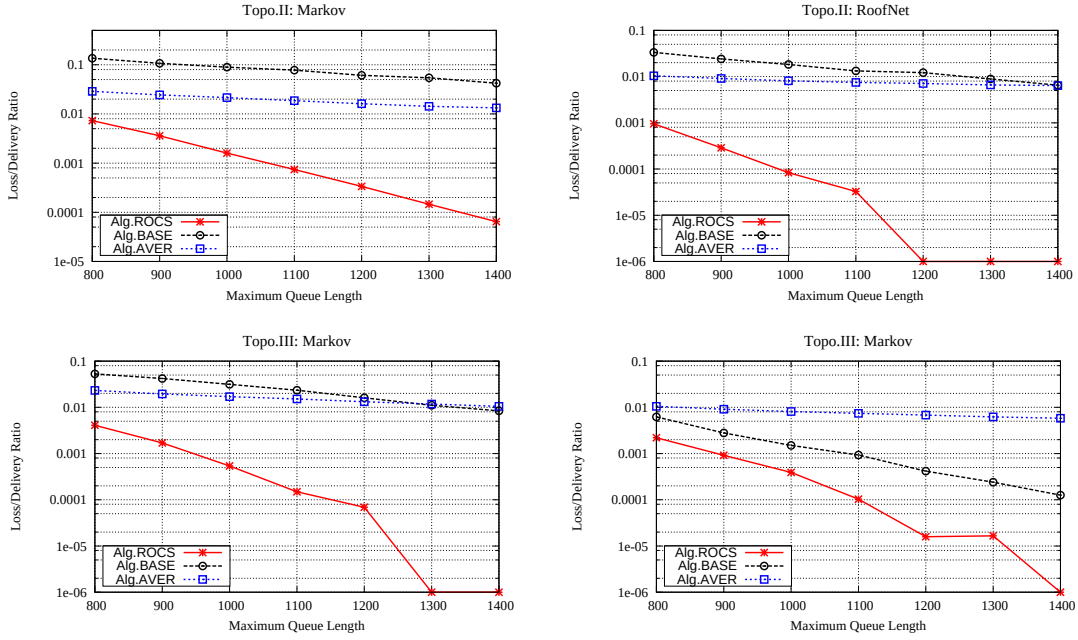


Figure III.10: Loss ratio (under different queue length limits).

collect the network information and make corresponding adjustment to the resource allocation. Experimental results demonstrate the robustness and efficiency of our algorithm. The study is especially meaningful for some data usage critical applications, such as mobile phone applications.

CHAPTER IV

A TIME-SCALE DECOMPOSITION APPROACH TO JOINT CONGESTION CONTROL AND SCHEDULING

In this chapter we present a time-scale decomposition approach to joint congestion control and link scheduling for resource management in time-varying wireless networks with feedback delays. We formulate this problem into an optimization framework constrained by the feasible capacity space. The objective of the optimization problem is to maximize the time integral of the aggregate substantial network welfare. The proposed solution is based on a time scale decomposition module, through which the congestion control algorithm is operated over a new capacity vector that follows the critical time scale. We further propose a penalty-aware scheduling mechanism to track the level of rate over-provisioning at each link. The simulation results show that the proposed approach achieves stable rate allocation and greatly reduces over-utilization.

The rest of this chapter is organized as follows. We first overview the proposed solution. Next we introduce the system model and the problem description. In the following two sections we propose our dynamic-aware congestion control algorithm and the simulation study. Finally we conclude the chapter.

Overview

The wireless network resource allocation has been a long-standing problem. It is recently ignited by the accelerated proliferation of smart devices, which have created increasing demand for wireless data services and resulted in network congestion problem. The inherent nature of wireless interference has motivated joint congestion control and scheduling as a solution to resource allocation in wireless networks [15, 16, 17, 18,

19, 37] and [38], where the global utility maximization problem is decomposed into three subproblems at different layers of the network stack - rate adaptation at transport layer, congestion degree generation at network layer and scheduling at MAC layer. For example, the congestion control algorithms are usually combined with the queue length based scheduling algorithms to achieve throughput optimality and fairness [38, 65, 66]. The analytical properties (e.g., stability) of these solutions studies strongly rely on two assumptions: 1) feedback delay in the network is negligible and 2) the wireless channel capacity is either static or its dynamics follow some simple models such as a Markov chain [15]. However, neither of the assumptions is close to real wireless environments. Feedback delays exist in real world wireless networks which have time-varying channel condition.

To address this problem, we present a time-scale decomposition approach to joint congestion control and scheduling for time-varying wireless networks. Our primary objective is to ensure the stability of congestion control while effectively utilizing the time-varying channel resource. We employ a new utility optimization framework, which relaxes the original framework from two aspects. 1) Its solution for rate allocation does not necessarily guarantee the aggregated utility of all flows to be optimal at each time instance, rather it targets at optimizing the aggregated utility over a certain time span. 2) It allows for packet loss in the optimal solution and imposes a corresponding penalty. This is in contrast with the original formulation where resource capacity imposes a hard constraint that ensures no packet loss at the system equilibrium. To estimate the potential packet loss, we assign a quantitative variable to each link. Specifically, each link is associated with a penalty function which evaluates its cost of capacity over-utilization. The optimization objective is to maximize the time-aggregated network net profit which is the difference between the aggregate utility and penalty. To reduce delay effect, we decompose the problem into two components at two different time scales by introducing

a virtual *slow time scale capacity* for each link. The congestion control component only observes the slow time scale link capacity, and as a result executes on a slow time scale. The scheduling component observes the real channel capacity and schedules the links at fast time scale. The critical time scale of the slow time scale capacity is used to ensure the stability of congestion control in the presence of network feedback delay, while the scheduling component reconciles the difference between the scheduled feasible link rate and the virtual slow time scale link capacity to minimize the penalty of capacity over-utilization. The implementation of the algorithm is fully distributed. Each computational unit involved in the cross-layer design autonomously updates its local status or resource allocation information by using the required feedback information.

System Model

We consider a multi-hop wireless network model, consisting of V nodes, collectively denoted as $\mathcal{V}$. The nodes communicate with each other via directed wireless links, denoted as $\mathcal{L}$. The end-to-end flow set is represented by $\mathcal{R}$. Each flow $r \in \mathcal{R}$ with sending rate x_r is associated with a utility function $U_r(x_r)$, which is concave and twice differentiable. The link set the flow r traverses is denoted as $\mathcal{L}(r)$, and the flows incident to link $l \in \mathcal{L}$ are denoted as $\mathcal{R}(l)$. The communications in a wireless network is subject to location dependent interference. In this work we adopt the concept of conflict graph [24] to model wireless interference. Each vertex in the conflict graph represents a wireless link of the original network. An edge exists between two vertices if their corresponding wireless links interfere with each other.

We employ a scheduling scheme that schedules the wireless links on a slotted time basis. In this scheme, one independent set I is selected from the conflict graph in each time slot and only the links corresponding to the vertices in I can be active. This

scheduling scheme ensures collision-free packet transmission, because there is no interference between any pair of nodes within the independent set. The scheduling scheme implicitly determines the capacity region.

Let c_l represent the amount of bits that can be transmitted per second (so-called original link capacity) along a link l once it is scheduled. We denote the original capacity vector by $\boldsymbol{c}$, where $\boldsymbol{c} = (c_l, l \in \mathcal{L})$. The feasible rate $\hat{c}_l$ of link l that is achievable under a scheduling scheme is the link capacity averaged over time when it is active. We use $\hat{\boldsymbol{c}} = (\hat{c}_l, l \in \mathcal{L})$ to denote the feasible rate vector. Let the L-dimensional column vector r^I represent the capacity vector of I, where $r_l^I = c_l$ if $l \in I$, and $r_l^I = 0$ otherwise. Further we define $\Lambda := \sum_I \alpha_I r^I$, where $\sum_I \alpha_I = 1$ and $\alpha_I \geq 0$. Λ is a convex hull [60], which contains all the feasible rate vectors [15] under any inference-free scheduling, i.e., $\hat{\boldsymbol{c}} \in \Lambda$.

We collect the notations used in this chapter in Tab. IV.1.

Notation	**Definition**
$v \in \mathcal{V} = \{1, 2, ..., V\}$	Node set
$r \in \mathcal{R} = \{1, 2, ..., R\}$	End-to-end flow set
$l \in \mathcal{L} = \{1, 2, ..., L\}$	Wireless link set
$\mathcal{L}(r)$	Links the flow s traverses
$\mathcal{R}(l)$	Flows incident to link l
Λ	Original Capacity Space
$\boldsymbol{P}_l$	Penalty function of link l
$\boldsymbol{c} = (c_l, l \in \mathcal{L})$	Original link capacity vector
$\hat{\boldsymbol{c}} = (\hat{c}_l, l \in \mathcal{L})$	Feasible link rate vector
$\hat{\boldsymbol{c}}^L = (\hat{c}_l^L, l \in \mathcal{L}) \in \bar{\Lambda}$	Slow time scale capacity vector
$\boldsymbol{y} = (y_l, l \in \mathcal{L})$	Aggregate flow rate vector
$\boldsymbol{x} = (x_r, r \in \mathcal{S})$	Flow sending rate vector
$\boldsymbol{\mu} = (\mu_l, l \in \mathcal{L})$	Link congestion price vector

Table IV.1: Notations

Problem Description

As we discussed in Chapter III, the cross-layer congestion control problem under time-invariant channel can be solved by the joint congestion control and scheduling algorithm described by Eq.(III.8) - Eq.(III.10). However, as being demonstrated in the study of [62], wireless channel capacity could be highly dynamic. We can formulate the *NUM* problem under time-varying channel, and formulation is similar to the formulation of W_T in Chapter III. The previous studies usually rely on some simplified wireless channel models to obtain the optimal rate allocation and prove the system stability. Another dynamic factor is feedback delay, which we have also discussed in Chapter III. Feedback delay under time-varying channel, could lead to performance issues to a cross-layer control algorithm, such as system instability, rate over-provisioning, etc.

Approach

We propose an optimization framework for the resource management problem of time-varying wireless networks and a time-scale decomposition approach to solve the problem.

Time-Aggregated Optimization Framework

To address the inherent challenge of handling time-varying channel capacity with the presence of feedback delay in congestion control, we present a new time-aggregated optimization framework W_1 for congestion control in time-varying wireless network. This framework relaxes the original framework from two perspectives: 1) its solution of rate allocation does not guarantee the aggregated utility of all flows to be optimal at each time instance, rather it targets at optimizing the aggregated utility over a certain time

span; 2) it allows for packet losses in the optimal solution and imposes a corresponding penalty. This is in contrast with the original formulation where resource capacity imposes a hard constraint that ensures no packet losses at system equilibrium, formally,

$$W_1: \quad \max \quad \int_0^T M(\boldsymbol{x}(t))dt \tag{IV.1}$$

where $\forall t \in [0, T]$,

$$M(\boldsymbol{x}(t)) = \sum_{r\in\mathcal{R}} U_r(x_r(t)) - \sum_{l\in\mathcal{L}} P_l(\sum_{r\in\mathcal{R}(l)} x_r(t), \hat{c}_l(t)) \tag{IV.2}$$

Recall that the feasible rate vector $\hat{\boldsymbol{c}}$ is determined by the link scheduling algorithm. This vector specifies the upper bounds of data rates that can be allocated over the links. In the dynamic channel scenario, $\hat{\boldsymbol{c}}$ is time-varying. With the presence of delay, it is almost impossible to achieve loss-free delivery with limited queue size, because the data rate allocated over a link could be higher than $\hat{c}_l$, which is known as the *overshooting* problem. To evaluate the cost of excessive rate allocation over a link, we introduce a penalty function $P_l(\cdot)$ for each link. The penalty function is a continuous, nondecreasing convex function of the difference between the allocated rate and the feasible rate. Fig. IV.1 presents an example of the penalty function curve.

Time-Decomposed Congestion Control and Scheduling

The perturbation of a channel state is associated with its fast time scale component. Directly tracking the instantaneous channel variation may lead to improper utilization with the presence of delay. To address this problem, we present a time-scale decomposition approach. Instead of directly using $\hat{\boldsymbol{c}}$ generated by the scheduling module, we

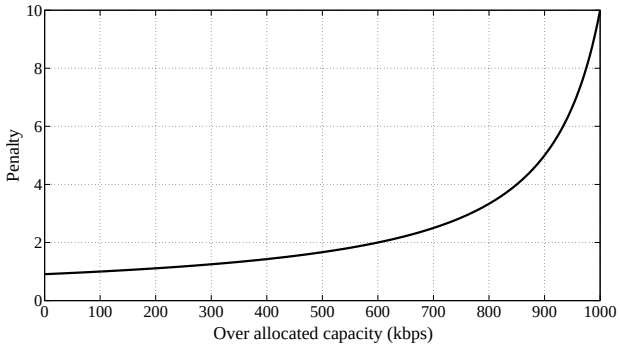


Figure IV.1: Penalty function example.

introduce a virtual capacity vector $\hat{\boldsymbol{c}}^L$, called the *slow time scale capacity*, which varies sufficiently slow so that it can be tracked by the rate controller even with the presence of feedback delays. We use $\hat{\boldsymbol{c}}^L$ to constrain the rate allocation. Moreover, $\hat{c}_l^L$ can be used together with the feasible rate vector $\hat{c}_l$ to estimate the overshot rate of a link l. Now, problem W_1 is converted to problem W_2:

$$\begin{aligned}
\max \quad & \int_0^T (\sum_{r\in\mathcal{R}} U_r(x_r(t)) - \sum_{l\in\mathcal{L}} P_l(\hat{c}_l^L(t), \hat{c}_l(t)))dt \\
\text{s.t.} \quad & \sum_{r\in\mathcal{R}(l)} x_r(t) \le \hat{c}_l^L(t), \forall l \in \mathcal{L}, \\
\text{over} \quad & \hat{\boldsymbol{c}}^L(t) = \Omega(\hat{\boldsymbol{c}}(t)), \hat{\boldsymbol{c}}(t) \in \Lambda(t), \forall t \in [0,T]
\end{aligned}$$

Note that we use $\Omega(\cdot)$ to represent the process that generates the slow time scale capacity vector $\hat{\boldsymbol{c}}^L$. Taking the Lagrangian of W_2, we obtain:

$$
\begin{aligned}
& \max \int_0^T (\sum_{r\in\mathcal{R}} U_r(x_r(t)) - \sum_{l\in\mathcal{L}} P_l(\hat{c}_l^L(t), \hat{c}_l(t)) - \\
& \sum_{l\in\mathcal{L}} \mu_l(t)(\sum_{r\in\mathcal{R}(l)} x_r(t) - \hat{c}_l^L(t)))dt \qquad \text{(IV.3)} \\
= \quad & \max \int_0^T (\sum_{r\in\mathcal{R}} U_r(x_r(t)) - \sum_{l\in\mathcal{L}} \mu_l(t) \sum_{r\in\mathcal{R}(l)} x_r(t))dt + \\
& \max \int_0^T (\sum_{l\in\mathcal{L}} \mu_l(t)\hat{c}_l^L(t) - \sum_{l\in\mathcal{L}} P_l(\hat{c}_l^L(t), \hat{c}_l(t)))dt \\
\text{over} \quad & \hat{\boldsymbol{c}}_l^L(t) = \Omega(\hat{\boldsymbol{c}}(t)), \hat{\boldsymbol{c}}(t) \in \Lambda(t), \forall t \in [0,T]
\end{aligned}
$$

Then we solve the above problem at discrete time instances, by decomposing it to two subproblems: $\mathcal{M}_1$ and $\mathcal{M}_2$.

$$
\begin{aligned}
\mathcal{M}_1: \quad \max \quad & \sum_{r\in\mathcal{R}} U_r(x_r(t)) - \sum_{l\in\mathcal{L}} \mu_l(t) \sum_{r\in\mathcal{R}(l)} x_r(t) \qquad \text{(IV.4)} \\
\text{s.t.} \quad & \sum_{r\in\mathcal{R}(l)} x_r(t) \le \hat{c}_l^L(t), \forall t \in [0,T]
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{M}_2: \quad \max \quad & \sum_{l\in\mathcal{L}} \mu_l(t)\hat{c}_l^L(t) - \sum_{l\in\mathcal{L}} P_l(\hat{c}_l^L(t), \hat{c}_l(t)) \qquad \text{(IV.5)} \\
\text{over} \quad & \hat{\boldsymbol{c}}(t) \in \Lambda(t), \forall t \in [0,T]
\end{aligned}
$$

$\mathcal{M}_1$ and $\mathcal{M}_2$ actually follow different time scales. $\mathcal{M}_1$ is the utility maximization problem over the slow time scale capacity $\hat{\boldsymbol{c}}_l^L(t)$, at time instance t. $\mathcal{M}_2$ is a new scheduling problem that integrates the overshooting penalty, called *Penalty-aware Scheduling*, which is on a fast time scale to track the channel dynamics.

