Geometric Modeling and Analysis of Dynamic Resource Allocation Mechanisms

by

Matthew Secor

B.S., University of California, Berkeley (1994)
S.M., Massachusetts Institute of Technology (1996)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2001

Author: ________________________________
Department of Electrical Engineering and Computer Science
February 5, 2001

Certified by: ________________________________
George Verghese
Professor of Electrical Engineering
Thesis Supervisor

Accepted by: [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Geometric Modeling and Analysis of Dynamic Resource Allocation Mechanisms

by
Matthew Secor

Submitted to the Department of Electrical Engineering and Computer Science
on February 5, 2001, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The major contribution of this thesis is the investigation of a specific resource allocation optimization problem whose solution has both practical application as well as theoretical interest. It is presented as a specific case of a more general modeling framework we put forth. The underlying question asks how to partition a given resource into a fixed number of parts such that the elements of the resulting partition can be scheduled among a set of user requests to minimize the worst case difference between the schedule and the requests. This particular allocation problem has not been studied before. The general problem is difficult in part because the evaluation of the objective problem is a difficult task by itself. We present a novel algorithm for its exact solution in a constrained setting and discussion of the unconstrained setting in, followed by a number of practical applications of these solutions. The solution to the constrained optimization problem is shown to provide sizable benefits in allocation efficiency in a number of contexts at a minimal implementation cost. The specific contexts we look at include communication over a shared channel, allocation of many small channels to a few users and package delivery from a central office to a number of satellite offices.

We also present a set of new fairness results for auction-based allocation mechanisms and show how these mechanisms also fall within our modeling framework. Specifically, we look at using auctions as mechanisms to allocate an indivisible shared resource fairly among a number of users. We establish that a straightforward approach as has been tried in the literature does not guarantee an fair allocation over a long time scale and provide a modified approach that does guarantee a fair allocation. We also show that by allowing users to strategize when bidding on the resource we can avoid the problem of unfairness, for some simple cases. This analysis has not been seen in existing literature.

Finally, an analysis of the deterministic and stochastic stability of our class of models is presented that applies to a large subset of the models within our framework. The deterministic stability results presented establish the ultimate boundedness of the lag of deterministically stabilizable models in our framework under a wide variety of quantizer-based scheduling rules. This variety of available rules can be used to further control the behavior of the lag of a stable mechanism.

We also discuss the application of existing stochastic stability theory to a large subset of the stochastic models in our framework. This is a straightforward usage of existing stability results based on verifying the satisfaction of a stochastic drift condition.

Thesis Supervisor: George Verghese
Title: Professor of Electrical Engineering

Acknowledgments

I would like to start by thanking the people and organizations most directly responsible for creating a work environment which made this work possible. Financial and administrative support for myself and my research was made available through United States Army AASERT grant number DAAH04-95-1-0473 and U.S. Army Research Laboratory Cooperative Agreement DAAL01-96-2-0001, and the Digital Signal Processing research group and teaching appointments within the Department of Electrical Engineering at MIT. In particular, I would like to thank Alan Oppenheim for his willingness to provide me a place in his research group for the last six years.

My thesis committee members, John Buck and John Tsitsiklis, provided me with valuable feedback on the structure and content of this thesis. They were instrumental in helping with the process of finalizing and presenting this research.

There are also many other people who have helped me in different ways. Anant Sahai came with me to MIT as a great friend, and we leave together the same. I would like to thank him for his advice, conversation and humor through these past years and hopefully many more in the future. In the DSP group, I have known many great people; all have impacted my work and life in some way or another. Of these, I would like to especially thank Giovanni Aliberti, Richard Barron, Stark Draper, Yonina Eldar, Chris Hadjicostis, Nick Laneman, Andrew Russell, Maya Said, Wade Torres, Kathleen Wage, Dianne Wheeler, and Greg Wornell. Vivian Mizuno has also been a fun conversation partner while waiting to meet with my advisor.

I owe a great deal to my advisor, George Verghese, who has been an unfailingly patient sounding board and teacher. Through many starts and stops, twists and turns, George has always tried to guide me towards a better understanding of the problems we have investigated. He is not only a great teacher and advisor, but a wonderful person that I feel privileged to know.

Finally, I must recognize the contributions of my family. My parents and brothers have given me a strong foundation for life upon which I have been able to grow with curiosity and confidence and for which I will always be grateful. In recent years, my wife Darla has provided the inspiration and motivation needed to finish this thesis. I appreciate the sacrifices she has made, and look forward to many years of making it up to her.

For

Ryden

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we develop stability properties and applications of a simply-defined class of dynamic resource allocation mechanisms. The mechanisms are meant to solve the problem of repeatedly allocating a shared resource among a number of competing users according to some set of requested or assigned resource fractions. The resource may be a group of computing processors, a network of queues, a shared communication link or even a limited number of bits in a quantizer. This allocation problem arises in a number of contexts, such as input-queued switching [51], packet radio networks [66], reentrant queueing networks [13], and vector Σ-Δ encoding [7]. In all of these contexts one of the goals is to allocate the resource so that each user gets the fraction of the resource that it requests or requires, or that the allocation be *fair*.

The aspects of this class of mechanisms we investigate fall into two major categories: *stability* and *performance*. We will call a mechanism stable when the internal state of the mechanism remains bounded and long-term fairness guarantees are met. In the first part of this thesis, we present a number of conditions that guarantee stability or instability in various subclasses of mechanisms. A subset of the general model we analyze automatically guarantees long-term fairness when the state is bounded (in a sense we define later); at the end of this thesis we present an alternative model based on auction mechanisms that guarantees boundedness but exhibits bias in the fairness guarantees.

The performance of a mechanism relates to its short time-scale properties. The performance of a stable mechanism concerns *how well* the mechanism achieves an equitable allocation. This is described when we discuss stability; a stable mechanism whose state is bounded yet varies widely may be termed less fair than other mechanisms. We will see that there are many rules we can use to schedule a resource among a set of users that all guarantee stability; when we choose among these rules we can control (to some degree) the performance of the system as well. We investigate performance more closely in the second part of the thesis, which looks in detail at the design of a very specific allocation mechanism and a number of practical applications of the mechanism.

We isolate a class of simple allocation mechanisms that can be used to model the contexts mentioned above and investigate whether or not the mechanisms ensure an equitable, or *fair* allocation of the resource. For our purposes, a perfectly fair allocation is one that splits the resource exactly according to the requests from the users. In practical situations we can not guarantee a perfectly fair allocation at every decision point so we discuss fair scheduling mechanisms whose allocations approach the perfectly fair allocation over longer periods of time.

The main focus of this thesis is to investigate a simple resource allocation model that shares significantly with a number of existing different dynamic resource scheduling models. Our model can be understood very simply, and allows us to build intuition about other models that share the same basic structure, while also allowing us to solve some specific allocation problems easily.

1.1 Resource Allocation

The basic allocation problem we consider can be summarized as follows. A number of customers (users, sessions) all generate work over time that requires the use of a specific resource. Each customer has an assigned nominal fraction of the resource that he is allowed to use. However, the resource has a limited number of possible configurations between which it is switched in successive intervals of time, according to a scheduling rule. Each configuration can not necessarily accommodate all of the customers simultaneously, and is only able to process a prescribed amount of work based on the customers that are serviced.

As a concrete example, we begin with a simple scheduling problem adapted from [61]. A traffic intersection is controlled by two pairs of traffic lights: one pair for each of the two intersecting roads, and one light for each direction on each road. The goal is to choose a pair of lights to turn (or keep) green (and turn or keep the other pair red) at fixed intervals so that only one road is allowed to use the intersection during each interval while not allowing the queues of waiting cars to become too long. The choices are made based on the number of cars waiting in the four directions at the beginning of the interval (we assume the lights change instantaneously, and all cars that do not make it into the intersection before the light turns red are stopped). The incoming traffic in the four directions is modeled as a random process, each direction having a well-defined average arrival rate. The cars may travel at a fixed rate through the intersection when they have a green light.

This simple problem has all of the elements that we will be discussing in this thesis. First, we have a core static resource allocation problem — allocate the intersection to one of two pairs of traffic streams. The particular allocation problem here is fairly simple, and in fact is a simple case of the input-queued switching problem described later. Second, we have a dynamic scheduler that repeatedly makes an allocation decision according to constantly changing inputs (numbers of waiting cars) that depend on the previous scheduling decisions and external inputs (new arrivals).

The first question that we are interested in is modeling, designing and understanding the core resource allocation problem from a geometric point of view. The next question is how to model the dynamic scheduler. The choices the scheduler is allowed by make may be very simple, as in this example, or possibly much more complicated where the resource may represent a large network of communication links with a large number of users competing for bandwidth. Finally, once we have the dynamics and the core allocation problem, we would like to understand the effects of various reasonable scheduling strategies. By scheduling strategy we mean a rule that makes an allocation decision based on some set of inputs (or requests). The main property of a scheduling strategy we look at is whether or not it stabilizes the system (keeps the inputs bounded). This will be taken up in much more detail later on.