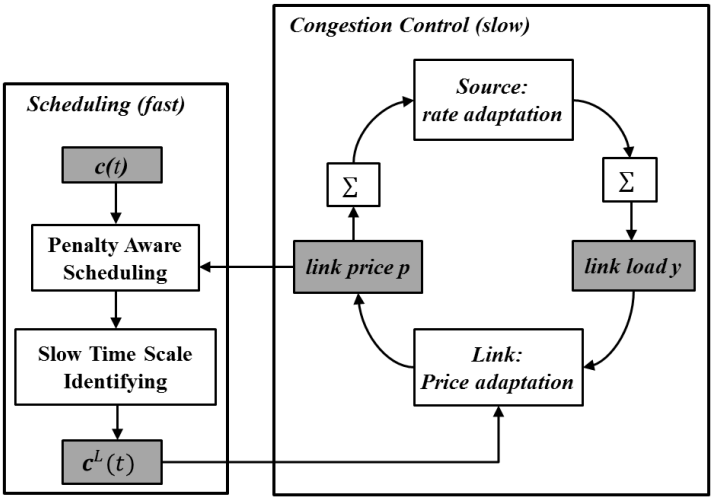


Figure IV.2: Time decomposition approach overview.

Fig. IV.2 provides an overview of our approach. The *slow time scale capacity identifying* component keeps monitoring the instantaneous feasible rate vector and generates the slow time scale capacity vector. The *congestion control* component determines how to allocate the sending rates. The *penalty-aware scheduling* component schedules the wireless links and determines the feasible rate vector $\hat{\boldsymbol{c}}$. The slow time scale capacity vector $\hat{\boldsymbol{c}}^L$ is derived from $\hat{\boldsymbol{c}}$ based on the properties of a slowly-varying system, which will be discussed in the following sub-section. On the other hand, $\hat{\boldsymbol{c}}^L$ have impact on $\hat{\boldsymbol{c}}$ via the penalty function $P(\cdot)$.

Identifying The Slow Time Scale Capacity Vector

Identifying the slow time scale capacity vector is the key step towards realizing the time decomposition based resource management. The slow time scale capacity vector $\hat{\boldsymbol{c}}^L$ should vary slowly so that at a particular time instance t, it can be considered as a frozen parameter and the congestion control algorithm has an isolated equilibrium point

[67]. This requires the derivative of $\hat{\boldsymbol{c}}^L$ with respect to time, $\left\|\dot{\hat{\boldsymbol{c}}}^L\right\|$, is sufficiently small. Furthermore, the slow time scale capacity should be insensitive to delay effect, which means the difference between $\hat{c}_l^L(t)$ and $\hat{c}_l^L(t-\tau)$ is bounded. If the system is described by $\dot{x} = f(\boldsymbol{x}, \hat{\boldsymbol{c}}^L)$, the slow time scale capacity vector needs to satisfy the following conditions:

$$\int_{t_0}^{t_0+\tau_{max}} \left\|\dot{\hat{\boldsymbol{c}}}^L(t)\right\| dt \leq K_1 \tag{IV.6}$$

$$\left\|\frac{\partial f(\boldsymbol{x}, \hat{\boldsymbol{c}}^L)}{\partial \hat{\boldsymbol{c}}^L}\right\| \leq K_2 \left\|\boldsymbol{x} - \boldsymbol{x}^*(\hat{\boldsymbol{c}}^L)\right\| \tag{IV.7}$$

Here τ_{max} is the maximum link round-trip delay. K_1, K_2 are positive constants. Condition (IV.6) can be simply interpreted as: the accumulated variation of the slow timescale capacity over any time interval of $[t_0, t_0+\tau_{max}]$ is upper bounded by a particular constant value K_1. The bound K_1 determines how "slow" the capacity variation is. Condition (IV.7) is derived based on Lemma 9.8 of [67]. It specifies that the derivative of the system dynamic along the trajectory of the slow time scale capacity is upper bounded. Overall, (IV.6) and (IV.7) require the trajectory of $\hat{\boldsymbol{c}}^L$ to be "sufficiently" slow and the variation of the system along the direction of the capacity trajectory is upper bounded. Under the constraints, the slow time scale capacity vector is considered as a "frozen" or slowly varying parameter, hence the congestion control system has an equilibrium point at each time instance (on the slow time scale). In our system, we denote the upper bound of the derivative of $\hat{c}_l^L$ as δ_l.

Our congestion control component is based a primal-dual controller [46] which adjusts both the rate and the congestion price. At a particular time instance, if the slow time scale capacity vector is regarded as $\hat{\boldsymbol{c}}^L$, the control algorithm is

$$\dot{x}_r(t) = \lambda_r \left(1 - \frac{q_r^d(t)}{U_r'(x_r(t))}\right) \tag{IV.8}$$

$$\dot{\mu}_l(t) = \left(\gamma_l(y_l^d(t) - \hat{c}_l^L)\right)_{\mu_l}^+ \tag{IV.9}$$

Here $q_r(t) = \sum_{l \in \mathcal{L}(r)} \mu_l^d(t)$ and $y_l(t) = \sum_{r \in \mathcal{R}(l)} x_r^d(t)$ represent the delayed congestion price signal and rate signal respectively. Since the system is regarded as a slowly varying system, it can follow the corresponding capacity trajectory. We use the utility function $U_r(x_r) = \omega_r log x_r$. Now we investigate the stability of the controller (IV.8)-(IV.9) around the equilibrium point $(\boldsymbol{x}^*, \boldsymbol{\mu}^*)$. After linearization, the Laplace transforms of (IV.8) and (IV.9) are:

$$x_r(s) = \frac{1}{s + \frac{\lambda_r(x_r^*)}{\omega_r}} \left(-\frac{\lambda_r(x_r^*)}{\omega_r} x_r^* q_r(s) + x_r(0)\right) \tag{IV.10}$$

$$\mu_l(s) = \frac{1}{s}\left(r_l(\mu_l^*) y_l(s) + \mu_l(0)\right) \tag{IV.11}$$

Hence the transfer function can be easily obtained:

$$G(s) = diag\left\{\frac{\lambda_r x_r^*}{\omega_r s + \lambda_r}\right\} D(s) R^T(-s) diag\left\{\frac{\gamma_l}{s}\right\} R(s) \tag{IV.12}$$

where $D(s) = diag\left\{e^{-s\tau}\right\}$. $R_{lr} = e^{-s\tau_b}$ if $l \in \mathcal{L}(r)$, and $R_{lr} = 0$ otherwise. By Theorem 6.12 of [46], the sufficient condition of local asymptotic stability is

$$\gamma_l \leq \frac{\mu_l^*}{\hat{c}_l^L \tau_{max}}, \forall l \in \mathcal{L} \tag{IV.13}$$

where τ_{max} is the maximum value among all the round trip delay values associated with l. In Tab. IV.2 we present a distributed congestion control algorithm: $Alg.C$. The algorithm involves two time scales: one fast time scale for the scheduling component and one slow time scale for the congestion control component.

Algorithm II: *Alg.C*
Capacity Update (fast)
At each scheduling slot:
A link is selected according to S
Update $\hat{\boldsymbol{c}}(t)$
At the beginning of each capacity update slot,
generate $\boldsymbol{c}^L(t)$ from *Alg.S'*
Congestion control (slow)
On receiving an packet,
$\forall r \in \mathcal{R}$, perform *Rate adaptation*:
$\dot{x}_r(t) \leftarrow \lambda_r \left(1 - \frac{q_r^d(t) x_r(t)}{\omega_r}\right)$
At the beginning of each price update cycle,
$\forall l \in \mathcal{L}$, perform *Price Adjustment*:
$\dot{\mu}_l(t) \leftarrow \left(\gamma_l (y_l^d(t) - c_l^L)\right)_{\mu_l}^+$

Table IV.2: Congestion control algorithm

Penalty Aware Scheduling

We propose a penalty aware scheduling algorithm, which schedules the wireless links according to the link congestion degree, instantaneous channel condition and over-utilization penalty. Recall the link scheduling problem $\mathcal{M}_2$ over the time-varying capacity region $\Lambda(t)$:

$$\begin{aligned} \max \quad & \left(\sum_{l \in \mathcal{L}} \underbrace{\mu_l(t)\hat{c}_l^L(t) - P_l(\hat{c}_l^L(t), \hat{c}_l(t))}_{w_l} \right) \\ \text{over} \quad & \hat{\boldsymbol{c}}(t) \in \Lambda(t), \forall t \in [0, T] \end{aligned} \tag{IV.14}$$

$\mathcal{M}_2$ can be naturally interpreted as maximizing the summation of all link weights across the network, where the weight of link l is denoted by w_l. As indicated by (IV.14), w_l consists of two components: the product of the slow time scale capacity and the rate, w_1, and the penalty value w_2. w_1 is essential for achieving the throughput optimality [30]. w_2 can be simply understood as a numerical measurement of the penalty caused by the excessive utilization of link capacity. Specifically, we define the function $P_l(\cdot)$ as follows:

$$P_l(\hat{c}_l^L(t), \hat{c}_l(t)) = \begin{cases} 0 & \text{if } \Delta c_l(t) \leq 0 \\ \frac{a_l}{b_l - \Delta c_l(t)} & \text{otherwise} \end{cases}$$

where

$$\Delta c_l = \hat{c}_l^L(t) - \delta_l - \hat{c}_l(t)$$

Here $\hat{c}_l^L(t) - \delta_l$ is an estimation of the lower bound of the slow time scale capacity that will be generated in the current slot. a_l and b_l are both positive constant values.

Now the scheduling principle is straightforward: at the beginning of each scheduling slot, the server of the scheduling algorithm needs to make a scheduling decision $S(t)$ such that

$$S(t) = \arg\max \sum_{l \in \mathcal{L}} w_l(S(t)) \tag{IV.15}$$

Towards this end, we define an incentive variable d_l, $\forall l \in \mathcal{L}$,

$$d_l(t) = \mu_l c_l(t) + P_l(\hat{c}_l^L(t), \hat{c}_l(t)) \tag{IV.16}$$

At the beginning of each scheduling slot, the server selects one link with the maximum incentive value. The intuition behind this is: during each slot, the scheduling is based on the real time channel condition $\boldsymbol{c}(t)$. To increase w_1, the scheduler needs to select a link with larger $\mu_l c_l(t)$. Meanwhile, to reduce the penalty, the scheduler should schedule a link with larger $P_l(\hat{c}_l^L(t), \hat{c}_l(t))$, hence the capacity $\hat{c}_l$ will increase and the penalty will decrease in next slot.

We use a heuristic algorithm to estimate the slow time scale capacity. More specifically, we define a capacity update periods T_c, and bound the change of the link capacity between two adjacent capacity update periods. In Tab. IV.3 we present a distributed slow time scale capacity vector estimation algorithm *Alg.S'*, through which each link autonomously generates its slow-time-scale capacity over time. At the beginning of each capacity update slot, link l first loads the current $\hat{c}_l(t)$. The algorithm ensures the change of capacity $|\hat{c}_l^L(t) - \hat{c}_l^L(t-1)|$ is bounded by δ_l. The capacity also satisfies $\eta_l \frac{\tau}{T_c} \ll |\hat{c}_l^L|$, where T_c represents the length of one capacity update slot and η_l is a positive constant.

Algorithm I: *Alg.S'*

1. $\forall l \in \mathcal{L}$, update $\hat{c}_l(t)$
2. **if** $\hat{c}_l(t) \in [\hat{c}_l^L(t-1) - \delta_l, \hat{c}_l^L(t-1) + \delta_l]$
3. $\quad \hat{c}_l^L(t) \leftarrow \hat{c}_l(t)$
4. **else if** $\hat{c}_l(t) < \hat{c}_l^L(t-1) - \delta_l$
5. $\quad \hat{c}_l^L(t) \leftarrow \hat{c}_l^L(t-1) - \delta_l$
6. **else**
7. $\quad \hat{c}_l^L(t) \leftarrow \hat{c}_l^L(t-1) + \delta_l$
8. **end if**
9. Return $\hat{c}_l^L(t)$

Table IV.3: Slow time scale capacity vector estimation

Cross-Layer Design

Our algorithm can be realized via a cross-layer design, including the wireless link scheduling component and the congestion control component, which are on different time scales. The scheduling component works on the media access control (MAC) layer. It is based on the *TDMA* mechanism and the maximum weight matching algorithm: the time is divided into slots, at the beginning of each scheduling slot, the scheduling decision is made in order to maximize the aggregate link weight. The penalty-aware link weight is defined in Eq.(IV.16). We use *Alg.S'* to obtain the slow time scale capacity vector $\boldsymbol{c}^L$ from the feasible rate vector, which is directly generated by the scheduling component.

The congestion control component implicitly follows a slow time scale because it uses $\hat{\boldsymbol{c}}^L$. Here we use a primal-dual controller for price adaptation and rate adaptation, as listed in Tab. IV.2. The link congestion price variable is generated on the network layer, where each link calculates the congestion price according to the current slow time scale capacity and the load over the link. The rate of a flow is controlled by the source node, on the transport layer, using the aggregate congestion price along the flow route.

Simulation Study

We evaluate the performance of our approach through trace-drive simulation.

Experimental Setup

We use the packet level network simulator ns-2. The MAC layer module of the simulated networks is developed based on the ns-2 *TDMA* module. It adopts the penalty aware scheduling mechanism. Furthermore, we have implemented a primal-dual controller as the congestion control module.

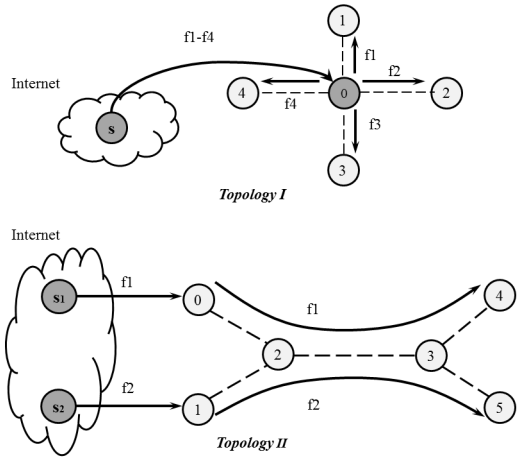


Figure IV.3: Simulation networks.

We use two network topologies, as shown in Fig. IV.3: the first topology is a wireless LAN network consisting of one wireless access point (wireless router), four wireless clients and four data flows. The other topology is a multi-hop wireless network with multiple wireless routers and two data flows. In both scenarios, the data flow sources in the Internet connect with the wireless routers via wire-line connections.

The utility function of flow r is: $U_r(x_r) = \omega_r log x_r$, where ω_r is 1.0. In our experiments, the queue size of each node is 100 (in packets). We consider packet loss due to queue overflow. The lost packets are not retransmitted. The forward delay and backward delay are both $100ms$ for any source-link pair, unless otherwise specified.

We use the following two types of wireless network traces to simulate the time-varying capacities of the wireless links: The first trace is a 4-state Markov chain trace with uniform inter-state transition probability. In each channel state, the link capacity can be one of the following values: 6Mbps, 7Mbps, 8Mbps and 9Mbps. The second trace is retrieved from the Roofnet trace [62], which contains the original packet transmission information in a IEEE 802.11b mesh network. In our simulation, the original link capacity vector is updated every $200ms$.

We compare the performance of our algorithm, called $Alg.O$ in this section, with two baseline algorithms:

- $Alg.B_I$ is a joint congestion control and scheduling algorithm with a fixed routing, where the congestion control follows Eq.(III.9) - Eq.(III.10), and the scheduling follows Eq.(III.8), as described in Chapter III The algorithm is based on the cross layer congestion control algorithm as introduced in [15]. It keeps monitoring the channel conditions and adjusts flow rate allocation. To focus on the performance comparison, we removed the routing component of the algorithm.

- $Alg.B_{II}$ is a heuristic joint congestion control and scheduling algorithm. It adopts the same congestion control mechanism with $Alg.B_I$, however, the scheduling policy used here is a round-robin style link scheduling mechanism, which implies the every link has equal probability to be scheduled.

Slow Time Scale Capacity

In Fig. IV.4, we plot the instantaneous feasible rate $\hat{c}_l$ and the slow time scale capacity $\hat{c}_l^L$ of link $0 \rightarrow 1$ of Topology I. Obviously the feasible rate is highly dynamic, due to the time-varying channel. In contrast with $\hat{c}_l$, $\hat{c}_l^L$ changes slowly. Since the slow time scale capacity reduces the perturbation, it is beneficial to the stability of the system.

Flow Rate Allocation

Now we examine the flow rate allocation with different network configurations. First, we investigate the performance of the baseline algorithms with Topology 1. In Fig. IV.5 we plot the results of $Alg.B_I$ and $Alg.B_{II}$, Clearly using the baseline algorithms results in very fluctuating outputs. Next we evaluate the performance of our

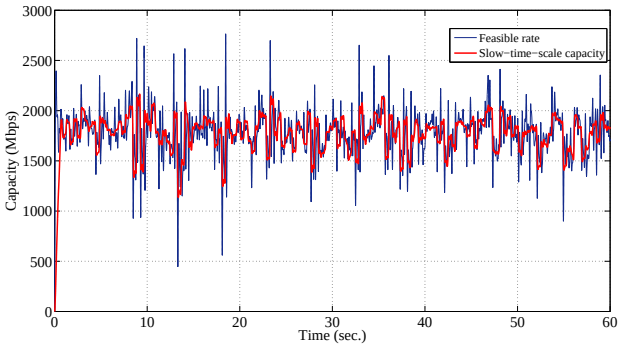
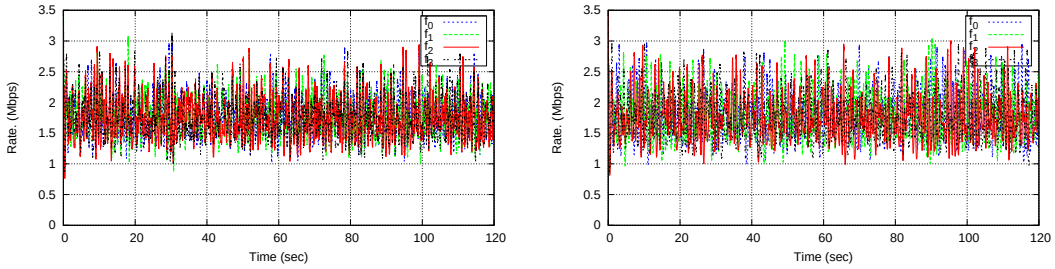


Figure IV.4: Feasible rate and slow time scale capacity.



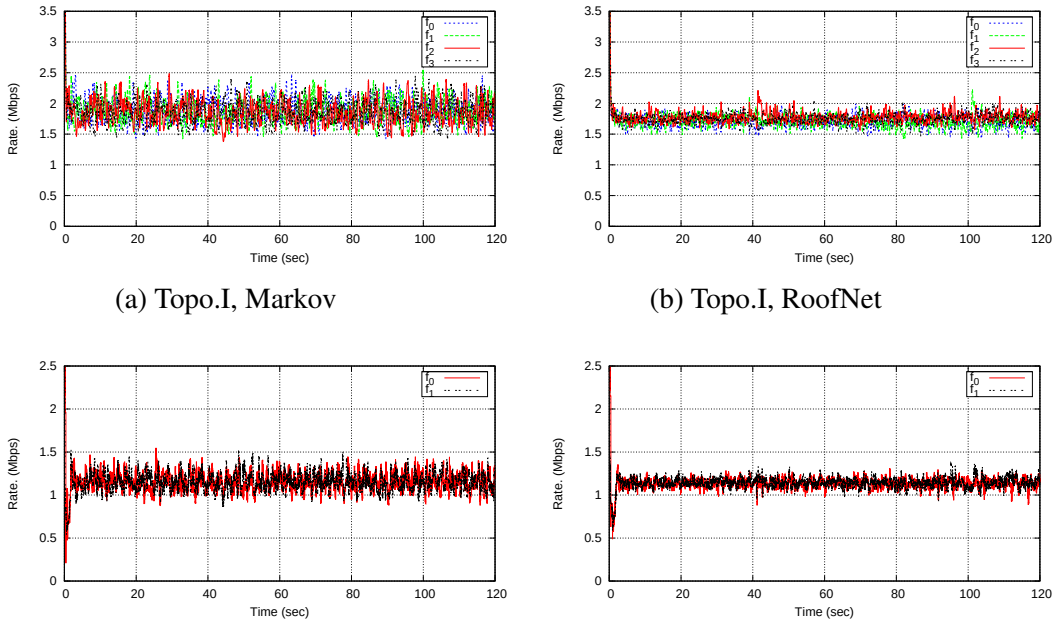
(a) Topo.I, Baseline Algorithm I (b) Topo.I, Baseline Algorithm II

Figure IV.5: Sending rates under the baseline algorithms

approach under various network configurations. Fig. IV.6 illustrates the instantaneous flow rates generated by our algorithm. The flow rates also vary over time under the time-varying channel, but the burstiness is greatly reduced compared with the baseline algorithms. Meanwhile, we note that the flow rates maintain some stability properties: they are bounded even with the presence of feedback delay.

Packet Delivery and Loss Ratio

In order to evaluate the efficiency of our algorithm in packet delivery, we introduce *loss ratio*, which is defined as *the number of packets lost* divided by *the number of packets delivered*. This metric can be interpreted as the quantity of packets lost for each



(a) Topo.I, Markov

(b) Topo.I, RoofNet

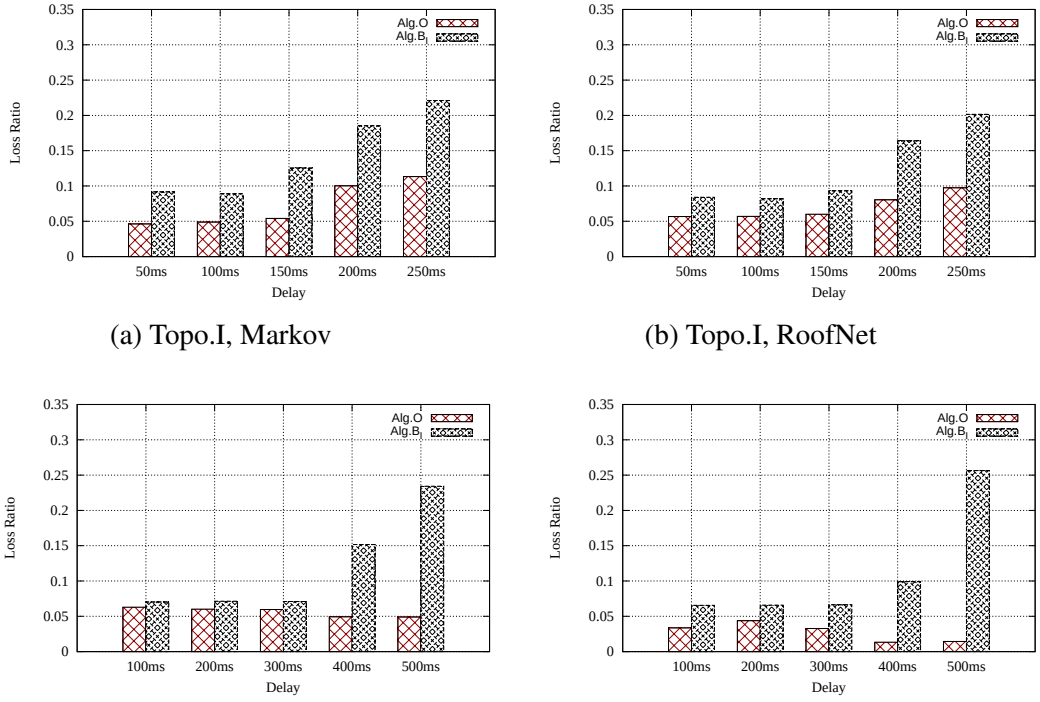
(c) Topo.II, Markov

(d) Topo.II, RoofNet

Figure IV.6: Sending rates under the optimal algorithm.

successfully delivered packet. We change the delay values and plot the corresponding loss ratios of $Alg.O$ and $Alg.B_I$. Here we assume the forward delay and backward delay are identical. For each network scenario, the experiment runs for 400 seconds. Fig. IV.7 shows the results across different delay values. We notice that $Alg.B_I$ suffers performance degradation as the delay value increases. Especially, the loss ratio with the Markov trace is usually higher than the loss ratio with the RoofNet trace. This is because the Markov trace is more bursty than the RoofNet trace. Compared with the baseline algorithm, $Alg.O$ is more resilient to delay effect: the loss ratio remains under a relatively low level. Therefore our approach is more efficient in controlling the loss ratio under the network dynamics.

Next we compare the packet delivery performance of the three algorithms in the multi-hop wireless network topology and plot the results in Fig. IV.8. The results imply that larger delay usually results in more packet loss. The number of delivered packets



(a) Topo.I, Markov

(b) Topo.I, RoofNet

(c) Topo.II, Markov

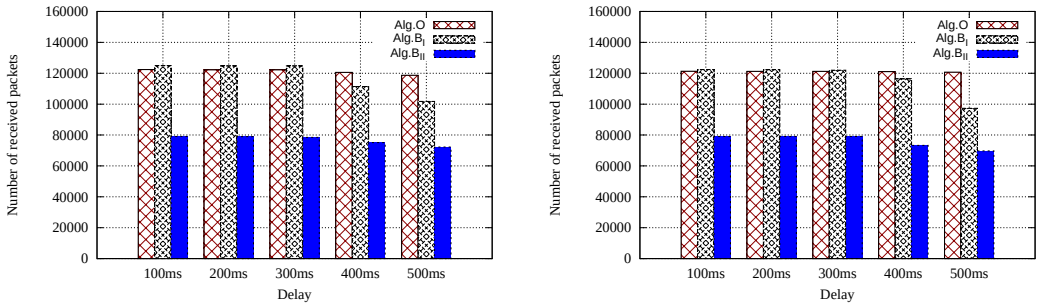
(d) Topo.II, RoofNet

Figure IV.7: Packet loss ratio of the simulation scenarios.

under $Alg.B_{II}$ is obviously less than the results of the other two algorithms. This is because the round-robin scheduling adopted by $Alg.B_{II}$ does not guarantee throughput optimality. We also observe that the throughput performance under $Alg.O$ is close to or even sometimes better than $Alg.B_I$. This proves that $Alg.O$ achieves good throughput performance while reducing packet loss.

Discussion

Similar to ROCS, the time-scale decomposition approach presented in this chapter is also a joint congestion control and scheduling solution for time-varying wireless networks with the presence of feedback delay. The major difference is that the time-scale



(a) Markov (b) Roofnet

Figure IV.8: Packets delivered in the multi-hop network.

decomposition is based on the virtual capacity vector (scheduled capacity vector). Furthermore, we present a penalty-aware scheduling algorithm that is on a fast time scale, which conservatively tracks the instantaneous channel condition.

CHAPTER V

DYNAMIC-AWARE MOBILE APPLICATION OFFLOADING

In this chapter we look beyond the network infrastructure and investigate the approaches for managing distributed application execution over mobile devices and the back-end platform. In particular, we present a dynamic mobile application task offload solution which migrates the computation-intensive tasks from a mobile application to its back-end system at runtime based on the user demand and network condition. Our solution aims to improve the overall mobile user experience by minimizing the energy consumption.

The remainder of the chapter is organized as follows. First we overview our technique, then in the following section we present the problem formulation. In the next two sections we introduce the dynamic-aware offload system and present the test results. Finally we conclude the chapter.

Overview

Optimizing the resource usage within the wireless network infrastructure benefits the performance of wireless networks. However, this is barely enough to satisfy the growing user requirements on mobile applications, with a series of constraints that stem from CPU speed, memory size, disk space, and battery life. This motivates the application layer solutions in the ecosystem consisting of mobile terminals, mobile applications and external computational resources (such as mobile cloud). Our second tier solution belongs to this category. Our goal is to enhance the overall user experience with mobile systems. In this work, we capture the impact of a mobile application task execution on

user experience using the total energy consumed by this task during its execution. The rationales behind this model are: 1) the battery lifetime is a critical factor for mobile systems, with longer battery life leading to better user experience [68]; 2) this metric implicitly reflects, though not directly linked to, the overall application execution time. The longer execution time usually leads to higher energy consumption. At the same time, we admit that our model cannot capture other aspects of application performance and their impact to user experience. For example, this model does not consider highly interactive applications, where round-trip message latency plays an important role in the user QoS.

To minimize energy consumption, we employ the approach of *offloading*: a mobile application can be distributedly executed over mobile devices and a back-end platform. In particular, we introduce a dynamic mobile application offload system to intelligently migrate the computation-intensive tasks from a mobile application to a back-end server at runtime. Task offloading has been widely adopted in mobile applications. An example is the *Mobile Cloud Computing* (MCC) paradigm [69, 70] that enhances mobile user experience by leveraging cloud resources. Existing works have explored the possibilities of offloading mobile applications at various levels of granularity, by using techniques like virtualization, partition, migration and remote execution [6, 13, 71, 72, 73]. Offloading mobile applications to resource-rich devices leads to several immediate benefits. When the cost of offloading is low compared with local execution, the battery energy consumption is reduced. This could greatly extend battery life. Moreover, the mobile application can use the computational resources of the external devices. With abundant resources, even the computation-intensive tasks, such as speech recognition, image processing and decision making can be well supported.

However, in real world applications, many unpredictable factors may largely affect the cost of offloading, such as network condition and user demand. Generally, the decision of offloading is made only if the cost of local execution is higher than the cost of remote execution (mainly incurred by migrating the mobile task to the back-end platform and receiving the results) [6]. Therefore, how to make the offload system intelligently migrate the tasks from mobile terminals becomes important.

We present a dynamic-aware mobile application offload system to accomplish cost-effective offloading. The system is capable of migrating the computation-intensive tasks of a mobile application to the back-end platform, such as a server. We model the execution of a computation task by the concept of *execution scheme*, which refers to a particular execution plan of the task. A task may contain multiple execution schemes. We design a runtime solver to make a decision on which execution scheme should be applied to the application, according to the quality of the wireless connection and the user demand of the application. The offload system consists of four modules.

- *The execution scheme module*. Identifying the execution schemes of a mobile application is the prerequisite of offloading. Generally, the computation-intensive and non-native components are good candidates for remote execution.

- *The energy cost models*. The decision of execution scheme selection is made by using the models. The models are derived through dynamic profiling and regression techniques. By fitting the dynamic information collected into the models, both the energy consumption of the local execution and the energy consumed during offloading can be evaluated.

- *The solver module*. By incorporating multiple dynamic factors, including the user demand and the network condition, the solver module makes a decision on which execution scheme is optimal.

- *The remote mobile method execution module*. This module enables the seamless invocation of a mobile method (a function originally on the mobile application) on the back-end server. This module is similar to the Java RMI API. The mobile application can invoke the remote execution of a method on the server, as if it is invoked locally on the mobile terminal. The module is responsible for the implementation of mobile task migration, including registering the method with the server, serializing the parameter objects and retrieving the results after the remote execution is completed.