We investigate all of these issues in the following chapters; for now we would like to discuss them with relation to our traffic example. We have already seen that the central allocation problem for our traffic example is very simple: we have two possibilities. To model this situation geometrically, we say that we have two possible configurations, or schedules, that comprise a *schedule constellation.* A schedule constellation is a bounded finite collection of points that lies in the scheduling state space. The state space is the space in which the inputs (requests) lie. For this example, the inputs are the numbers of waiting cars, so the state space would be the set of nonnegative integer vectors of length 4. If the vector of inputs is labelled in order: *North*, *East*, *West*, *South*, then the two configurations correspond with the points $(C, 0, 0, C)$ and $(0, C, C, 0)$.[1] We can think of a configuration as describing how many cars are allowed to go through in the four directions during a single interval. In this case, we allow C cars to move through the intersection from each of two opposing directions during each interval. Given the current state, the state at the beginning of the next interval is found by adding the new arrivals and subtracting off the configuration.

By doing this, we have defined the scheduling problem as a dynamic feedback system whose behavior can be understood by considering the state trajectories. Our goal then is to determine a scheduling strategy that keeps the number of waiting cars bounded. A simple rule might find the direction that has the largest number of waiting cars and turn the corresponding lights green. A different rule might find the pair — either (North, South) or (East, West) — that has the largest total number of waiting cars and turn the corresponding lights green. Which of these achieves the goal of keeping the number of waiting cars from growing unbounded? This is an example of the type of question we take up in more detail in the chapters to come.

We describe a generalization of this problem which includes a number of situations in the current literature, including input-queued packet switching [51], constrained queueing (packet radio networks) [66], fair queueing [57] and other communication and computational contexts. We also look at some of the scheduling rules used in these situations and show how they all fit within our common framework.

One interesting aspect of the model we present here is its ability to capture dependencies between users and within a distributed resource, such as a queueing network, in the same geometric framework as simpler models. The model does not take the place of domain-specific models such as queueing networks; restricting our view to a specific domain often allows us to gain even more insight into a particular problem. However, our framework allows us to think about and visualize a number of very different dynamic resource allocation problems within a common context. For example, our framework allows the state vector (number of waiting cars above) to vary over a continuous space, incorporating other possible dynamic feedback systems including a vector Σ-Δ encoder, which we discuss in more detail later. We might also model systems in which materials are arriving at a processor according to some stochastic characterization, and we must choose the configuration of the processor during a sequence of rounds to make use of all of the materials.

A subset of the models we are considering here can be captured with the following description:

> Given a constellation (finite set) of points $\mathcal{S}$ in $\mathbb{R}^N$ whose convex hull contains a specified vector $\bar{\mathbf{r}}$ and given a distance measure $d(\boldsymbol{\ell}, \mathbf{s})$, does $\boldsymbol{\ell}[n]$ stay bounded as $n \to \infty$ in the following dynamic equation:
>
> $$\boldsymbol{\ell}[n+1] = \boldsymbol{\ell}[n] + \bar{\mathbf{r}} - \mathbf{s}[n] \ ,$$
>
> where $\mathbf{s}[n]$ is the point in the constellation closest to $\boldsymbol{\ell}[n]$ under the metric $d(\boldsymbol{\ell}, \mathbf{s})$? If $\boldsymbol{\ell}[n]$ stays bounded, how large is the region within which $\boldsymbol{\ell}[n]$ stays?

The *lag* variable $\boldsymbol{\ell}[n]$ plays an important role in our discussions. We think of $\boldsymbol{\ell}[n]$ as the state of the system; it is the accumulated mismatch between the actual allocations and the desired allocations. By keeping $\boldsymbol{\ell}[n]$ small, we guarantee that this mismatch does not grow unbounded. Under many choices of constellation and metric as we will show, $\boldsymbol{\ell}[n]$ remains bounded. If the constellation has a high density of points, $\boldsymbol{\ell}[n]$ travels in a trajectory that remains close to $\bar{\mathbf{r}}$. We

[1]We are assuming here that there are always at least C cars waiting in the directions we choose; in a more practical setting we would augment this collection of configurations to include all configurations that schedule at most C cars in two opposing directions.

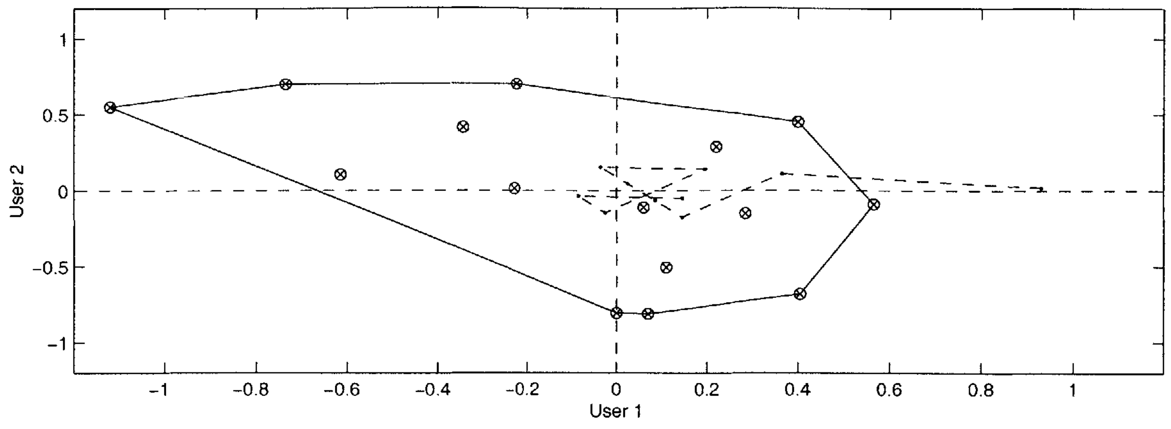


Figure 1-1: State evolution example with an arbitrary constellation. The trajectory is indicated with a dashed line, the convex hull of the constellation by a solid line, and the constellation points by $\otimes$.

can think of this as a simple control problem, where the control $\mathbf{s}[n]$ is chosen as a function of $\ell[n]$ in order to keep $\| \ell[n+1] - \bar{\mathbf{r}} \|$ small. A sample trajectory of $\ell[n]$ is shown in Figure 1-1 for a particular constellation using the minimum squared distance as the distance metric, and with $\bar{\mathbf{r}} = 0$. This figure gives us a good view of the general dynamics of these systems when they are stable. When the state variable $\ell[n]$ is outside the confines of the constellation it will converge towards the constellation. This is not always the case, and depends crucially on the constellation and metric. The behavior of this large-scale convergence, when it happens, generally depends on the interplay between the structure of the constellation as visible from the outside and the properties of the distance metric. When $\ell[n]$ is inside the constellation, the behavior of the state depends much more on the internal structure of the constellation; the location and spacing of the points inside the constellation determine the actual trajectory. These characteristics govern steady state performance, ability to adapt to changing inputs, and scheduling accuracy. The distance metric also affects the small-scale behavior of the lag. By choosing an appropriate metric, we can enforce preferences we might have about the relative desirability of different lag values. We can not enforce arbitrary preferences, but in Section 4.2 we give a few illustrative examples of how we might approximately match a specific set of preference.

We look at these two aspects — large-scale stability and small-scale performance — by both developing global stability results for different classes of metrics and system definitions, and investigating how we might design the structure of a constellation to optimize the small-scale behavior. We discuss the connection between the constellation design problem and vector quantization in Chapter 5.

Our problem was very simply stated, yet can be easily extended to encompass a number of interesting scheduling problems, as we will show later. If we modify the dynamics in the canonical scheduling problem to allow for time-varying external input signals $\mathbf{r}[n]$:

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - \mathbf{s}[n]$$

then we have something that is much closer to the model we develop later on. This can be simply captured with the block diagram in Figure 1-2. In the figure, the Q block represents a minimum-distance quantizer whose output is the point $\mathbf{s}[n] \in \mathcal{S}$ that minimizes the distance $d(\ell[n], \mathbf{s}[n])$.

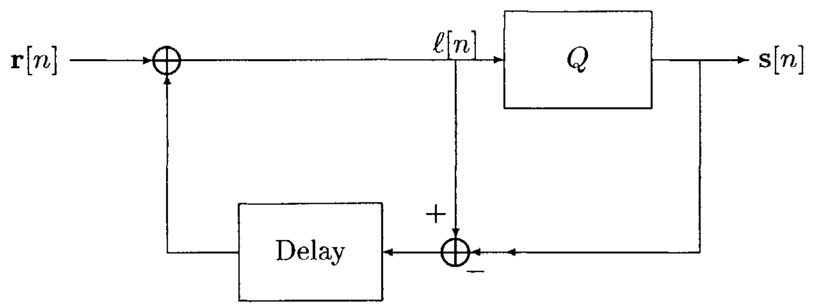


Figure 1-2: Block diagram of the canonical dynamic equation.