Compared with the existing works, our solution is dynamic-aware and cost-effective. We consider the network and user demand dynamics. The offload system determines how the computation-intensive tasks of a mobile application could be executed to achieve the optimal energy consumption. The contributions of the study are summarized as follows. (1) We propose a novel theoretical framework that characterizes the problem of dynamic-aware mobile application offloading. (2) We present a dynamic profiling and regression based approach to model the relationship between runtime dynamics and energy cost. (3) We implement a remote execution module, which enables seamless migration of mobile task and can be easily applied to legacy code. Furthermore, it requires no change to the kernel of the mobile operating system.

Problem Formulation

In this section, we first discuss the impact of the dynamic factors, such as the quality of wireless connection and the user demand, on energy consumption during offloading. Then we present the formulation of the decision-making problem.

Network Condition

Offloading a task from a mobile terminal relies on data transmission via wireless networks, either WiFi networks or cellular networks. Both sending and receiving packets consume battery energy. In fact accessing WiFi and 3G/4G networks is one of the major sources to drain the battery of mobile devices [7, 74, 75]. The energy consumed during wireless network accessing depends on the quality of wireless connections, which can be evaluated by metrics like signal strength, signal noise ratio (SNR), etc. As indicated by the study of Ding et al. [75], weak signal strength may trigger the rate adaptation at the physical layer [76] and thus increase the transmission time and energy consumption. In addition, weak signal strength may also lead to packet retransmission and reconnecting with the access points, which further increases energy consumption. Offloading the same task may result in significantly different amount of energy consumption. Generally, better wireless connectivity implies lower transmission cost. In this study, we use *Received Signal Strength Indicator* (RSSI) to measure the quality of a wireless connection. RSSI is widely used to measure the power level in a received radio signal [77]. Generally, a higher RSSI value implies a stronger received signal.

User Demand

The second dynamic factor we consider is user demand. User demand greatly affects the workload which is directly related to the energy consumption. For the same type of task, the workload can be dynamic due to varying user demand. If a task is resource-intensive, it may be qualified for remote execution. On the other hand, if it is lightweight, data transmission may drain more battery energy than local execution.

User demand can affect energy consumption in the following two aspects.

- The user demand level determines the "workload" assigned by the user to the mobile application in local tasks.
- The user demand level determines the amount of data transmitted in the remote tasks during offloading.

However, it is extremely difficult to generally define the level of user demand. In the mobile applications used in our study, we assume user demands are linked with some user input related parameters, such as the dimension of a map or the size of an image, so the relationship between user demand and energy consumption can be quantitatively evaluated.

Fig. V.1 shows a simple example of this relationship. We have observed the energy cost of a micro-benchmark application: *N-Queen Puzzle* solver on an HTC Inspire 4G smartphone, where different numbers of queens represent different user demand levels. We use the online energy estimation tool *PowerTutor* [78] to measure the energy consumed locally by the CPU. *PowerTutor* can be used to measure the aggregated energy consumption of a mobile application during any specified time interval. The plot shows when $N \geq 8$, the energy consumption increases sharply. On the other hand, when N is small, the energy consumption is low.

Execution Scheme

We define an *execution scheme* ψ as a particular execution plan of a mobile task $\mathscr{T}$. Let $\mathcal{F} = \{m_1, m_2, ..., m_k\}$ be the collection of methods invoked under the scheme. In an execution plan, some methods may be executed locally and others may be executed remotely. For instance, for the same group of methods $\mathcal{F}$, let $\psi^{\mathscr{L}}$ be the scheme with all local executions, and $\psi^{\mathscr{R}}$ be a scheme with several methods executed remotely, then $\psi^{\mathscr{L}}$ and $\psi^{\mathscr{R}}$ are two distinct schemes. Fig. V.2 shows an example of a simple mobile

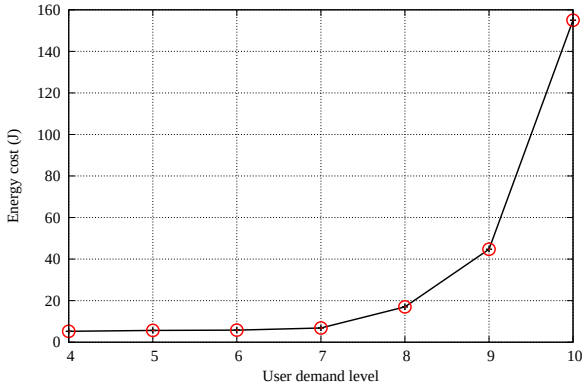


Figure V.1: Energy cost of the *N-Queen Puzzle* application with different user demand levels.

application with only one method that is qualified for remote execution. The execution schemes are very straightforward: the first is a scheme with the local execution of the service method, and the other one is a complementary scheme of the first one, with the remote execution of the method. The energy consumption of a scheme includes the energy consumed locally and the energy consumed by migrating some tasks (if there are any) to an server.

First we need to identify the possible execution schemes, which form the solution space for the runtime solver, so the optimal scheme at a time instance will be selected from the candidate scheme pool. The structure of a mobile application implicitly determines the solution space of scheme selection. More specifically, it determines which components can be offloaded. In MAUI [6] and CloneCloud [13], any method that neither accesses native resource nor creates user interface on a mobile device can be offloaded. In our study, we explore the possibility of merging the loosely coupled methods into one bundle, which can be offloaded at the same time instance. In addition, we also inspect whether a method is a good candidate for remote execution: only the computation-intensive methods will be offloaded. Reducing the frequency of remote

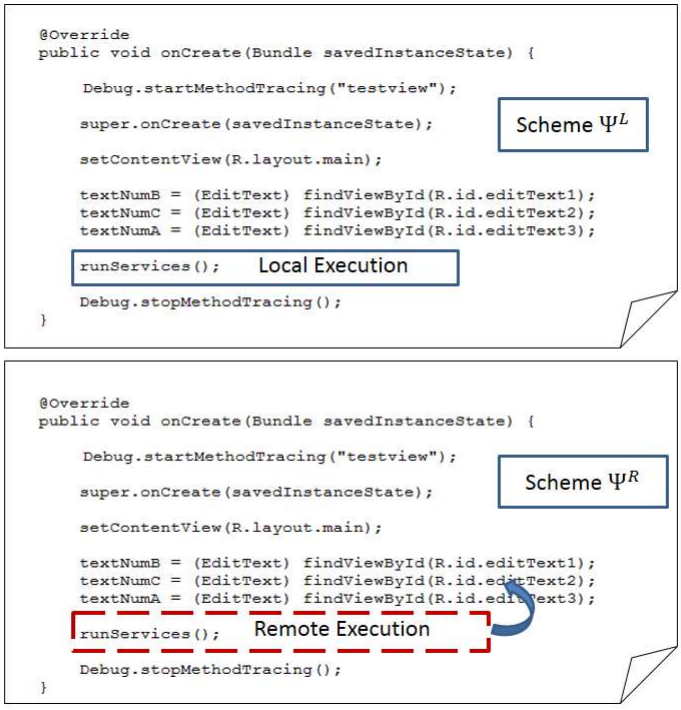


Figure V.2: Partition schemes.

execution can effectively reduce the extra overhead incurred by decision making and control message transmission.

Generalized Problem Formulation

Next we introduce the formulation of the mobile application offloading problem. The formulation relies on the assumption that the energy consumption of a mobile application is closely related to user demand and network condition.

We consider the execution of a *computation task* T of a mobile application. The network condition and user demand may vary during the execution of the task. With the dynamic environment, the mobile application may experience multiple execution scheme changes to accomplish task T. Let the set Ψ be the collection of all the execution schemes that are applicable. We say that an execution scheme ψ_u is feasible if $\psi_u \in \Psi$.

We denote the sequence of the schemes during the entire task execution period as $\Phi_T = (\psi_1, \psi_2,...,\psi_n)$. Let the set of all the feasible scheme sequences be Λ, clearly we have $\Phi_T \in \Lambda$. Furthermore, let $x_c(u)$ denote the quality of the wireless network connection, and $x_w(u)$ denote the user demand level, during the execution of scheme ψ_u. Let E_{Φ_T} represent the total energy consumption during the service time. The energy consumption of a scheme may consist of the energy consumed locally, denoted by $E^{\mathscr{L}}$, and the energy consumed by offloading, denoted by $E^{\mathscr{R}}$. Clearly if the scheme does not involve any remote execution, $E^{\mathscr{R}} = 0$. Based on our assumption, the remote execution related energy consumption of a scheme $E^{\mathscr{R}}$ is regarded as a function of the wireless network condition and the user demand. Then the total energy consumed by T can be described by the following equation.

$$\begin{aligned} E_{\Phi_T} &= \sum_{u=1}^{n} E(\psi_u) \\ &= \sum_{u=1}^{n} (E^{\mathscr{L}}_{\psi_u}(x_c(u)) + E^{\mathscr{R}}_{\psi_u}(x_c(u), x_w(u))) \end{aligned} \tag{V.1}$$

Eq.(V.1) implies that the energy consumption of the local computation is only related to user demand, and the energy consumption by offloading is related to both network condition and user demand. The problem of resource management for task T is to find the optimal execution scheme sequence Φ^*_T, which results in the minimum accumulated energy consumption. Let C denote the set of all possible network quality levels and W denote the set of user demand levels. The generalized problem G is formulated as follows.

$$G: \quad \text{minimize} \quad E(\Phi_T) \tag{V.2}$$

$$\text{subject to} \quad \Phi_T \in \Lambda,$$

$$\text{and} \quad x_c(u) \in C, x_w(u) \in W, \forall u \in [0, n) \tag{V.3}$$

The formulation provides a guideline for deriving a practical solution.

Approach

In this section, we introduce our dynamic-aware offload system. Our system consists of a method migration module, a runtime profiler and a solver. The method migration module provides a tunnel for the transient remote execution of the offloaded tasks. The runtime profiler collects the real time information of the network condition and user demands and updates the profiles to the solver. The solver keeps evaluating the energy consumption levels of different feasible execution schemes. It changes the current scheme if a more energy-efficient scheme is found. The parameters required by the remote execution are serialized and delivered through a TCP connection based "tunnel". The the server remotely executes the methods and returns the results to the mobile application. Fig. V.3 shows an overview of the system architecture.

We start by presenting the implementation of the Android platform based method migration module. Then we discuss modeling the energy consumption using regression techniques. The energy model will be used by the solver to evaluate the energy costs of local execution and remote execution at runtime. Finally, we introduce the algorithm of the solver.

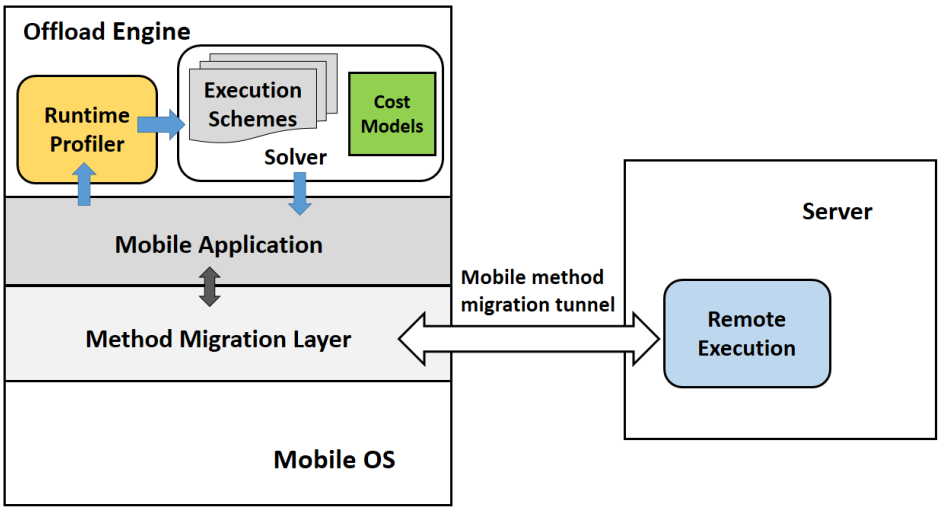


Figure V.3: System overview.

Mobile Application Method Migration

The method migration module provides interfaces for the client (the mobile terminal) and the server to accomplish task offload. At runtime, a particular component of the mobile application can be decoupled from the local execution and migrated to the server for remote execution. The component should re-join the mobile application by integrating the results of the remote execution with the mobile application. Existing works have demonstrated schemes that operate on different levels of dynamic task offload: the operating system level (full process or virtual machine) [58], the method level [6], or the thread level [13].

We have used a method level task offload module: a lightweight version of the Java *Remote Method Invocation* (Java RMI) API [79]. Java RMI enables distributed implementation of Java applications: the methods of remote Java objects can be invoked from other Java virtual machines [80]. This framework was not initially designed for mobile platforms. Based on the existing library, we develop an Android platform compatible

RMI module. Moreover, we extend the original Java RMI library by adding more functions that are useful for mobile method migration.

The module is developed by following three guidelines. (1) It enables method level migration. Other levels of migration usually require additional overhead. For instance, the OS level migration requires identical execution environments on the mobile application side and the server side. The process of virtual machine migration leads to a high cost due to additional management, like context preservation. The thread level migration requires a synchronization mechanism to achieve finer granularity of task offload. (2) It is convenient to deploy. The module requires no change to the underlying mobile operating system and is loosely coupled with the host application. (3) It is highly scalable. The module provides generic APIs that be easily used by developers to support task migration of any non-native method.

Fig. V.4 explains how the module accomplishes method level migration. The migration of a method from the mobile application is essentially to execute the code of the method on the server side Java virtual machine. The server maintains a naming module: application registry, which is similar to the registry in Java RMI. All the methods that can be offloaded need to register with this module. In addition, the server has access to the codebase of these methods (either locally available or accessible via URL protocols). At the client side, if a local object initializes a remote method call, the object invokes the remote method via the remote interface, as if it is calling a local method. Then the underlying migration module sends a query to the server and the server looks up the interface in the registry. If it has been registered, the parameters of the method will be serialized into byte-stream and sent to the server. At the server side, a remote object, which is of the same type with the local object at the client, calls the method. The result of the execution will be returned to the client as byte-stream and de-serialized. By using

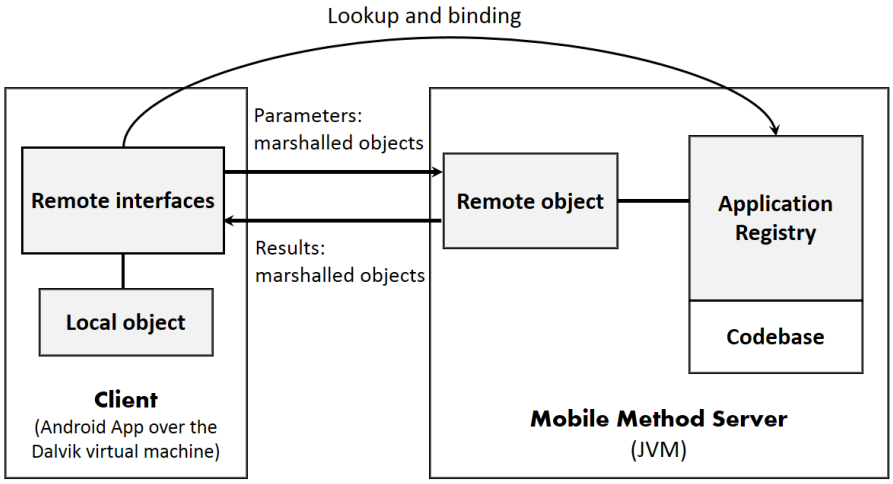


Figure V.4: Mobile application method migration.

this framework, a non-native mobile method can be seamlessly migrated to an external device and later resume from the migration point.

Energy Cost Models

The energy cost models are prediction models used by the runtime solver to evaluate the energy consumption levels of the feasible schemes. As Eq.(V.1) shows, the energy consumption of an execution scheme Ψ consists of the locally consumed energy and the energy consumed by task migration if the scheme contains remote execution. The energy consumption is dominated by the dynamic factors: network condition and user demand. As discussed in Sec. V, we use RSSI to evaluate network condition. The other dynamic factor user demand is non-trivial to define for different applications and scenarios. In our model, we introduce the *User Demand Mapping* approach to evaluate the user input related variables. As Fig. V.5 shows, user demand effect can be mapped to some user input related parameters for local execution schemes, which represents the

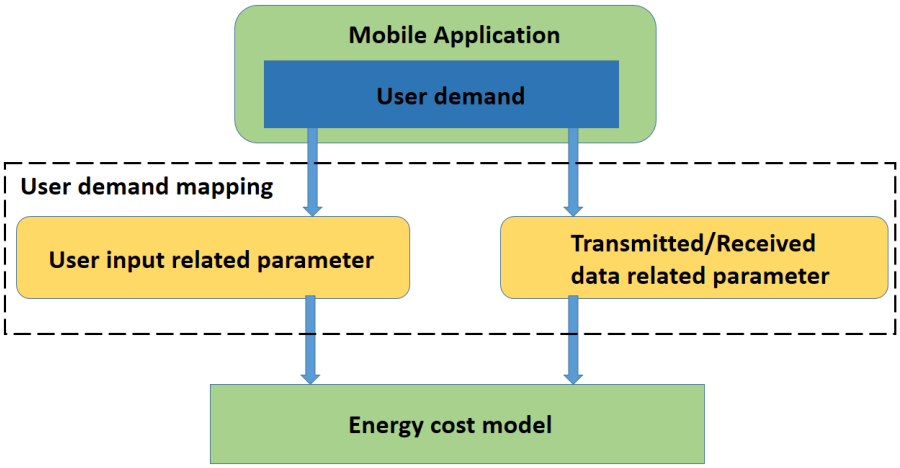


Figure V.5: User demand related input variables.

user related workload. Moreover, for the remote execution schemes, it is mapped to the size of transmitted and received data, which directly determine how much energy is consumed during offloading.

In order to accurately estimate the energy consumption, especially the energy consumed by offloading, we need to exploit the quantitative relationship between the dynamic factors and energy consumption. However, one energy model that applies to one mobile device may not be simply used by another device because the hardware specifications can be very different. Therefore we use profiling and regression techniques [81] to establish the cost models in an offline manner for each scheme.

Energy Model of Local Execution Schemes

Local execution schemes are schemes without task migration. As Fig. V.1 shows, the energy consumption of a local execution scheme is closely related to user demand. The higher the workload assigned by the user, the more energy consumed. We first use

the linear regression model for local energy consumption. Linear regression is a prediction model that describes the relationship between a dependent variable and multiple explanatory variables. The model can be used to predict the value of the dependent variable when it is considered to be linearly related to the explanatory variables. In our study, we assume user demand level can be described by a particular input variable that reflects the user-related workload, denoted by x_w. Our local execution energy model is represented by Eq.(V.4), where the local energy consumption $E_\psi^{\mathscr{L}}$ is the dependent variable, and the user demand variable x_w is the only explanatory variable. ε is the error term. The model is suitable for the applications in which the user demand variable has almost linear impact on energy consumption. In Fig. V.6, we plot the data samples obtained from an image processing application that adds Gaussian blur effect to user specified pictures. In this application, we use the image size as the user demand related variable. Again we have used *PowerTutor* to measure the image processing task. We measure the energy consumed by CPU with different user demand levels. Clearly the linear model describes the relationship between the variables. However, the linear regression model may be biased for some applications, such as the local energy cost of the *N-Queen Puzzle* application, shown in Fig. V.1.

$$E_\psi^{\mathscr{L}}(x_w) = \gamma + \beta x_w + \varepsilon \tag{V.4}$$

Energy Model of Remote Execution Schemes

The energy consumption of a remote execution scheme is dominated by the quality of wireless connection and user demand. To better understand this, let us investigate where the battery energy is drained while offloading a mobile task. First, the task migration and re-integration obviously consume energy, and the amount of energy consumed

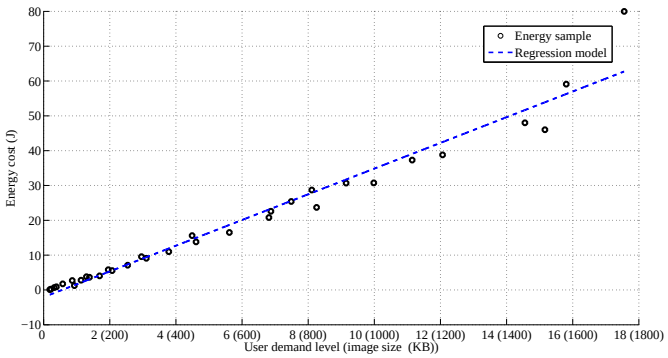


Figure V.6: Linear regression model of the local execution scheme (Gaussian blur application).

is directly related to how many bytes of data have been transmitted and received. This involves two parts: the energy consumed by transmitting the serialized method parameters to the server, and the energy consumed by receiving the serialized result from the server. In addition, the method migration module itself also consumes battery energy. If the size of data transmitted to the server is denoted by w_s, the size of data received by is w_r, and the energy consumption of the runtime solver is E_s, then the total data size is $x_w = w_s + w_r$. The energy consumption of a remote execution scheme can be expressed by the following equation.

$$E_{\psi}^{\mathscr{R}}(x_c, x_w) = E_{\psi}^{'}(x_c, x_w) + E_s \tag{V.5}$$

Here $E_{\psi}^{'}(x_c, x_w)$ represents the energy consumed by data transmission.

In our model, we mainly consider the impact of user demand and network condition, and the solver related energy consumption is regarded as a constant value for a particular computation task. We still first use linear regression to establish a quantitative model. In our case, the dependent variable is the energy consumption of a remote execution

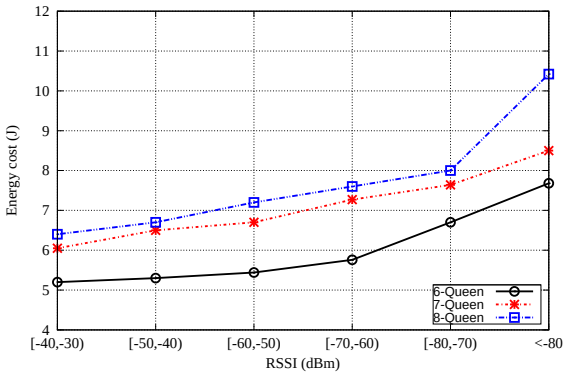


Figure V.7: Impact of signal strength.

scheme ψ and the explanatory variables include network condition and user demand level.

We first profile the energy cost under various conditions with different wireless connection quality levels and user demand levels. We use the total amount of data transmitted and received during offloading as the user demand related input parameter and RSSI as the indicator of the network condition. We used the method migration module, the *N-Queen Puzzle* application, and the energy profiler *PowerTutor* [78] to collect the energy consumption samples. The way we collect the samples is as follows: each sample is corresponding to a scenario with a particular network condition level and a user demand level, and we measure the energy consumption of the critical computation task: the *N-Queen Puzzle*.

In Fig. V.7, we present the average energy cost during offloading, with different RSSI levels. As expected, weaker signal strength usually leads to higher energy cost. Especially, when the RSSI is worse than $-80dBm$, the energy cost increases sharply. Fig. V.8 illustrates the average energy cost of code migration, when the user demand related parameter (size of the transmitted data) is different. Obviously higher demand level would cost more energy.

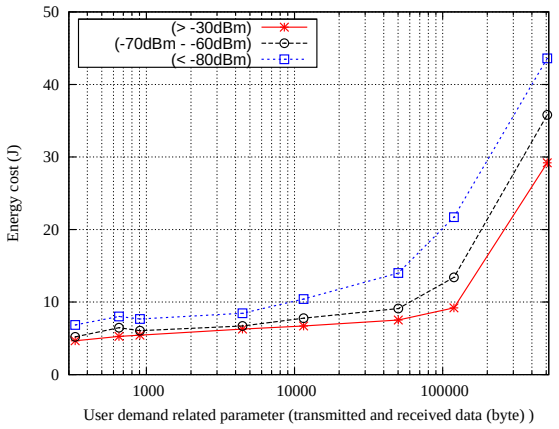


Figure V.8: Impact of the size of transmitted data.

Now we formally define our linear regression model of energy cost $E_{\psi}^{\mathscr{R}}$ of scheme ψ in Eq.(V.6). Here x_c and x_w represent the normalized RSSI value and user demand related parameter (the size of transmitted and received data) respectively. The coefficients in the regression model is obtained based on the least square estimation. For instance, the regression models for the *N-Queen Puzzle* application and the image processing (Gaussian Blur) application are illustrated in Fig. V.9 and Fig. V.10, where the predicted values are close to the actual samples. The coefficients of the remote execution energy model of the *N-Queen Puzzle* application are $\gamma = -2.4872$, $\alpha = -0.1482$ and $\beta = 0.0001$. With the regression models, the energy consumption of a partition scheme at a particular time instance can be predicted. The prediction statistics of the two models are listed in Tab. V.1.

$$E_{\psi}^{\mathscr{R}}(x_c, x_w) = \gamma + \alpha x_c + \beta x_w + \varepsilon \tag{V.6}$$

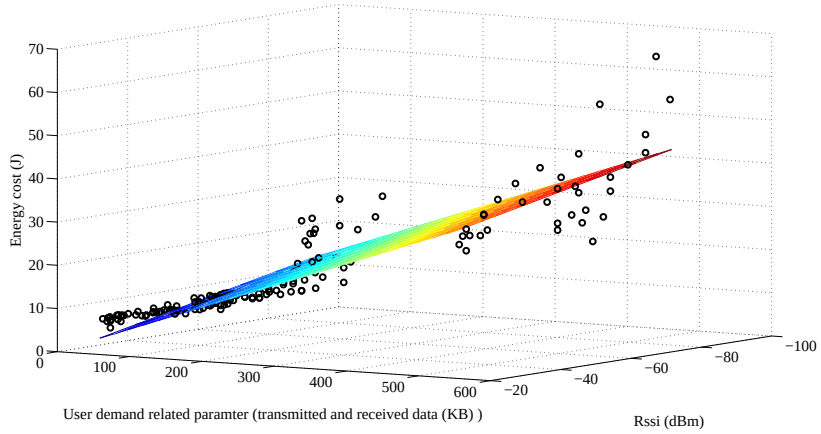


Figure V.9: The linear regression model of the remote execution scheme of the *N-Queen Puzzle* application.

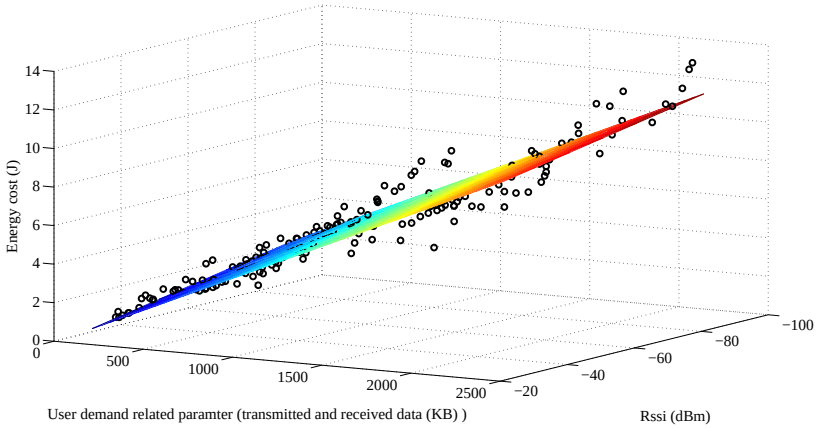


Figure V.10: The linear regression model of the remote execution scheme of the *Image Processing* (*Gaussian blur*) application.

Mobile Applications	R^2	RMSE
N-Queen Puzzle	0.8911	4.3377
Image Processing: Gaussian blur	0.9404	0.7562

Table V.1: Prediction statistics of the linear regression models

Optimized Energy Models

Although linear regression effectively models the relationship between the energy cost and the explanatory variable(s) if they are "linearly related", like the energy model of the local execution scheme of the image processing application, it may fail to accurately predict the dependent variable when the relationship is not linear, for instance, the energy model of the local execution scheme of the *N-Queen Puzzle* application. To solve the problem, we introduce the *regression tree* model [82], a widely adopted predictive model in data mining and machine learning. Unlike the linear regression model, which uses the same linear equation to predict the dependent variable over the entire space of the explanatory variables, the regression tree model partitions the space into multiple smaller sub-places, or cells, and each of the cells is associated with a particular simple predictive model. The regression tree model uses a tree structure to represent the recursive partition of the data space. Each interior node is associated with a classification question, and the edges connecting two nodes represent the answers. By taking different branches under a node, the data space is partitioned by using the criteria specified by the node. A new prediction starts from the root and terminates at a leaf node which represents the simple prediction model for the particular data tuple. The traversal from the root to the leaf is achieved by using the partition rules associated with the interior nodes and branches.

In Fig. V.11, we plot the regression tree energy model of the local execution scheme of the *N-Queen Puzzle* application. In this example, if the user demand level (WL: the number of queens) is greater than or equal to 11, the predicted energy cost of the local execution is $963.4J$, otherwise we traverse to the left child of the root node and keep searching until we find the correct leaf node. Fig. V.12 presents the regression tree model of the energy cost of a remote execution scheme. The partition in this model is based

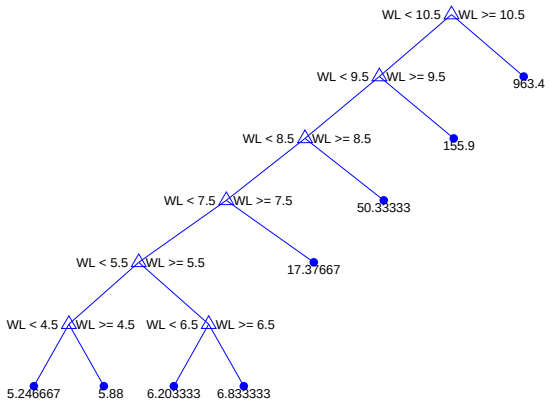


Figure V.11: Regression tree model of the local execution scheme (*N-Queen Puzzle*).

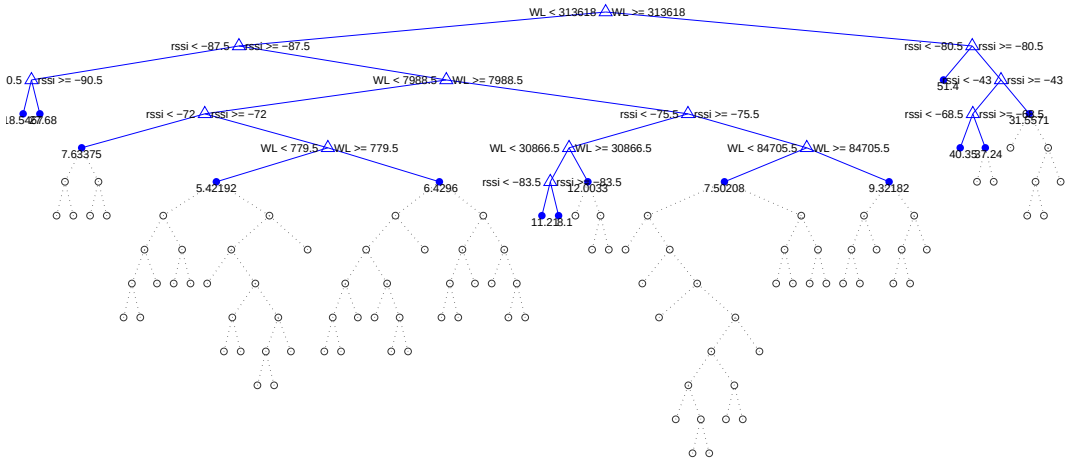


Figure V.12: Regression tree model (partial) of a remote execution scheme (*N-Queen Puzzle*).

on two variables: RSSI and user demand (WL: the total data transmitted/received). The regression errors are listed in Tab. V.2. Compared with the RMSEs of the linear regression models, the RMSEs of the regression tree models are smaller.