1.2 Structure

This thesis is structured as follows. We begin in Chapter 2 with a survey of related literature, and give an overview of the most closely related work. In Chapter 3 we continue with a definition of the dynamic allocation mechanism model, consisting of a core static allocation model and a dynamic feedback mechanism. In terms of the canonical scheduling problem from the previous section, the core static allocation problem is simply a model for the scheduling decision that must be made at every round. The same rule is used every round; only the input (the state) changes value. In addition to defining the problem, we give a number of examples that show how it can model a number of real applications. This static allocation problem is interesting by itself, as it is related closely to quantization constellation design. Once we have the core static problem established, we move to defining the more general dynamic allocation problem model that provides the basis for the last half of the thesis. We define the allocation problem and the class of *quantizing* scheduling policies that we will investigate. We also look at a number of queueing network models that can be modeled in our framework. We show that some of the scheduling policies used for these models can be categorized as quantizing scheduling policies. This is followed in Chapter 4 by a development of stability and stabilizability results for both deterministic and stochastic dynamic models when using quantizing scheduling policies.

Chapter 5 is an investigation of a constellation design problem that arises in a specific resource allocation context. We present a general discussion of the problem as well as a solution for a constrained version of the problem. The next chapter, Chapter 6, applies the constellation design solution from Chapter 5 to a dynamic context. This results in a number of practical applications of the design solution that increases the efficiency of some communication and transportation resource allocation mechanisms.

This is followed in Chapter 7 by a discussion of another version of our dynamic model which allows the core quantizer to be replaced with an auction. Auctions behave like quantizers but with a few important differences. A naïve auction-based allocation mechanism (published in the literature) is shown to not guarantee fair allocation of the resource to the competing users. We then develop an alternate method that does guarantee a fair allocation on average. Finally, we extend the model to allow the users to strategize, or choose their own bids as inputs to the auction mechanism. We show that the ability to strategize results in approximately fair allocations for finite numbers of repeated allocations, and asymptotically fair allocations for indefinitely repeated allocations.

We finish with our conclusions and descriptions of a number of interesting extensions and other ideas. Finally, the Appendix contains all long proofs, in order to make the text more readable.

1.3 Contributions

The major contribution of this thesis is the investigation of a specific resource allocation optimization problem whose solution has both practical application as well as theoretical interest (Chapters 5 and 6). It is presented as a specific case of a more general modeling framework put forth in Chapter 3. The underlying question asks how to partition a given resource into a fixed number of parts such that the elements of the resulting partition can be scheduled among a set of user requests to minimize the worst case difference between the schedule and the requests. This particular allocation problem has not been studied before. The general problem is difficult in part because the evaluation of the objective problem is a difficult task by itself. We present a novel algorithm for its exact solution in a constrained setting and discussion of the unconstrained setting in Chapter 5, followed by a number of practical applications of these solutions in Chapter 6. The solution to the constrained optimization problem is shown to provide sizable benefits in allocation efficiency in a number of contexts at a minimal implementation cost. The specific contexts we look at include communication over a shared channel, allocation of many small channels to a few users and package delivery from a central office to a number of satellite offices.

We also present a set of new fairness results for auction-based allocation mechanisms in Chapter 7 and show how these mechanisms also fall within our modeling framework. Specifically, we look at using auctions as mechanisms to allocate an indivisible shared resource fairly among a number of users. We establish that a straightforward approach as has been tried in the literature does not guarantee an fair allocation over a long time scale and provide a modified approach that does guarantee a fair allocation. We also show that by allowing users to strategize when bidding on the resource we can avoid the problem of unfairness, for some simple cases. This analysis has not been seen in existing literature.

Finally, an analysis of the deterministic and stochastic stability of our class of models is presented in Chapter 4 that applies to a large subset of the models within our framework (as described in Chapter 3). The deterministic stability results presented in Chapter 4 establish the ultimate boundedness of the lag of deterministically stabilizable models in our framework under a wide variety of quantizer-based scheduling rules. This variety of available rules can be used to further control the behavior of the lag of a stable mechanism as illustrated in Section 4.2.

We also discuss the application of existing stochastic stability theory to a large subset of the stochastic models in our framework. This is a straightforward usage of existing stability results based on verifying the satisfaction of a stochastic drift condition.

Chapter 2

Background

The work in this thesis relates to a number of existing areas of study. In this chapter, we discuss how our work here grows out of and differentiates itself from such existing research.

The basic dynamic model we use, a simple dynamic feedback system with a quantizer in the loop, is similar to a number of other models that have been used in a number of other contexts, including Σ-Δ encoders [7, 35, 43], input-queued switches [51, 30, 15, 62], reentrant multi-class networks [42, 38], feedback control [52], constrained queueing networks [66, 65], fair allocation strategies [3, 57]. Although most of these models are more complex in some way or another than our model, they all share a similar basic structure.

Fairness

The problem of defining a fair allocation of a resource has been considered in a number of different ways. At its most basic, fairness is simply the notion that every competing user gets a turn with the resource within a finite amount of time after requesting the resource; this sense is used in distributed algorithm analysis [45]. However, this limited notion of fairness can not guarantee any particular fraction of the resource to a user. In many data communication and multi-processing contexts, assigning a fraction of a resource to a user is a very common method of sharing a resource. Each user is nominally given entitlement to a fraction ρ_i of a shared resource, and then the allocation of the resource to the users over time is chosen in such a way as to match these fractions as closely as possible. A method for choosing these allocations in a simple discrete-time setting is given in [3], and is called p-fair scheduling. A fixed number of identical processors are allocated among a number of competing users during a number of identical rounds. The (integer) number of times that a user has won until a given round N is within 1 of the desired number $\rho_i N$ under p-fair scheduling.

We may also use "fairness" coefficients to control other aspects of the system, as in the generalized processor sharing (GPS) algorithm used to allocate a shared communication link [57], where the coefficients that are assigned do not decide the fraction of the bandwidth allocated, but govern the delay properties of packets as they pass through the link. In these applications ordinary fairness is guaranteed if the input queues do not overflow. However, by choosing coefficients as in the GPS algorithm, we can affect how close to fair a mechanism is for each user. The coefficients make it possible to control how much delay each user sees relative to the other users. We discuss a few examples of this kind of control in Section 4.2.

Fairness has also been extended to apply to situations in which the total amount of resource available determines the fractions that should be given to each user. Approaches in this category include min-max fairness, proportional fairness, and others [49]. We assume here that the fractions are already chosen and that the only question left is how to allocate the resource in order to meet the desired fractions.

However, our model also allows the desired fractions to change from round to round. This happens, for example, in queueing systems where the desired amount of service is equal to the number of waiting packets, which changes over time. This can also be used as a model for a Σ-Δ encoder, where the shared resource is a small number of bits to be allocated at each round among a number of signals. The signals may take values over a wide range. The ability to vary the fractions may also be useful when the priorities we assign the users change over time.

Stability

Our proof of deterministic stability under all smooth norms in Chapter 4 relies on a number of properties of convex sets and polytopes. The needed properties are detailed in Appendix A and more extensive material can be found in [58, 33, 5] and other convex analysis and linear programming texts.

Two existing approaches to showing stability for stochastic systems under specific scheduling policies are relevant to the systems we study here. Lyapunov techniques are often used to establish stability (i.e. boundedness of queue lengths) of queueing systems [51, 38]. In these cases, establishing stability involves showing that a specific drift condition holds. The drift condition requires that a nonnegative Lyapunov function of the state always be decreasing outside of a compact region of the state space. This approach has been applied to queueing networks [38] and has been generalized to show different forms of stochastic stability for a wide variety of Markov chains over general state spaces (which include many discrete-time queueing models and dynamic systems) in [53]. Closer to our model, which operates in Euclidean space, are the results in [40] establishing basic properties of Markov chains (in Euclidean space) that satisfy drift conditions, and results in [46] and [25] (which extends the result in [46] to hold for more general increment distributions). The approaches used in these works relies on modeling the values of the Lyapunov function as a supermartingale when the drift condition is satisfied. An accessible introduction to martingales can be found in [24]. Using a martingale approach can be more general because it does not assume that the process of interest is Markovian as in [53]. In our model, the structural dynamics are Markovian, but the introduction of stochastic models whose parameters are correlated over time can invalidate this property. Our results rely heavily on the presentation in [25], which develops properties of real-valued stochastic processes that satisfy the same strict stochastic drift condition we use in this thesis.

Another method involves using fluid models of general stochastic networks [52, 14, 13, 15]. We do not use this method, but it is important to illustrate how this work differs from what we do here. The work on fluid models has shown that the stability of a member of a large class of stochastic networks is equivalent to the stability of a related fluid model, obtained through a limiting process. In [52] especially, a stochastic model is presented that is very similar to the dynamic model presented in Chapter 3. The model consists of a network of queues that route jobs probabilistically either to each other or out of the network. The main thrust of the work is to establish a relationship between stability of the original model and stability of the fluid approximation for these kind of networks under stationary policies, and also to determine cost-optimal policies. The approach uses fluid models obtained through methods similar to those in [13]. The final results establish a class of policies called feedback regulation policies that are simpler to implement than average-cost optimal policies, but are stabilizing and track the performance well.