Mobile Applications	RMSE
N-Queen Puzzle	1.5291
Image Processing: Gaussian blur	0.0621

Table V.2: Regression errors of the regression tree models

Runtime Solver

The solver determines what execution scheme should be used during the next time interval. The solver relies on a heuristic algorithm to seek the optimal execution scheme from the scheme pool at runtime. The algorithm provides a solution to Problem V.2.

The mechanism of the runtime decision making is illustrated by Fig. V.13. The runtime profiler keeps monitoring RSSI and user demand. At each time instance t_u (the beginning time point of a scheme cycle), the solver collects runtime profiles from the profiler. Then it predicts the energy consumptions of the candidate schemes by using the regression models. If a better scheme exists (the predicted energy cost of the candidate scheme is lower than the current scheme), the current scheme will be replaced by the optimal scheme during this cycle. The solver algorithm is listed in Tab. V.3. Here δ is a constant serving as the tolerance value of decision making.

In our implementation, the solver relies either on a simple linear polynomial calculation (liner regression model) or a tree (regression tree) search, and this calculation is not so frequent. Therefore the energy consumption of the solver is negligible compared with the energy consumption of other components, such as data transmission and object

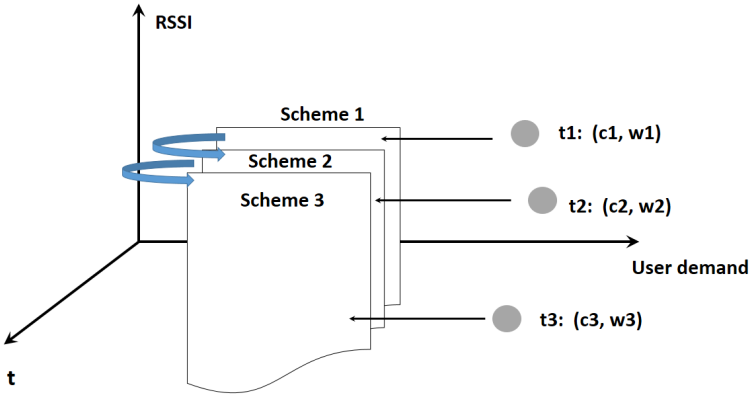


Figure V.13: Runtime solver

serialization/de-serialization. This is why the energy consumption is not regarded as a variable in the model, not like the network condition and user demand. However, when we profile the energy consumption of different schemes, the solver energy consumption (if there is any energy consumed by the solver) is also considered in the profiling process because we measure the total energy consumption that includes the energy consumption of the solver.

At the beginning of each cycle:
1) Collect the runtime profile $p(x_c(t_u), x_w(t_u))$
2) Predict the energy consumption of each candidate scheme
3) Find the optimal scheme:
$\psi^* = \arg\min_{\psi_i \in \Psi} E_{\psi_i}$
4) Make runtime decision
if $E_{\psi^*} + \delta < E_{\psi(t_u)}$ ($\psi(t_u)$: current scheme)
replace $\psi(t_u)$ with ψ^*
invoke the method migration model if necessary
end if

Table V.3: Execution scheme selection algorithm.

Evaluation

In this section we evaluate the performance of our offload system by using real world applications.

Experimental Setups

Benchmark Applications

We use the following four benchmark mobile applications.

- *N-Queen Puzzle*. The user specifies the dimensions of the chess board and the results (the positions of the queens) are computed by the mobile application and displayed on the board. The critical task of the application is the N-Queen placement, which can either be solved locally or remotely. The user demand level of an execution is determined by the dimensions of the chess board (Level 1 is the user demand level of the 4-Queen puzzle. When the user demand level increases by one, the chess board dimension increases by one).

- *Image Processing I* - Gaussian blur effect. The application can add Gaussian blur effect to any user specified image. The critical task of the application is the Gaussian blurring process, which can either be executed by the mobile terminal or by the server. The user demand level of an execution is determined by the dimensions of the image.

- *Image Processing II* - Sharpen effect. The application can add sharpen effect to any user specified image. Similarly, the critical task is the sharpen process, which can be executed locally or remotely. The user demand level of an execution is also related to the dimensions of the image.

- *Ant Colony Game* - A real time strategy game in which users train a group of ants to collect food scattered around the map. The computation-intensive task is the ant-colony optimization algorithm, which helps find the optimal path to the food. The task contains two independent components that can be executed locally or remotely: the pheromone update component and the solution producing component. The user demand level of an execution is related to the size of the map (Level 1 is the user demand level of a 4×4 map. When the user demand level increases by one, the map dimension increases by one).

In the image processing applications we use the following two groups of images (Portable Network Graphics (PNG) format).

User demand level	1	2	3	4	5	6	7	8	9
Size (KB)	13	35	96	263	411	642	1001	1545	2371
Width (pixels)	56	146	243	404	505	631	788	984	1229
Height (pixels)	88	92	152	253	316	394	492	615	768

Table V.4: User demand description of the image processing applications: Group I

User demand level	1	2	3	4
Size (KB)	37	20	678	1169
Width (pixels)	154	108	800	922
Height (pixels)	116	82	626	692

Table V.5: User demand description of the image processing applications: Group II

Experiment Platforms

We use an HTC Inspire 4G smartphone as our mobile terminal. The energy consumption of the mobile terminal is measured by a power profiling tool *PowerTutor* [78].

The tool can be used to measure the energy consumed by different components on a mobile platform, such as CPU, WiFi and monitor, of a particular mobile application. In our experiments, we only measure the energy consumed by CPU and WiFi. Every data point collected in the experiments is an averaged value of at least five samples.

We deploy two types of back-end platforms:

- A *Lenovo* T400 laptop with *Intel* Core 2 Duo CPU ($2.26GHz$) and $3GB$ memory. We deploy a Java based server program on a Windows 7 operating system, which can respond to the remote execution requests from the mobile terminal.

- An *Oracle* virtual machine (*Virtual Box*) instance with quad-core CPU and $8GB$ memory. The server program is deployed on an Android operating system, so it can process the Android platform dependent program execution.

The two back-end servers use distinct Java environments: the regular Java virtual machine and the Dalvik virtual machine. This is to prove that the proposed system is compatible with different platforms. The advantage of using Dalvik virtual machine at the server side is that even some Android dependent methods can be seamlessly offloaded. The serialized objects passed between the client and server can be parsed by differen types of servers.

Offload Decisions

We first study how the system makes offload decisions at runtime in dynamic environments. The benchmark applications used in this test are *N-Queen Puzzle*, *Image Processing II* and *Ant Colony Game*. The *N-Queen Puzzle* and the *Image Processing II* applications have two execution schemes: a scheme with local execution and a scheme with remote execution. The *Ant Colony Game* application has four schemes: *Scheme I*

is the local execution scheme. *Scheme II* is an execution scheme that executes the two components of the optimization algorithm remotely. *Scheme III* is an execution scheme that only executes the solution producing component remotely. *Scheme IV* is an execution scheme that only executes the pheromone update component remotely. We have collected the runtime decisions (local execution or remote execution of the computation-intensive components) across different RSSI levels and user demand levels. For instance, if the N-Queen placement algorithm or sharpen effect algorithm is solved by the mobile terminal, the decision is called local execution decision, otherwise, it is called remote execution decision. The decision results of the first two applications are plotted in Fig. V.14 and Fig. V.15. From the empirical observations we can easily make the following conclusion: if the quality of the wireless connection is poor and the user demand level is low, it is highly possible the computation is conducted locally. In contrast, a task with high user demand level with excellent network condition is usually offloaded and executed remotely. Sometimes even if the received signal strength is weak, the task is still executed remotely because the local execution consumes too much battery energy. In Fig. V.16 we compare the energy consumption of the dynamic-aware execution and several static execution schemes (*Scheme I* - *Scheme III*) of the Ant-colony optimization algorithm of the *Ant Colony Game* application. From the results we find that the energy consumption of the dynamic-aware execution outperforms the other schemes.

Energy Consumption

Now we evaluate the energy consumption of the benchmark applications running with the offload system, under three RSSI levels: excellent (RSSI I: $< -40dBm$), fair (RSSI II: $-40dBm--79dBm$) and poor (RSSI III: $-85dBm--80dBm$). The mobile applications repeatedly work at each user demand level, and we measure the average

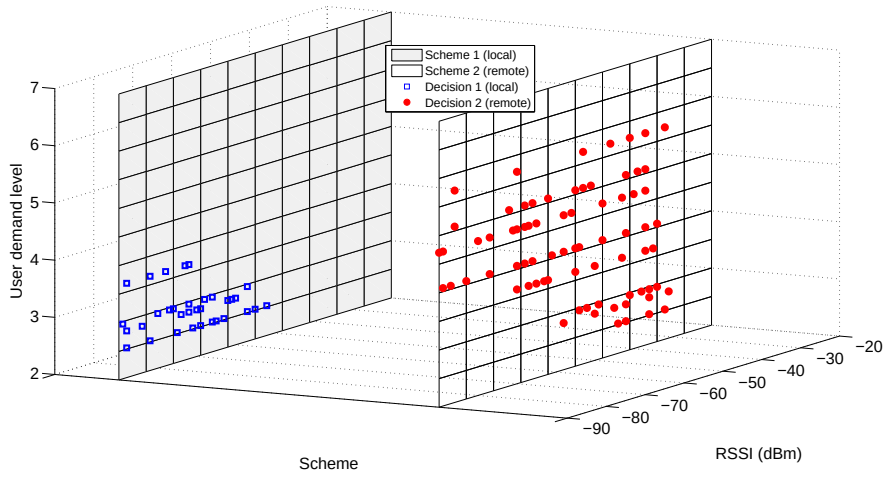


Figure V.14: The offload decisions in various scenarios: *N-Queen Puzzle*.

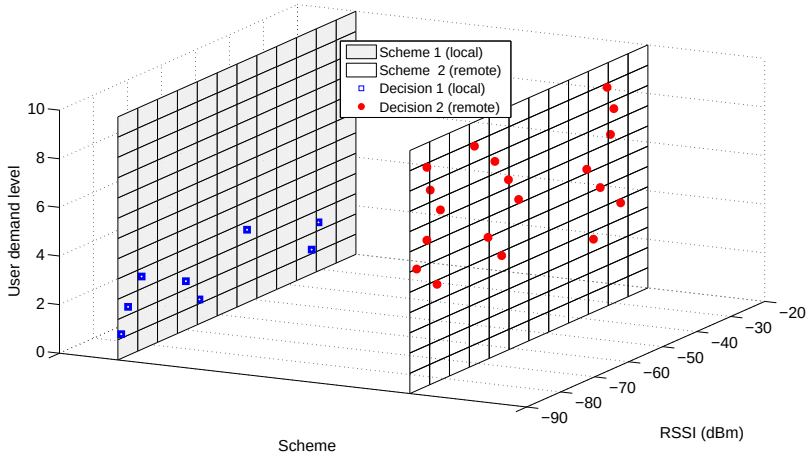


Figure V.15: The offload decisions in various scenarios: *Image Processing II*.

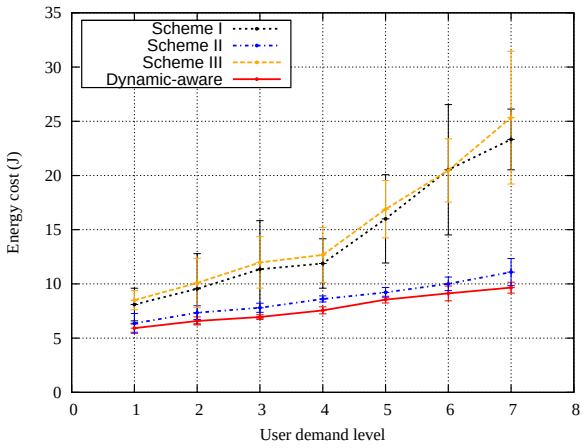


Figure V.16: Comparison between the dynamic-aware execution and the static execution schemes (the *Ant-colony Optimization* algorithm of *Ant Colony Game*).

energy consumption. The user demand levels of the image processing applications are listed in Tab. V.4. The results in Fig. V.17 - Fig. V.20 imply that when the user demand level is low, the energy consumption with offload system is very close to the energy consumption of local executions because the tasks are usually executed locally in such scenarios, which is consistent with the runtime offload decision observations. When the user demand level is high, the advantage of remote execution is obvious: the energy consumption is lower than the local executions. We also notice the effect of wireless network condition. Poor quality of a wireless connection usually results in higher energy consumption. Notably, the poor connection quality ($-80dBm - -85dBm$) may greatly degrade the performance of the offload system.

Execution Time

Reduced execution time is another advantage of a mobile application offload system. Because the back-end servers are usually resource-rich platforms, the execution time of a computation-intensive task is expected to be less than the local execution. In our

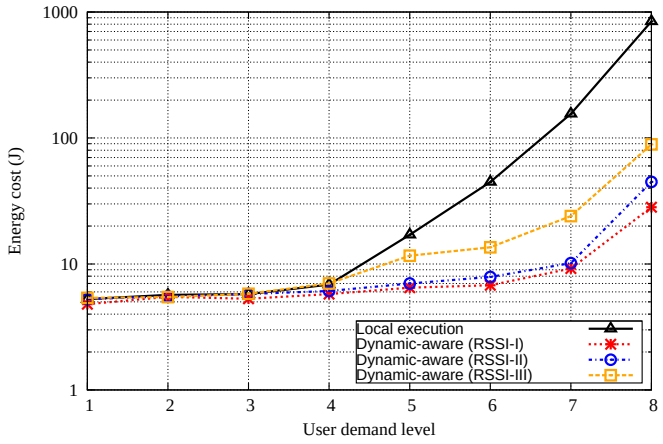


Figure V.17: Energy consumption: *N-Queen Puzzle*

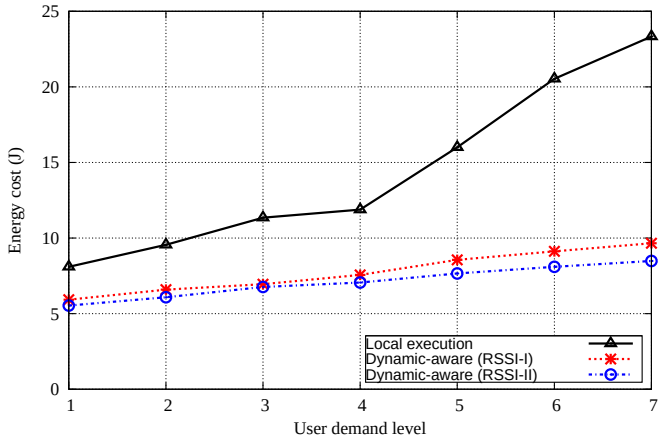


Figure V.18: Energy consumption: *Ant Colony Game*

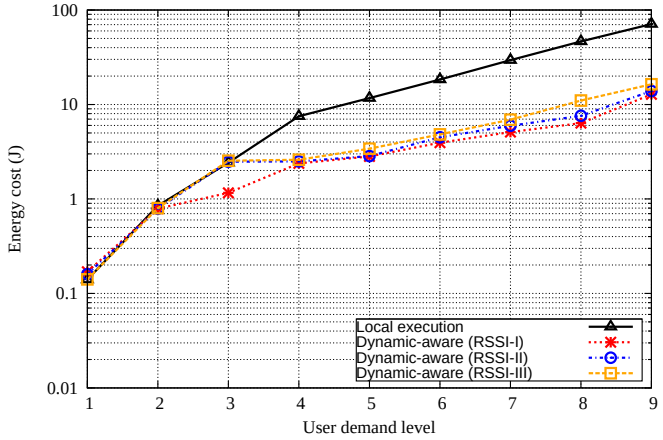


Figure V.19: Energy consumption: *Image Processing I*

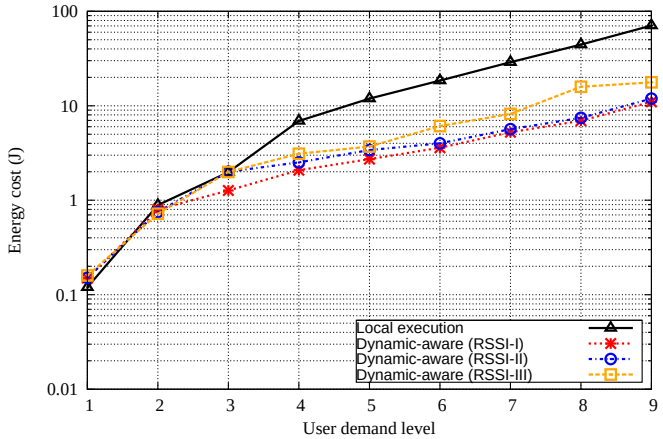


Figure V.20: Energy consumption: *Image Processing II*

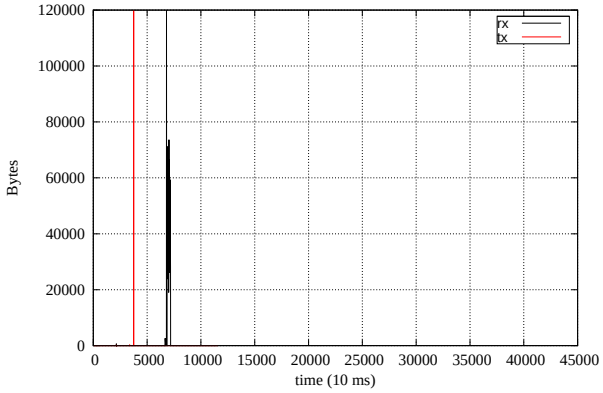
evaluation, the execution time includes the time to execute the task remotely on the server and the time to transfer to/from the server. Next we evaluate the performance of our system. The user demand levels of the image processing applications are listed in Tab. V.4.