Although our presentation as well as our model are similar to those in [52], our focus is different.

We concentrate on the geometric nature of the allocation problems; our model allows a greater variety of problems with control constraints to be incorporated, including constrained queueing problems like those in [66, 65]. The work in [52] also relies on the fact that the fluid model, including the scheduling policy, must somehow be obtainable from the stochastic model, which is not always straightforward for arbitrary scheduling policies. The results in [52] give us a method of converting from fluid policies to discrete policies, but do not investigate directly the analysis of discrete policies.

The model we discuss here also differs from stochastic network models such as the one in [52] and those discussed in [53] by the fact that the state evolves in a continuous space while the controls are restricted to comprise a discrete finite set. Models such as reentrant queueing networks have a finite set of control possibilities, but evolve on a countable space as do most queueing models. The fluid models in [52] use continuous controls as limits of discrete controls. Another similar model is a storage model [53] which has very similar dynamics to the problem we consider here but assumes inputs that are chosen from a continuous domain rather than the discrete constellation we use here. The aspect of our model that differentiates it from these other models is the finite and discrete nature of the values that the controls can take while still allowing the state to vary continuously.

Another paper [34] establishes the stochastic stability of a very general class of feedback quantization mechanisms, including delta modulation, adaptive quantization, differential pulse code modulation and others. It is not immediately clear that the published results can be applied directly to the feedback structure we use here, but it is likely that the results could be extended to include this case. Stability is shown as the almost sure convergence of averages of arbitrary bounded measurable functions of the processes that define the model. The model in [34] allows state-dependent quantization and complex state evolution dynamics. The result is very general and depends on technical assumptions that are not always easy to translate into specific applications.

Queueing Theory

Our basic approach is similar to the sequencing and routing work with open multi-class queueing networks [39, 13, 52] and constrained queueing networks [65]. Our work differs in that it uses an underlying structure which allows an arbitrary geometric formulation of the scheduling possibilities and a continuous state space on which the model state evolves. The queueing network model in [13, 52] allows for integer numbers of jobs to be scheduled at discrete times; the overall schedules are limited to be integer vectors. However, the queueing network model also incorporates variable processing times and arrival times. In our work, we use a discrete-time queueing model where scheduling decisions are made in rounds indexed by integer times. We discuss how we might extend our model to include variable processing times in the concluding chapter.

The differences between our model and traditional queueing network model (including reentrant multi-class networks) make it possible for a system to have no possible work-conserving (or non-idling) policies. This means that stability results for queueing networks such as [38, 39, 4] do not apply directly. As we discuss below, the constrained queueing model in [65] shares this property with our model, but does not allow for arbitrary scheduling constellations.

Reentrant Multi-class Networks

Much of the work related to reentrant multi-class networks [42, 38, 37, 13] deals with policies that look only at the whether or not the queues are empty, not the number of jobs in the queues. The scheduling policies that are interesting to them, such as first-come-first-served and static buffer priority policies, choose to service queues based only on whether or not the queues are empty, along with some other augmenting rule such as static priorities to choose between nonempty queues. All of these scheduling policies are consistent with the maximum size scheduler, or 1-norm based quantizing scheduler we discuss in Section 3.5. There are many interesting results dealing with stability and performance with this class of scheduling policies [42, 38, 4, 37, 13]. An example in [13] illustrates a virtual blocking problem in a reentrant system. Because of the structure, two queues that are not directly tied together block each other, and reduce the maximum throughput. However, this problem is only present because the scheduling policy just takes into account the emptiness of the queues, not the number of jobs in nonempty queues. The scheduler is a first-come-first-served (FCFS) static buffer priority (SBP) policy. Although it is non-idling, it does not take into account the queue lengths and as such is subject to problems similar to the maximum size matching algorithm in the input-queued switching literature.

Input-Queued Switching

The input-queued packet switching problem involves selecting the configuration of an $N \times N$ switching fabric in order to route packets from the inputs to the outputs and keep the input queues from growing unbounded [15, 51].

A number of existing scheduling techniques used for the input-queued switching problem can be modeled closely with our dynamic model [30, 51, 15], using an appropriate choice of distance metric in our quantizing scheduler of Chapter 3. Our model places some of the previous solutions within a geometric context in which to think about the problem. Our geometric approach indicates why certain approaches might or might not work better than others, or might not work at all. It confirms the basic results and difficulties that have been identified already.

There are two main approaches to the input-queued switching problem. The first uses a single first-in-first-out (FIFO) queue for each input; this has been shown to limit throughput [32] to $(2 - \sqrt(2)) \approx 58.6\%$ of the maximum possible because of blocking problems at the heads of the queues. The second approach uses a separate queue at each input for each possible output; this approach is called virtual output queueing (VOQ) [51]. Within the VOQ approach, there have been a number of different techniques proposed for achieving the maximum throughput (see [51] for references). A number of these techniques can be described as maximal matching algorithms because they all make scheduling decisions at each round by computing a maximal matching in a graph that describes the switch and queue status. A general result that applies to all maximal matching algorithms can be found in [15]. It guarantees that any maximal matching scheduling algorithm achieves at least 50% throughput. The earliest scheduling approaches used a maximum size matching, which tries to schedule as many queues as possible within the constraints of the switch; this has been shown to achieve full throughput in simulation under reasonable assumptions, and is also known to be unstable using more general assumptions [51]. The next approaches [51, 15] used a maximum weighted matching, where the schedule is chosen by finding a maximal matching that maximizes the sum of the queue lengths corresponding to the scheduled packets. This matching is shown to achieve full throughput; however, the computation of the maximum weighted matching is more complex than other methods, and does not lend itself to easy implementation in hardware. Another matching that has been introduced in order to simplify the scheduling is the greedy weighted matching [30]. This method seems to guarantee full throughput in simulation, but the theoretical lower bound on the maximum throughput is still only 50%.

All of these scheduling techniques are consistent with using a static minimum-distance quantizer to make the scheduling decisions. This is the view we take of the input-queued switching problem.

In Section 3.5.1 we discuss this in more detail. Our framework places all of the approaches in a common setting in which we can analyze and compare them in a different manner than is done now.

Much like the fair queueing model that we talk about later, there are also algorithms [30] based on accumulating credits over time as a way of determining the relative priorities of the input queues. These credit algorithms provide the ability to control the system in two additional ways. First, by choosing the rates at which credit is accumulated for the different queues, the queues are given relative priorities in order to modify the delay properties of the system. Second, the credit accumulation serves as a traffic smoothing device that reduces the effects of bursty traffic. We discuss how this credit mechanism can be related to our model in Section 3.5.4.

Constrained Queueing / Packet Radio Networks

Our model is also closely related to the constrained queueing model presented in [65]. The input-queued switch described above is in fact a special case of a constrained queueing network. The model incorporates constraints between servers in a queueing network that can be used to capture features of parallel processing problems, or of a broadcast packet radio network [66]. The constrained queueing network itself is a general model whose basic structure can also be captured within our geometric model. By viewing the constraints as restricting the possible schedules that can be chosen at the scheduling instants, we can construct a schedule constellation containing all of the possible schedules. Stabilizability results for these constrained queueing networks are developed in [65] as a generalization of results in [66]. The networks are not constrained to operate in discrete-time and jobs are allowed to have variable processing times, although the scheduling algorithms that are developed make their decisions at well-defined instants in time. This is required because the constraints may be global, and require the synchronization of scheduling control among all of the servers in the network.

The development of the model and stability results in this thesis in Chapters 3 and 4 parallel to some degree the work on the constrained queueing model. Our focus in Chapter 4 is on determining the stability region of our model as is done in [66, 65], but also on finding a larger class of scheduling rules that guarantee stability. The major difference between the two is that our systems allow the inputs and state of the model to lie in a continuous space, rather than the countable state space used in the constrained queueing model. The other important difference is that our model operates in discrete time much like the model in [66].

The constrained queueing model adds an interesting level of complexity to the problem of queueing network scheduling. Although it makes the problem of scheduling more difficult, it also allows the model to capture a wider variety of contexts such as packet radio networks, input-queued switches and others. We show that a conceptually simple scheduling policy based on minimizing the distance between the state and schedule maintains stability for all feasible input rates as in [65], while the problem of reducing the complexity of the scheduling policy is also important and not investigated in depth here (see e.g. [30]). A common restriction used to make stability guarantees is to assume the policies are non-idling, or are work-conserving. For a general constrained queueing model the constraints between servers make it sometimes impossible to use non-idling policies, as we can see with the input-queued switching problem. The class of reentrant multi-class queueing networks discussed above can be seen as a special case of this model, where each station can serve only one of a number of buffers during each round. The Cartesian product of these sets of schedules for each station will give us a constellation of possible overall schedules. However, non-idling policies are always feasible for reentrant networks.