First we use *Image Processing II* to investigate data transmission with different wireless connections: a stable connection with high RSSI value and an intermittent connection with very low RSSI value. In the test, the mobile terminal sends an image to the server, then the server processes the image and sends the result back. We collect the data sent and received in each time slot ($10ms$). The transmission behaviors in Fig. V.21 show that when the quality of the network connection is high, the task migration is accomplished shortly after the request is sent to the server. However, when the connection is weak, it takes much longer to receive the processed image. In addition, it costs much more time to establish a connection with the server under the poor network condition.

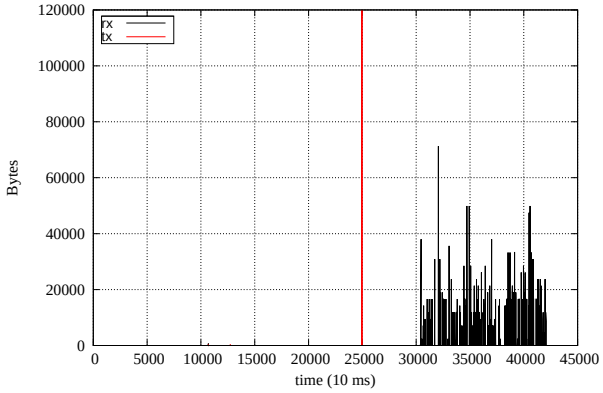
We plot the average execution times of the benchmark applications with different user demand levels and RSSI levels in Fig. V.22 - Fig. V.24. Again the execution time of a task is close to the local execution when the workload is low. When the workload is high, the performance of execution time with the offload system obviously outperforms the local executions. We also notice the impact of network condition. A wireless connection with poor received signal strength could significantly increase the execution time.

Comparison with Heuristic Offload Algorithm

Next we compare the performance of our offload algorithm with a heuristic offload algorithm, which is unaware of the dynamics and always prefers remote executions. We investigate the average energy consumption of the algorithms under four different RSSI



(a) Data transmission with stable wireless connection.



(b) Data transmission with intermittent wireless connection.

Figure V.21: Data transmission under different network conditions (*Image Processing II*).

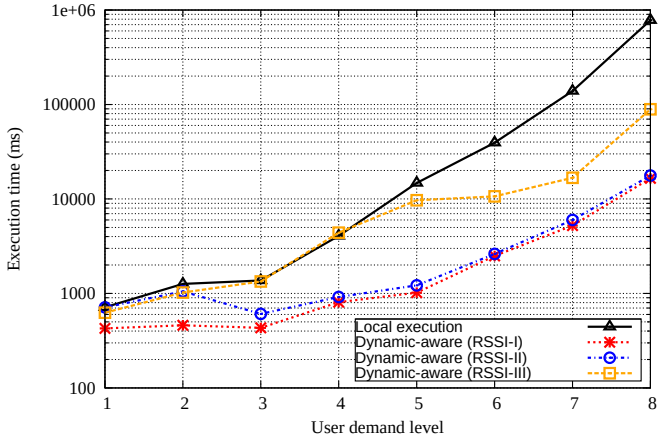


Figure V.22: Execution time: *N-Queen Puzzle*

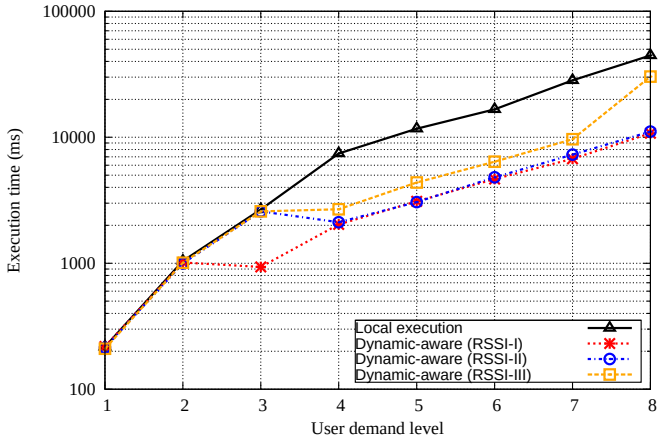


Figure V.23: Execution time: *Image Processing I*

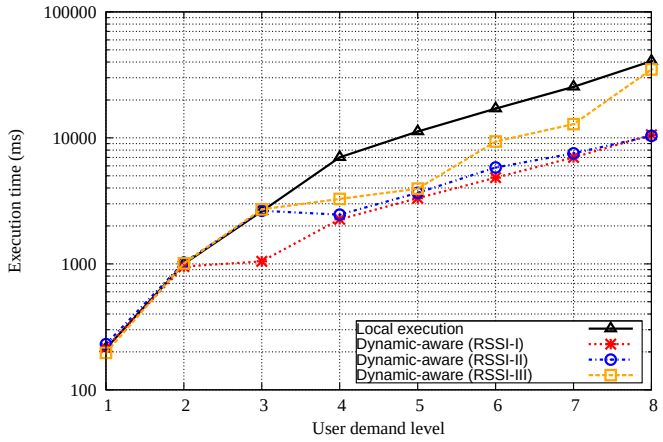
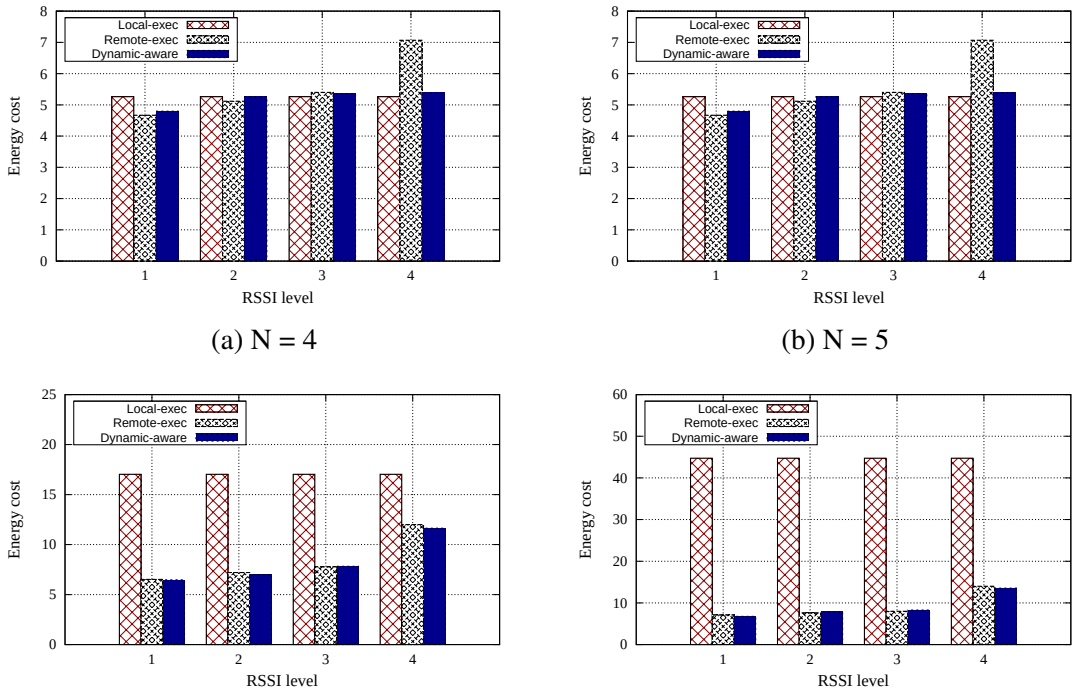


Figure V.24: Execution time: *Image Processing II*

levels: 1) $-25dBm--45dBm$; 2) $-45dBm--65dBm$; 3) $-65dBm--85dBm$ and 4) $\leq -85dBm$. Especially, the fourth level represents connections with very weak signal strength, such as the intermittent wireless connections. Packet transmissions with such a connection usually consume much more energy. We test the *N-Queen Puzzle* application and the *Image Processing I* application which uses the images listed in Tab. V.5.

We plot the test results in Fig. V.25 and Fig. V.26. Overall, our offload algorithm outperforms the other two execution strategies. Even under the worst case scenarios, the performance of the proposed algorithm is very close to the best performing algorithm. With a high user demand level, the advantage of offloading is impressive. We have also noticed that the heuristic algorithm leads to high energy consumption with intermittent connections. This is possibly due to the high energy cost of packet transmissions and the re-transmissions when some initial attempts fail. This also demonstrates the importance of adopting an intelligent offload strategy, which can flexibly adjust the offload decision according to runtime dynamics.



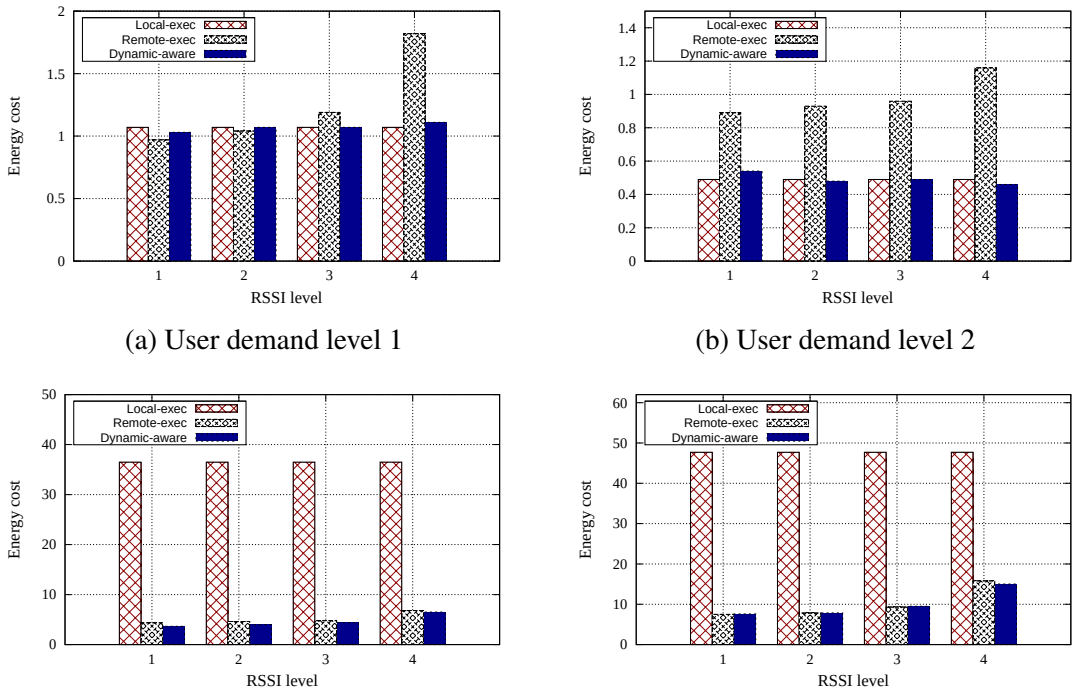
(a) N = 4 (b) N = 5

(c) N = 8 (d) N = 9

Figure V.25: Energy consumption of *N-queen Puzzle*.

Discussion

The resource constraints of mobile terminals have compromised the quality of service of mobile applications. To solve the problem, the task offload approach is proposed to migrate the intensive computations to some external devices, in order to extend the capabilities of the mobile platforms and reduce resource consumption. However, it is challenging to achieve cost-efficient offloading with the presence of dynamic factors, such as network condition and user demand. In this chapter, we have discussed a dynamic-aware mobile application offload solution. The proposed system aims to minimize the energy consumption of offloading by adjusting execution schemes at runtime. We present a theoretical formulation for the offloading problem and propose a heuristic algorithm to find the solution. In order to predict the cost of offloading in dynamic environments, we propose cost models by using profiling and regression techniques. The



(a) User demand level 1

(b) User demand level 2

(c) User demand level 3

(d) User demand level 4

Figure V.26: Energy consumption of *Image Processing I*.

execution decisions are made by evaluating the levels of energy consumption of different schemes using the cost models.

The current objective of our solution is to reduce energy consumption. However, other quality of service metrics, like execution time, are also very important to the performance of a mobile application. It is possible that an energy-efficient solution may not result in the optimal execution time. The experimental results indicate that the execution time of a task is also related to user demand and network condition, which implies the cost models can be established based on execution time. Therefore the proposed system can also be adjusted to find optimal execution schemes that yield the best executions time.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This dissertation explores the robust resource management solutions for mobile wireless systems in dynamic environments. In this chapter, we review the work presented in the previous chapters and discuss several future research directions.

Conclusions

Resource constraints of mobile wireless systems can mainly be attributed to two problems: mobile data congestion in wireless networks and limited hardware capabilities of mobile terminals. The key to solving the data congestion problem in wireless network infrastructure is to balance demand and supply to achieve optimal and fair resource allocation, which is nontrivial due to the complex nature of wireless communication and network structure. Notably, more challenges arise under dynamic environments. Time-varying wireless channel keeps reshaping the solution space of the resource management problem. Furthermore, the delay problem challenges the coordination of resource providers and consumers. The resource constraint problem of mobile platforms awaits discovery of new solutions that transiently and seamlessly transfer the computation-intensive components from mobile terminals to resource-rich devices. A challenge to this task is the cost of offloading changes with user demands and network condition.

In this dissertation, we propose a two-tier dynamic-oriented resource management solution towards achieving efficient resource utilization in mobile wireless systems. Overall, our solution framework is established along two tiers. The first tier involves

cross-layer (the MAC layer to the Transport layer) optimization based techniques (Chapter III and IV) that improve the network performance by efficiently utilizing wireless resources under dynamic scenarios. The second tier of scheme operates on the application layer. We propose a mobile code offload approach (Chapter V) that leverages external resources to augment the capabilities of mobile wireless systems. We summarize our work as follows.

- The robust joint congestion control cross-layer control algorithm (Chapter III) solves the resource allocation problem in time-varying multi-hop wireless networks with presence of feedback delay. This approach relies on capacity space projection to form a new capacity space based on the slow time scale channel capacity, which substantially reduces the delay effect. Unlike most of the exiting joint congestion control and scheduling algorithms, our algorithm offers robust management with time delay.

- The time scale decomposition based joint congestion control and scheduling approach (Chapter IV) aims to maximize the time integral of the aggregated substantial network utility. In this approach the resource allocation is operated over the critical time scale. Meanwhile, a penalty aware scheduling that uses a penalty function to estimate the level of rate over-provisioning can effectively track the fast time scale channel dynamics.

- The dynamic-aware mobile application offload system (Chapter V) extends the capabilities of resource-constrained mobile platforms. The system enables seamless remote execution of mobile applications. The proposed energy cost models characterize energy consumption under varying user demands and network connection. The execution schemes of a mobile application are selected intelligently at runtime to achieve minimal energy consumption. Compared with the previous

mobile code offload systems, we have considered network and user dynamics, and our approach can be easily applied to existing mobile applications.

Future Research

Like the wireless sensor networks challenged our understanding of computing in the early twenty-first century, again we are facing a new era of computing evolution featured with smart phones, tablets and wearable computing devices, which offer enormous possibilities to improve our life. Meanwhile the ever-increasing requirements of mobile computing on computational resources urge us to explore more intelligent strategies of managing the mobile ecosystems. In the rest of this chapter, we outline several interesting and promising directions of future research.

- **Provisioning qualify of service for diverse requirements.** Our dynamic-aware mobile application offload system is a prototype of qualify of service oriented hybrid mobile computing platform, which has mainly targeted energy saving by offloading mobile applications to external servers. However, the requirements on quality of server may be diverse. As pointed out by Satyanarayanan et al. [58], delay, jitter, and bandwidth could all influence the design of an offload system. For example, some mobile applications are sensitive to latency and jitter, like online games and live video, therefore energy consumption may not be the system's first priority. Many research results have demonstrated the possibility of provisioning other types of quality of service during offloading. The *Odessa* [52] system enables adaptive offloading of interactive perception mobile applications, in which latency is considered as an important metric for adaption of offloading. By redefining the cost models, our technique can be extended to provision quality of service for different user requirements.

- **Recognizing interactions in a mobile wireless system.** Mobile applications are not stand-alone applications. As the existing studies [83, 84, 75] reveal, in the lifetime of a mobile application, a mobile application might interact closely with WiFi networks or cellular networks. Very similar to what we have discussed in Chapter III and Chapter IV, the complex cross-layer interactions may largely affect the overall performance of a mobile application. Some interesting research problems emerge with this. For example, how to optimize the way a mobile application accesses the networks to reduce energy consumption and how to reduce the misunderstandings between two different software systems (network and mobile application) to avoid runtime failures. This research could be extremely helpful for designing high-performance network protocols and mobile applications.

- **Composition model based mobile cloud computing.** Mobile cloud computing is a new trend of mobile application development, particularly the composition development model [69, 85], which carries a similar concept with our dynamic-aware offload system. In this model, a mobile application can contain some reusable service components, which can be shipped to cloud on demand. Some interesting problems in this area include how to design an architecture that efficiently integrates the cloud service and mobile applications and how to enhance the performance by introducing parallelism. Furthermore, introducing dynamic-aware modules into mobile cloud computing applications would be a bright research direction.

BIBLIOGRAPHY

[1] Apple Reports. http://www.apple.com/pr/library/2012/10/25apple-reports-fourth-quarter-results.html, October 2012.

[2] Google official blog. http://googleblog.blogspot.com, March 2013.

[3] Cisco Inc. Cisco visual networking index: Global mobile data traffic forecast update, 20112016, February 2012.

[4] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile data offloading: how much can wifi deliver? In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 26:1–26:12, New York, NY, USA, 2010. ACM.

[5] Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, Z. Morley Mao, and Xu Chen. Comet: code offload by migrating execution transparently. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 93–106, Berkeley, CA, USA, 2012. USENIX Association.

[6] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[7] Abhinav Pathak, Y. Charlie Hu, and Ming Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM.

[8] Cunqing Hua, Song Wei, and Rong Zheng. Robust channel assignment for link-level resource provision in multi-radio multi-channel wireless networks. In *IEEE International Conference on Network Protocols, ICNP 2008.*, pages 157–166, Oct. 2008.

[9] Gaurav Sharma, Ravi R. Mazumdar, and Ness B. Shroff. On the complexity of scheduling in wireless networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, MobiCom '06, pages 227–238, New York, NY, USA, 2006. ACM.

[10] I.-H. Hou, V. Borkar, and P.R. Kumar. A theory of qos for wireless. In *Proceedings of the 28th IEEE International Conference on Computer Communications, INFOCOM 2009*, pages 486 –494, April 2009.

[11] Longbi Lin, N.B. Shroff, and R. Srikant. Asymptotically optimal energy-aware routing for multihop wireless networks with renewable energy sources. *IEEE/ACM Transactions on Networking*, 15(5):1021–1034, Oct.

[12] Liang Dai, Yuan Xue, Bin Chang, Yanchuan Cao, and Yi Cui. Optimal routing for wireless mesh networks with dynamic traffic demand. *Mobile Networks and Applications*, 13(1-2):97–116, April 2008.

[13] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[14] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.

[15] L. Chen, S. H. Low, M. Chiang, and J. C. Doyle. Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM 2006*, pages 1 –13, April 2006.

[16] Lijun Chen, S.H. Low, and J.C. Doyle. Joint congestion control and media access control design for ad hoc wireless networks. In *Proceedings of the 24th IEEE International Conference on Computer Communications, INFOCOM 2005*, volume 3, pages 2212 – 2222 vol. 3, March 2005.

[17] Mung Chiang. Balancing transport and physical layers in wireless multihop networks: jointly optimal congestion control and power control. *IEEE Journal on Selected Areas in Communications*, 23(1):104 – 116, Jan. 2005.

[18] Xiaojun Lin and N.B. Shroff. The impact of imperfect scheduling on cross-layer rate control in wireless networks. In *Proceedings of the 24th IEEE International Conference on Computer Communications, INFOCOM 2005*, volume 3, pages 1804 – 1814 vol. 3, March 2005.

[19] Jang-Won Lee, M. Chiang, and R.A. Calderbank. Jointly optimal congestion and contention control based on network utility maximization. *IEEE Communications Letters*, 10(3):216 – 218, March 2006.

[20] Ioana Giurgiu, Oriana Riva, and Gustavo Alonso. Dynamic software deployment from clouds to mobile devices. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 394–414, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[21] Steven H. Low and David E. Lapsley. Optimization flow control i: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[22] Samarth H. Shah, Kai Chen, and Klara Nahrstedt. Dynamic bandwidth management in single-hop ad hoc wireless networks. *Mobile Networks and Applications*, 10:199–217, February 2005.

[23] Yuan Xue, Baochun Li, and Klara Nahrstedt. Optimal resource allocation in wireless ad hoc networks: a price-based approach. *IEEE Transactions on Mobile Computing*, 5(4):347 – 364, April 2006.

[24] Kamal Jain, Jitendra Padhye, Venkata N. Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, MobiCom '03, pages 66–80, New York, NY, USA, 2003. ACM.

[25] I-Hong Hou and P. R. Kumar. Utility-optimal scheduling in time-varying wireless networks with delay constraints. In *Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '10, pages 31–40, New York, NY, USA, 2010. ACM.

[26] Shihuan Liu, Lei Ying, and R. Srikant. Throughput-optimal opportunistic scheduling in the presence of flow-level dynamics. *IEEE/ACM Transactions on Networking*, 19(4):1057–1070, Aug. 2011.

[27] Sujay Sanghavi, Loc Bui, and R. Srikant. Distributed link scheduling with constant overhead. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '07, pages 313–324, New York, NY, USA, 2007. ACM.

[28] V. J. Venkataramanan and Xiaojun Lin. On wireless scheduling algorithms for minimizing the queue-overflow probability. *IEEE/ACM Transactions on Networking*, 18(3):788–801, June, 2007.

[29] Changhee Joo, Xiaojun Lin, and N.B. Shroff. Understanding the capacity region of the greedy maximal scheduling algorithm in multi-hop wireless networks. In *Proceedings of the 27th IEEE International Conference on Computer Communications, INFOCOM 2008*, pages 1103–1111, 2008.

[30] L. Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, Dec. 1992.

[31] Xinzhou Wu and R. Srikant. Regulated maximal matching: A distributed scheduling algorithm for multi-hop wireless networks with node-exclusive spectrum sharing. In *Proceedings of 44th IEEE Conference on Decision and Control, and the European Control Conference. CDC-ECC '05.*, pages 5342–5347, Dec. 2005.

[32] P. van de Ven, S. Borst, and S. Shneer. Instability of maxweight scheduling algorithms. In *Proceedings of the 28th IEEE International Conference on Computer Communications, INFOCOM 2009*, pages 1701–1709, April 2009.

[33] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Algorithmic aspects of capacity in wireless networks. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '05, pages 133–144, New York, NY, USA, 2005. ACM.

[34] Liang Dai, Yuan Xue, Bin Chang, and Yi Cui. Throughput optimization routing under uncertain demand for wireless mesh networks. In *Proceedings of IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, MASS 2007*, pages 1–11, Oct. 2007.

[35] Srisankar Kunniyur and R. Srikant. End-to-end congestion control schemes: utility functions, random losses and ecn marks. *IEEE/ACM Transactions on Networking*, 11(5):689–702, Oct. 2003.

[36] Mung Chiang, S.H. Low, A.R. Calderbank, and J.C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255 –312, Jan. 2007.

[37] A. Eryilmaz and R. Srikant. Joint congestion control, routing, and mac for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1514–1524, Aug. 2006.

[38] Xiaojun Lin and N.B. Shroff. Joint rate control and scheduling in multihop wireless networks. In *Proceedings of 43rd IEEE Conference on Decision and Control, CDC 2004*, volume 2, pages 1484–1489, Dec. 2004.

[39] Leonidas Georgiadis, Michael J. Neely, and Leandros Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1):1–144, April 2006.

[40] Xiaojun Lin, N.B. Shroff, and R. Srikant. A tutorial on cross-layer optimization in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1452–1463, Aug. 2006.

[41] A. Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. *IEEE/ACM Transactions on Networking*, 15(6):1333–1344, Dec. 2007.

[42] G. Sharma, N.B. Shroff, and R.R. Mazumdar. Joint congestion control and distributed scheduling for throughput guarantees in wireless networks. In *Proceedings*

of 26th IEEE International Conference on Computer Communications, INFOCOM 2007, pages 2072–2080, May 2007.

[43] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. In *Proceedings of the 27th IEEE International Conference on Computer Communications, INFOCOM 2008*, pages 619–627, April 2008.

[44] J.J. Jaramillo and R. Srikant. Optimal scheduling for fair resource allocation in ad hoc networks with elastic and inelastic traffic. *IEEE/ACM Transactions on Networking*, 19(4):1125–1136, Aug. 2011.

[45] Fan Qiu, Jia Bai, and Yuan Xue. Optimal rate allocation in wireless networks with delay constraints. *Ad Hoc Networks*, 13, Part B(0):282 – 295, 2014.

[46] Rayadurgam Srikant. *The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications)*. SpringerVerlag, 2004.

[47] L. Bui, A. Eryilmaz, R. Srikant, and Xinzhou Wu. Asynchronous congestion control in multi-hop wireless networks with maximal matching-based scheduling. *IEEE/ACM Transactions on Networking*, 16(4):826–839, Aug. 2008.

[48] Lei Ying, R. Srikant, A. Eryilmaz, and G.E. Dullerud. Distributed fair resource allocation in cellular networks in the presence of heterogeneous delays. *IEEE Transactions on Automatic Control*, 52(1):129–134, Jan. 2007.

[49] Guohua Zhang, Yiyu Wu, and Yonghe Liu. Stability and sensitivity for congestion control in wireless mesh networks with time varying link capacities. *Ad Hoc Netw.*, 5(6):769–785, Aug. 2007.

[50] M.Z. Win, P.C. Pinto, and L.A. Shepp. A mathematical theory of network interference and its applications. *Proceedings of the IEEE*, 97(2):205–230, Feb. 2009.

[51] Byung-Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th conference on Hot topics in operating systems*, HotOS'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[52] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[53] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):19–26, Jan. 1998.

[54] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 48–63, New York, NY, USA, 1999. ACM.

[55] Jason Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 61–66, May 2001.

[56] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 272–285, New York, NY, USA, 2007. ACM.

[57] C. Young, Y.N. Lakshman, T. Szymanski, J. Reppy, D. Presotto, R. Pike, G. Narlikar, S. Mullender, and Eric Grosse. Protium, an infrastructure for partitioned applications. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 47–52, May 2001.

[58] Mahadev Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, Oct.-Dec. 2009.

[59] OSGI Alliance. http://www.osgi.org/main/homepage, March 2013.

[60] S. Toumpis and A.J. Goldsmith. Capacity regions for wireless ad hoc networks. *IEEE Transactions on Wireless Communications*, 2(4):736 – 748, July 2003.

[61] M.J. Neely, E. Modiano, and C.E. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE Journal on Selected Areas in Communications*, 23(1):89–103, Jan 2005.

[62] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 121–132, New York, NY, USA, 2004. ACM.

[63] J. K. Hale and S. M. Verduyn Lunel. *Introduction to functional-differential equations*, volume 99 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1993.

[64] M. Jankovic. Control lyapunov-razumikhin functions for time delay systems. In *Proceedings of the 38th IEEE Conference on Decision and Control*, volume 2, pages 1136 –1141, 1999.

[65] M.J. Neely, E. Modiano, and Chih ping Li. Fairness and optimal stochastic control for heterogeneous networks. *IEEE/ACM Transactions on Networking*, 16(2):396–409, 2008.

[66] S. Sarkar and L. Tassiulas. End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 1, pages 564–569 Vol.1, 2003.

[67] H.K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[68] Yan Gu, V. March, and Bu-Sung Lee. Gmoca: Green mobile cloud applications. In *Proceedings of the First International Workshop on Green and Sustainable Software (GREENS)*, pages 15–20, June 2012.

[69] Fangming Liu, Peng Shu, Hai Jin, Linjie Ding, Jie Yu, Di Niu, and Bo Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Communications*, 20(3):14–22, 2013.

[70] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013.

[71] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: A partition scheme. In *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '01, pages 238–246, New York, NY, USA, 2001. ACM.

[72] Yong Li, Guolong Su, Pan Hui, Depeng Jin, Li Su, and Lieguang Zeng. Multiple mobile data offloading through delay tolerant networks. In *Proceedings of the 6th ACM Workshop on Challenged Networks*, CHANTS '11, pages 43–48, New York, NY, USA, 2011. ACM.

[73] Cong Shi, Vasileios Lakafosis, Mostafa H. Ammar, and Ellen W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, pages 145–154, New York, NY, USA, 2012. ACM.

[74] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[75] Ning Ding, Daniel Wagner, Xiaomeng Chen, Abhinav Pathak, Y. Charlie Hu, and Andrew Rice. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 29–40, New York, NY, USA, 2013. ACM.

[76] Chi-Yu Li, Chunyi Peng, Songwu Lu, and Xinbing Wang. Energy-based rate adaptation for 802.11n. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 341–352, New York, NY, USA, 2012. ACM.

[77] M. Sauter. *From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband*. Wiley Online Library: Books. Wiley, 2010.

[78] Lide Zhang, B. Tiwana, R.P. Dick, Zhiyun Qian, Z.M. Mao, Zhaoguang Wang, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 105–114, 2010.

[79] Java RMI tutorials. http://docs.oracle.com/javase/tutorial/rmi, January 2014.

[80] Java RMI Home. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, January 2014.

[81] J. Cohen. *Applied Multiple Regression/correlation Analysis for the Behavioral Sciences*, volume 1. Routledge, 2003.

[82] L. Breiman. *Classification and regression trees*. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[83] Bo Zhao, Byung Chul Tak, and Guohong Cao. Reducing the delay and power consumption of web browsing on smartphones in 3g networks. In *Proceedings of 31st International Conference on Distributed Computing Systems (ICDCS)*, pages 413–422, 2011.

[84] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Profiledroid: Multi-layer profiling of android applications. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 137–148, New York, NY, USA, 2012. ACM.

[85] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, June 2011.