We show in Section 3.5.3 that if we only consider discrete-time scheduling problems, the constrained queueing model and algorithms that stabilize it are very closely related to the model and schedule rules we present in this thesis.

Fair Queueing

Basic fair queueing algorithms schedule a shared resource among a number of competing users, each of whom has been assigned a fairness coefficient. These fairness coefficients are meant as a way of specifying what fraction of the resource a user should nominally receive relative to other active users.

We can view these algorithms as specific examples of quantization-based scheduling with more complexity. The generalized processor sharing (GPS) model in [57] and the more general rate-proportional server (RPS) model in [63] fall into this category. The GPS model uses a measure of accumulated fairness credit as a state variable that is input to a basic quantizer. The accumulation of credit depends on the state of the queued work; as long as a queue is full, the credit accumulates at a constant rate. The differences between algorithms, as captured in the RPS model, is how they reinitialize the state when a queue becomes full again. This credit model is also used in [30] in the input-queued switching context.

The p-fair scheduler is another lag-based algorithm that uses a more complicated scheduling rule. The model in [3] is essentially the same as the backlogged fair queueing model. Each of a number of users has a reservation which is some fraction of the total resource available. The number of independent resources that can be allocated to the users is larger than 1 in general, but there are no constraints on how the resources can be allocated. At each round, a complex rule is used to decide which users are allocated which resources. This rule, being a function of the current lag and the reservations, can also be seen as a mapping that divides the lag state space into allocation regions. However, the function depends on the actual reservations, and so is not very flexible with respect to changes in reservations. All of the norm-based scheduling mechanisms do not depend on the reservation, simply the current lag and the current scheduling constellation.

Fair queueing provides an interesting motivation for the study of larger classes of stabilizing scheduling rules because a wider variety of rules allows more flexibility in the specification of the closed-loop behavior of the mechanism. We discuss in Section 4.2 a possible way we can extend the weighting coefficient-based fairness specification to a more general way of specifying fairness relationships between users.

Vector Quantization

A recent comprehensive survey of quantization can be found in [23]. The areas of quantization that apply most directly to the work presented in this thesis are vector quantization (VQ) [21] and Σ-Δ quantization [26, 27].

VQ is a much-studied area of quantization theory. Although the problem statement we use in Chapter 3 has significant differences from the VQ problem, it has a similar spirit. The literature on VQ is vast, and a good starting point would be [23]. We are interested mostly in two parts of the basic VQ problem, which are covered well in [21, 60]: the basic model formulation and the actual quantization procedure. The basic structural model consists of a collection of quantization points and a distortion metric. Input vectors are quantized by choosing the closest quantization point according to the distortion metric. We are not concerned with codebooks because the quantization points are not meant to be coded to a standard bit representation, but instead are directly represented by the configuration of a resource; we discuss this difference in Section 5.6. Our basic static model in Chapter 3 is similar to the structural quantization model, and the scheduling mechanism we use is equivalent to a minimum-distance vector quantizer. We are not investigating a general constellation design problem; instead, Chapter 5 investigates a particular scenario in which we solve a specific constellation design problem that is different in structure from VQ design problems. The second part of VQ that is similar to part of Chapter 5 is the problem of finding the closest quantization point given an input vector. This is a difficult problem in general [23], as the obvious approach in the absence of specific structure in the constellation is a brute force search through the possibilities. In higher dimensions, the computation involved becomes infeasible. A class of constellations whose geometric structure is based on lattices simplifies this computation [11, 23, 17]. This division between lattices and general constellations shows up as well in Chapter 5, where our scheduling (quantization) algorithm becomes simple when the constellation is equivalent to a particular lattice. Otherwise, the the scheduling problem is NP-hard [23] in the worst case.

Another area of interest in quantization theory is the study of Σ-Δ encoders; a comprehensive overview of Σ-Δ modulation can be found in [7]. Scalar Σ-Δ encoders have received much attention, and are used in many practical systems. The noise properties are studied in [22], and decoding algorithms and stability analysis of Σ-Δ encoder structures is covered in [26]. The basic single-loop Σ-Δ encoder is equivalent in overall structure to our feedback dynamic scheduler in Chapter 3. However, our scheduler is really equivalent to a Σ-Δ encoder with a vector input, which has received little attention in the literature. The only work specifically related to vector Σ-Δ quantizers relates to the control of three-phase switching power converters [43, 55]. These are essentially two-dimensional Σ-Δ encoders with a simple hexagonal quantization constellation. The vector Σ-Δ structure is compared in [55] with existing methods of controlling the converters based on pulse-width modulation (PWM) techniques [28]. In [43], the power spectrum of the error signal for the vector Σ-Δ encoder is derived. There has been a wide variety of approaches to the PWM problem, using a large number of control structures. Essentially, they attempt to approximate a desired reference signal by switching three terminal voltages on and off so that an averaged version of the switching signal (the load voltage) approximates the reference signal. The core control dynamics take the reference signal as input and output a sequence of switching configurations that achieve this goal. The configuration constellation has a fixed hexagonal structure, and there is no discussion of the distortion metric used. These results are not easily generalizable to higher dimensions and, as we discuss later, the boundedness of the internal state is not guaranteed for arbitrary constellations. The question of boundedness is not addressed directly in this work. The model we present in this thesis can be seen as a generalization of the vector Σ-Δ structure in [43]. We also discuss a generalization in the concluding chapter when we allow the scheduling decisions to be made at arbitrary discrete times; this extended model is (in 1 or 2 dimensions) equivalent to the interpolative Σ-Δ modulator discussed in [44] if the sampling times are allowed to vary.

Static Resource Allocation

The allocation problem that we present in Chapter 5 fits within the general model put forth in [29]. In [29], the basic framework is that of choosing an allocation (or partition) of a resource to a number of competing goals such that a given objective function is maximized. The problem presented in Chapter 5 fits within this framework, if we consider the objective function to be minimizing the worst-case error over all possible input vectors. However, this objective function does not fit within any of the problem classes in [29] that lend themselves to easy solution. Our allocation problem shares similar goals with a few other problems in the literature, including the change-making problem [47], the structured partitioning and vehicle routing problems [1], the vector quantization problem [23] and other bin packing and assortment problems. We show later in Chapter 6 that the problem we pose can be used to model practical applications; we discuss a few applications including reducing the latency of data sent over a shared link by a number of competing users, reducing the scheduling complexity of a particular communication link scenario, and minimizing the overhead cost associated with shipping packages or processing packets in systems with fixed costs for busy resources.

The *change-making problem* has been studied in [47, 69, 8], which give optimal and heuristic algorithms for its solution. This is an optimization problem that involves minimizing the number of coins with fixed denominations used to make a certain amount of change. It differs from our problem in a few ways. First, the number of coins of each denomination is unbounded; we consider a bounded number of resource pieces in Chapter 5. Second, even if we were to bound the number of coins of each denomination, the optimization goal is different; we try to choose denominations that minimize the difference between the change made and an unknown future requested amount, while the change-making problem seeks to minimize the total number of coins used to make an exactly achievable amount of change, given a known set of denominations. Last, the change-making problem only involves making change for a single customer. With an unbounded number of coins available, it does not make sense to talk about multiple customers because we can make change for each independently. However, with a fixed number of coins as in our problem, the customers now compete for the resource, and the problem becomes more interesting.

The allocation problem of Chapter 5 also has a similar flavor to the one-dimensional *bin packing* [48], *cutting stock*, and *assortment* [20] problems, but has a slightly different goal and constraint set. Our objective is to fit a fixed set of objects into a set of containers of unknown sizes whose total capacity is known to be the same as the total size of the objects. Our optimization objective is to choose the sizes of the objects to minimize the maximum overfill or underfill of any of the containers over all possible container sizes. We also assume in our formulation that a collection of smaller resources of given total size is equivalent to a single larger resource of the same size. This assumption is valid in some communication or computational contexts, but not necessarily in environments such as paper mills or lumber yards, where the cutting stock problem applies most directly.

Part of this problem also fits in the more general problem of partitioning. A general approach to partitioning sets of sized objects given a cost structure is described in [1], as it applies to bin packing, vehicle routing, queue scheduling and other applications. The focus in these cases is on determining an optimal allocation or partition. In Chapter 5 we look at the related, but different, problem of choosing the sizes of the objects to minimize the eventual cost when the objects are allocated.

There has also been a lot of research into vehicle routing and one-to-many delivery problems [2, 6]. In these problems, the emphasis is on choosing and routing vehicles through the destinations to minimize the cost of delivery. In the transportation application of Chapter 6 we assume here that all deliveries are done directly from source to destination and so the routing concerns are not important.

Auction and Game Theory

In recent years, economic resource allocation mechanisms — especially the auction — have gained attention in the computing resource allocation field [10, 67, 68]. Auctions are commonly used to price and allocate resources [50, 54]. They are attractive due to the simplicity of implementation, and their efficiency properties [16]. Recent work that studies the use of auctions to allocate processing resources in computing contexts includes a number of projects in [10], [31], [67], [68]. The major difference between the economic research and the computer-related research is that the computer-related work focuses more on using auctions as a tool, rather than studying them as independent systems.[1] The kinds of allocation guarantees we seek are not necessarily that the allocation be efficient, or maximize the profit of any of the participating agents. When using auctions as a tool for computing resource allocation, the goals are more focused on achieving performance guarantees, or meeting reservations; the economic basis of auctions is simply a means to some other end. Furthermore, the context in which they are used is much more controlled than the traditional context of allocating real world resources to human participants. This allows the properties of the overall system to be more closely modeled. For this reason, more recent research has focused on how to use auctions as mechanisms to achieve allocation goals that are much more difficult to attain using existing, more structured and centralized techniques.

Here we begin with a specific approach to using auctions for resource allocation in decentralized networks, detailed in [67] and called the Spawn system. The allocation mechanism used is simple and turns out to be analyzable for certain cases. The Spawn system uses a mechanism with dynamics very similar to those of our model. The results in [67] use empirical data to support the hypothesis that auctions will provide a fair allocation of a collection of shared computing resources among competing users. We present here some results that look at this of problem with a more analytical eye, and provide some initial results that help to understand the fairness of auction mechanisms when used in computing resource allocation problems.

[1]Auctions have always been used as tools for allocation in economic systems as well. However, the computer-related work neglects the other aspects of economic research into auctions.

Chapter 3

Dynamic Allocation Mechanism

At the core of this thesis is the problem of allocating limited resources among a number of competing users. Resource allocation is a long-studied problem, and arises in a large number of contexts. In different contexts, resources can represent any number of things: lumber, money, time, space, fuel. The users may also represent a wide variety of entities: corporations, employees, computer processors, integrated circuits, engines. In every context, we have a number of users that seek to make use of the same set of limited resources. Although the goals may also vary somewhat — minimize cost, maximize profit, equalize distribution — we consider a particular class of goals: trying to match a set of users' resource requests as closely as possible. The users' requests come in many forms; they may be bandwidth, queued jobs, processing time or a number of other possibilities.

These problems can be stated and analyzed using formal mathematical models. We will do this but we will rely heavily on geometric ideas and interpretations of the problems. Users' requests are seen as points in space, as are the possible resource schedules, or configurations of the resource. The problem of matching a set of requests with a configuration becomes an optimization problem: choose the configuration that is closest to the requests. We will also view dynamic scheduling problems later on as problems of controlling dynamic systems, and relate them to the geometric ideas we present here.

The basic dynamic allocation problem we will consider is how to choose configurations from round to round so that the fraction of the resource that a particular user receives over time is equal to (or close to) some desired amount. This amount may be a constant for each round, or may vary from round to round. A feedback scheduler is presented as a method of guaranteeing this, see Figure 3-1. There are three main parts to our scheduler. First is the description of the user resource requests $\mathbf{r}[n]$ at each round n. Users may request a constant amount each round, or the requests may vary according to some stochastic or deterministic rules. Second is the core static allocation problem which must be solved during each round to provide the actual schedule of resource allocations for the round. This consists of two pieces: the scheduling constellation S which specifies all possible schedules, and the scheduling rule $Q(\cdot)$ which specifies which schedule to use based on the current state of the system. Last is the actual feedback mechanism which updates the state based on the previous schedule. A combination of constellation and feedback rule is called a *model*, and the combination of a model with a scheduling rule is a *dynamic allocation mechanism*, or simply a *mechanism*. We also refer to a *static allocation mechanism* when we talk about a scheduling rule being used to determine a single schedule.

This chapter is structured as follows. In Section 3.1 we begin by describing the static allocation problem that serves as the core of our dynamic model. Then in Section 3.2 we describe our feedback scheduler model, and the scheduling rules that we consider here. Then, we refine this model to a *quantizing scheduler* in Section 3.3.

3.1 Static Problem

In order to formalize the structure of the model hinted at above, assume that we have N *users* of a single configurable *resource*. The resource has a number of possible ways of being scheduled for the users. We can think of these *schedules* as configurations of the resource. Each schedule is an N-dimensional vector that specifies in each component how much of the resource each user receives. It is not necessary in all cases that the sum of these components be the same for all configurations; this allows overhead costs and other constraints to be modeled by the set of schedules. The possible schedules depend on the application and the structure of the resource. For example, if there are 5 processors that make up the resource and can be allocated independently to the users, the possible schedules would consist of all nonnegative vectors that sum to 1 whose elements are multiples of 1/5. On the other hand, if there is a single processor that can handle more than one user at a time but with a performance penalty for multiple users, the schedules would be different.[1] We will use the notation $\mathbf{r}$ for the vector of users' requests (or reservations), and $\mathbf{s}$ for a schedule. The set of all possible schedules is denoted by $\mathcal{S}$.

Once we have the constellation set down, we need to determine some *scheduling rule*, or a method of determining which schedule is used when the users make their requests. For this thesis, we will concentrate on *static* scheduling rules $Q(\cdot)$, or ones that simply map a request $\mathbf{r}$ to a schedule $Q(\mathbf{r}) = \mathbf{s}$.

The scheduling rule that we look at here is a simple geometric rule. Given a finite set $\mathcal{S} \subset \mathbb{R}^N$ of points in space and a single point $\mathbf{r} \in \mathbb{R}^N$, find the point $\mathbf{s}^* \in \mathcal{S}$ such that some distance (or distortion) metric $d(\mathbf{r}, \mathbf{s}^*)$ is minimized. Or, in more formal terms,

$$\mathbf{s}^* = Q(\mathbf{r}) = \arg\min_{\mathbf{s} \in \mathcal{S}} d(\mathbf{r}, \mathbf{s}) \ .$$

As an example we would like to minimize the worst-case difference between the request and the actual amount of resource allocated to a user, we would choose $d(\mathbf{r}, \mathbf{s}) = \| \mathbf{r} - \mathbf{s} \|_\infty$. For specific applications, this or other measures of distance might make sense. We discuss the specific choice of $d(\mathbf{r}, \mathbf{s})$ below.

Another equivalent way to define our rule is to specify a region associated with each schedule such that if the vector $\mathbf{r}$ lies in the region, then the corresponding schedule is chosen. Define $\mathcal{R}_d(\mathbf{s})$ as the *allocation region* for a point $\mathbf{s} \in \mathcal{S}$. The allocation region for a point $\mathbf{s}$ is the set of points $\mathbf{r}$ such that the corresponding $\mathbf{s}^* = \mathbf{s}$, or

$$\mathcal{R}_d(\mathbf{s}) = \{\mathbf{r} \mid Q(\mathbf{r}) = \mathbf{s}\} \ .$$

We see that our choice of $d(\cdot)$ along with $\mathcal{S}$ determines these allocation regions. In turn, it is clear that the allocation regions completely determine the properties of the scheduling rule.

Scheduling rules for scheduling problems are not generally interpreted geometrically. Classical resource allocation problems deal with either a set of indivisible resources, or an arbitrarily divisible resource [29]. Queue scheduling problems are not seen geometrically because the decisions are generally binary-valued. This does not allow for arbitrary relationships and constraints between

[1]A simple example might be a processor that can handle two users by either allocating all of its resources to one user, or multitasking and incurring the overhead of context switching constantly between users. The configurations would be $(1, 0)$, $(0, 1)$, and $(0.5 - \alpha, 0.5 - \alpha)$ where 2α is the overhead cost.

resources. Our model is a user-centric one, where resources are defined as they appear to the users. We assume that the actual configuration of the resource is done transparently by the resource itself, and the scheduling rule's job is simply to choose an appropriate configuration.

Although in many cases the models are described in a way that would make the geometry clear, the rules themselves are justified by appealing to general intuitions of good performance. In [30], [51], [66, 65] and much of the GPS literature, although geometric notions are used in the proofs, the rules are not themselves interpreted geometrically. Part of our goal is to present a unifying geometric framework in which all of these rules can be looked at together.

It turns out that many of the rules used in the work just cited can be interpreted geometrically. Also, this interpretation gives us insight into why some of the rules may work and why some do not (or have not been proven to) in a dynamic setting. These examples will be discussed in more detail at the end of this chapter.

3.1.1 Applications

Before we get to the modeling of more dynamic allocation problems, let us get familiar with the types of problems we will be considering in this static context, specifically a few applications of our model to real scheduling problems. These applications will be described as simple static problems for now, but will be taken up in a dynamic form later.

Independent Processor Sharing

The first and most obvious application is that of allocating M identical independent processors among N competing processes. The set of possible schedules, $\mathcal{S}$, consists of all possible ways of allocating the processors to the processes, or all possible nonnegative integral vectors $\mathbf{s}$ of length N (assuming processes can make use of any number of processors) whose components sum to M. In this way, if we are only able to make our scheduling decision a single time, we will choose the schedule $\mathbf{s}$ that most closely approximates the workload $\mathbf{r}$ that the processes would like to put on the processors. If one process requires approximately twice as much work as another, then as long as we are treating them equally, that process should receive twice as many processors. We can also think of this as saying that each process should receive a number of processors that is close to the amount of work that process requires. One way to model this is to use our static allocation mechanism with $d(\mathbf{r}, \mathbf{s}) = \| \mathbf{r} - \mathbf{s} \|_\infty$. This choice minimizes the maximum deviation between the workload of a process and the actual number of processors the process is allocated.

We can extend this model to also cover constraints on the use of processors. If certain processes can not use certain processors, or there are limits on the maximum number for each process, we can capture these constraints by choosing a subset of the previous $\mathcal{S}$. As mentioned before, if there is an overhead cost for processing multiple processes, or for processing certain processes on particular processors, we can also capture this in the set $\mathcal{S}$.

Input-Queued Switch

A specific example of a constrained problem is an input-queued switch. As described in [51, 30, 15], the input-queued switch is a constrained scheduler, where we have M^2 switching elements connecting each of M inputs to each of M outputs. Each input has M queues, one for each of the outputs. The switch must choose which set of input queues (one from each input) to connect to the outputs without connecting two inputs to the same output. The set of possible schedules $\mathcal{S}$ is the entire set of matrices obtained by row and column permutations of the identity matrix. The element of a schedule in row i and column j represents the number of packets (1 or 0) that are transferred from input i to output j. This set is a very constrained subset of the possible ways of allocating the switching elements to the M^2 queues. The input request vector $\mathbf{r}$ is the matrix containing the numbers of packets in each of the input queues. Again, to keep the input queues short, we would like to choose a schedule $\mathbf{s}$ that is close to $\mathbf{r}$ in some way.

Generalized Processor Sharing

Similar to the above is the generalized processor sharing problem [57]. The basic issue is how to share a single communication link among a number of transmissions. This is similar to a trivial case of the input-queued switch, where we have 1 input and N outputs (each output corresponds to a transmission), except for the fact that each transmission also has a fairness coefficient. The fairness coefficient is a weight that tells how much priority each transmission has over the channel. The basic concept is that if one transmission has twice the priority of another (the fairness coefficient is twice as large), then that transmission should be able to use the link twice as often as the other when they are both competing for it. In this problem, the request vector is a measure of the amount of work in the queues, but weighted by the corresponding fairness coefficient. In this way, if a high priority transmission has few packets waiting, it may still be able to use the link even if the other lower priority transmissions have more packets waiting. We can fit this scenario in our static model by choosing the distance measure appropriately. The possible schedules are simply the unit vectors with a 1 in one location and 0's elsewhere. The distance measure can be seen to be related to $d(\mathbf{r},\mathbf{s}) = \| \mathbf{r} - \mathbf{s} \|_\infty$.

Auctions

The last application of our model we will look at is in auction-based resource allocation. Auctions can be used for many of the same resource allocation problems we have been discussing. The basic idea of a closed-bid single round auction is the same: a collection of users submits bids (or requests) for a shared resource. The allocation mechanism depends on the type of auction. Normally the highest bidders win, but the payment often depends on the bids themselves. This payment issue is what makes auctions an interesting contrast to the mechanisms we have mentioned so far. Our previous allocation mechanism implicitly assumed that the "payment" was simply the fact that the queue lengths, or amount of buffered data, decreased if a user was allocated some part of the resource. Thus, the next time around, the user would have less work and hence lower priority. However, auctions have separate methods of payment, and so the amount of payment may not coincide with the amount of resource allocated. We will show later that this can lead to systematic biases in simple dynamic auction-based allocation mechanisms. Apart from the payment issue, the decision making process in an auction is very similar to that of the allocation mechanisms we have been looking at. Instead of being described as a distance minimizing function, the auction mechanism uses a set of rules to decide the winners in an auction, but these rules constitute a function that maps bids (requests) to schedules, just as the scheduling rules we described earlier.

3.1.2 Allocation Mechanism as Quantizer

The static allocation mechanism we have presented here can be interpreted as a *quantizer*. There is a long history of work in the area of quantization; see [23] for a good overview of the fundamentals. As our model is stated above, it is immediately clear that it is equivalent to a kind of quantization problem. The description of the static model boils down to a very simple problem: given a particular input, find the member of a finite set of valid allocations that minimizes the distance between the allocation and the input. This divides up space into a collection of allocation regions, one for each possible allocation. The result is structurally equivalent to a vector quantizer [21], where we have a constellation of quantization points in $\mathbb{R}^N$, and an error metric. Each input point is quantized to the point in the constellation that minimizes the error between the input and the quantization point.

Much effort has been put into the examination and understanding of vector quantization and vector quantizer design (see [23] for a number of citations). Vector quantizer design approaches concentrate on the design of quantizer constellation such that the quantizers have small distortion. The applications we are interested in are resource allocation problems, and it is not always a question of designing the geometry of the set $\mathcal{S}$ directly, but designing the underlying resource structure which then determines the set $\mathcal{S}$. Existing vector quantization results prove useful both as inspiration and guidance, but as we show in Chapter 5, the set $\mathcal{S}$ can be constrained in different ways than in typical vector quantization applications. Whereas in quantizer design problems, a common goal is to minimize the expected error given a restriction on the number of quantization points allowed, the goal in Chapter 5 is to minimize the error given a restriction on the maximum number of pieces we can chop a single resource into. This restriction does not restrict the number of points directly, but does play a large role in determining the geometry of the set $\mathcal{S}$.

3.2 Dynamic Model

We have considered a static resource allocation model so far; we would like to extend the model now to include repeated allocation problems. We will concentrate on an extension that assumes a given resource is available to be scheduled during a sequence of rounds. A schedule is chosen at the beginning of each round and kept until the end of the round.

Consider a situation in which a resource with a *schedule constellation* $\mathcal{S}$ of possible configurations or *schedules* is to be repeatedly allocated to a set of N users during a sequence of allocation rounds by choosing an appropriate schedule during each round. We denote the desired fraction of the resource or *request* for user i during round n by $r_i[n]$ at round n, and the collection of requests by $\mathbf{r}[n]$. At the beginning of round n, a new schedule $\mathbf{s}[n]$ is chosen by a scheduler using a static mechanism according to our model from Section 3.1, and the users employ the resource according to the schedule for the duration of the round. The actual resource request $\ell[n]$ of a user at round n is equal to the request for that round plus whatever requests have not been met up until this point. The situation is shown in Figure 3-1, which differs from Figure 1-2 in that we now have a (possibly random) matrix $B[n+1]$ that shapes the effects of the allocation on the system. The actual schedule that is used is the product of a matrix $B[n+1]$ and the desired schedule $\mathbf{s}[n]$. The goal of the scheduler in the case of constant $\mathbf{r}[n] = \bar{\mathbf{r}}$ is to ensure the sequence of allocations $B[n+1]\mathbf{s}[n]$ has a time average close to the time average of the request vector $\mathbf{r}[n]$. The matrix $B[n+1]$ is one method of modeling indirect effects of the schedule on the system. We are allowed to choose $\mathbf{s}[n]$ arbitrarily, but it is then multiplied by $B[n+1]$ (which may be unknown at the time we choose $\mathbf{s}[n]$) in order to compute the actual resource allocation for the round. For example, $B[n+1]$ might represent the connectivity of a network of queues with probabilistic service times, as we will see later. Alternatively, there may only be a probability that the resources that we assign to each user are available, in which case $B[n+1]$ might be a diagonal matrix with Bernoulli random variables on the diagonal. In the simplest case, $B[n+1] = \mathrm{I}$, and $\mathbf{s}[n]$ directly affects $\ell[n+1]$.

We make a distinction between two classes of scheduling models. The first is the *deterministic model.* In this case, we view $B[n]$ as a constant matrix and $\mathbf{r}[n]$ as a deterministic, possibly time-

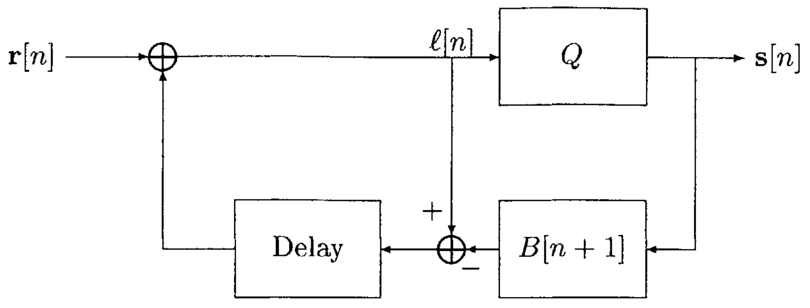


Figure 3-1: Block diagram of lag evolution equation.

varying, sequence of vectors. We can think of $\mathbf{r}[n]$ as modeling the relative amounts of different types of fluids entering a system. The second class is the general *stochastic model.* This includes all other stochastic characterizations of $\mathbf{r}[n]$; for example, ones in which $\mathbf{r}[n]$ models the arrivals to a queueing system. The deterministic model is important for a few reasons. First, it gives us a simpler model to think about. The scheduling mechanisms (with static scheduling rules) we have discussed become deterministic nonlinear systems, and can be visualized very easily. Because of this simpler characterization and visualization of the deterministic models, they allow for simpler (although not necessarily simple) analysis. Also, as we discuss in the next chapter, many of our stability results for deterministic models apply when we let $\mathbf{r}[n]$ vary over time within the confines of the constellation. This allows the results to apply to a larger set of models. Second, we might think of using a deterministic model as an approximation of a stochastic system in order to design scheduling rules more easily. However, although the analysis of the stability of a deterministic model is sometimes easier, in general, the stability of a deterministic model does not necessarily imply stability of a corresponding stochastic model.

We use a scheduling rule defined using the notion of a *lag* or mismatch between the accumulated scheduling choices and the accumulated requests. The lag evolution equation of the scheduler at round $n+1$ is defined — consistently with Figure 3-1 — to be

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - B[n+1]\mathbf{s}[n] \ . \tag{3.1}$$

Note that $\ell[n]$ is allowed to have negative components, i.e. 'lead' components. The block diagram in Figure 3-1 suggests that we can actually think of the scheduler as a nonlinear dynamic system, where the requests $\mathbf{r}[n]$ comprise the input signal, and the schedules $\mathbf{s}[n]$ comprise the output signal. This interpretation will be taken up in this and the next chapters, when we discuss the connection with Σ-Δ encoders.

We may also view the schedule $\mathbf{s}[n]$ as a control input to the lag evolution equation. At each round, $\mathbf{s}[n]$ is chosen as a function of $\ell[n]$. The way that $\mathbf{s}[n]$ is chosen depends on the information available and the specific goals of the scheduler. The input $\mathbf{r}[n]$ and matrix $B[n]$ will be modeled in one of a number of ways. The simplest deterministic problem we consider is where each is a constant at each round, $\mathbf{r}[n] = \bar{\mathbf{r}}$ and $B[n] = \bar{B}$. We also extend this to an $\mathbf{r}[n]$ which varies arbitrarily (including stochastically) in a region $\mathbf{R}$ that is a compact set lying within the constellation. Our stochastic models include an uncorrelated model, where $\mathbf{r}[n]$ and $B[n]$ are both uncorrelated sequences of random variables, as well as a more general model where both $\mathbf{r}[n]$ and $B[n]$ are stationary random processes with well-defined means and bounded higher moments.

The lag vector can be interpreted as a point in N-dimensional space that evolves over time. With this interpretation, we will view $\mathbf{r}[n+1]$ as a disturbance that moves the lag vector away from

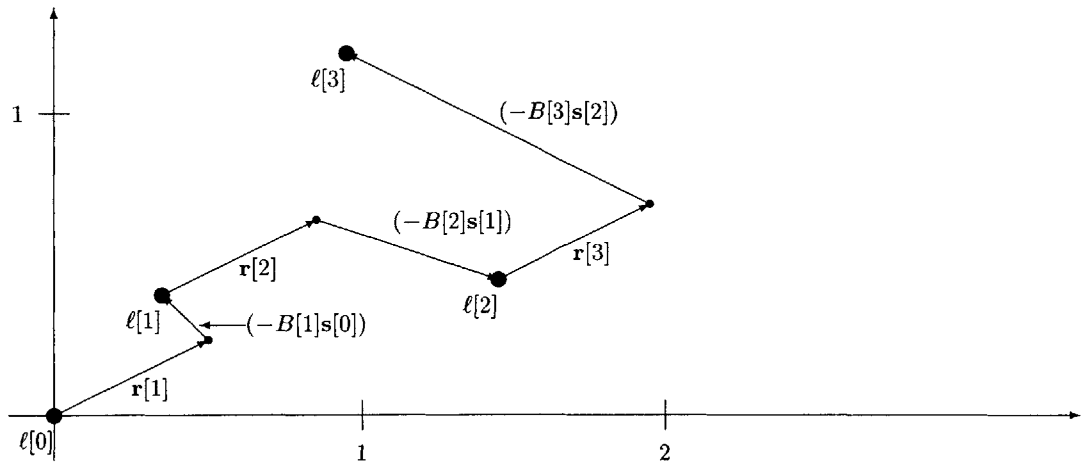


Figure 3-2: Lag evolution example with a fixed $\mathbf{r}[n] = (0.5, .25)$.

its current position, and we will try and choose $\mathbf{s}[n]$ so that $B[n+1]\mathbf{s}[n]$ counteracts this effect to some degree. Figure 3-2 shows an example of how the lag might evolve over time.

Two concepts that will come up later on in relation to this model are the set of *feasible* input rates and the *convex hull* $\mathrm{CH}(\bar{B} \cdot \mathcal{S})$ of a constellation of points $\bar{B} \cdot \mathcal{S}$. The set of feasible input rates is defined as the set of all vectors $\bar{\mathbf{r}}$ such that $\bar{\mathbf{r}}$ is in the interior of $\mathrm{CH}(\bar{B} \cdot \mathcal{S})$.

A *convex set* C is defined as a collection of points such that for any two points $c, d \in C$, all points $\alpha c + (1-\alpha)d \in C$ for $0 \le \alpha \le 1$. We will only be concerned with closed convex sets, and will simply call them convex. The *convex hull* of a collection of points $\mathcal{S}$ is the collection of all points that are convex combinations of points in $\mathcal{S}$. From this definition, a convex hull is a convex set. This makes the convex hull of a finite collection of points (our constellation) a polytope whose surface consists of a collection of flat faces [5].

Definition 3.1 (Convex Hull) *Given a set of N-dimensional points $\mathcal{S}$, the* convex hull *of $\mathcal{S}$, $\mathrm{CH}(\mathcal{S})$, is the set of points r such that*

$$r = \sum_{s \in \mathcal{S}} \alpha(s)s \ ,$$

for some set of $\alpha(s)$ where $\sum_{s \in \mathcal{S}} \alpha(s) = 1$ and $0 \le \alpha(s) \le 1$.

We define the interior of the convex hull, $\mathrm{ch}(\mathcal{S})$, similarly with the convex hull, except that we constrain $\alpha(s)$ so that $\sum_{s \in \mathcal{L}} \alpha(s) < 1$ for all $\mathcal{L} \subset \mathcal{S}$. This is equivalent to saying that a point in $\mathrm{CH}(\mathcal{S})$ is in the interior of the convex hull if it does not lie on the surface of the hull. Another way to define the interior is as the set of points for whom a finite neighborhood exists that is contained in $\mathrm{CH}(\mathcal{S})$, i.e. every point in the interior is a positive distance away from the surface of the hull.

Other properties of convex sets we use for our proofs are given in Appendix A.

3.2.1 Fairness and Scheduling

The example in Figure 3-2 shows how the lag evolves for a specific sequence of choices for $\mathbf{s}[n]$. These choices are made by a *scheduling rule.* In the scheduling applications we consider later in this chapter, the scheduling rule is designed to keep the lag vector bounded, in expectation or in actual value. If we know that the absolute value of the elements of the lag vector stay bounded below some finite number b for all rounds n, then the time average schedule will converge to the desired request $\bar{\mathbf{r}}$ when the request vector is constant. The smaller the absolute value of the lag, the closer the system is to maintaining an allocation of the resources close to the requests. To see this, first note that

$$\frac{\ell[n] - \ell[n-k+1]}{k} = \bar{\mathbf{r}} - \frac{1}{k} \sum_{m=n-k+1}^{n} \mathbf{s}[m] \ . \tag{3.2}$$

With each component of the lag bounded by $\pm b$, each component of the left side of Equation (3.2) is bounded by $\pm 2b/k$, which approaches 0 for increasing k. This implies that the time average of $\mathbf{s}[n]$ approaches $\mathbf{r}$ as the time window grows larger. Also, for smaller b, the time average schedule is closer to $\bar{\mathbf{r}}$. We can also use this argument to show that the time average of $\mathbf{s}[n]$ converges to $\bar{\mathbf{r}}$ in expectation if the system is stochastic. Also, if the variance of the lag at any round is bounded, the time average of $\mathbf{s}[n]$ converges asymptotically to $\bar{\mathbf{r}}$, i.e. the variance of the difference decreases to 0 as n increases. We say the system is *fair* if the size of the difference between $\bar{\mathbf{r}}$ and the time average of $\mathbf{s}[n]$ can be driven arbitrarily small.

Definition 3.2 (Deterministic Fairness and Bounded Fairness) *A scheduling mechanism is* deterministically fair *if for every $\epsilon > 0$, there exists a $k_0 > 0$ such that for all $n > n_0$ for a finite n_0 and for all $k \geq k_0$:*

$$\| \bar{\mathbf{r}} - \tfrac{1}{k} \textstyle\sum_{m=n-k+1}^{n} \mathbf{s}[m] \| = \frac{\| \ell[n] - \ell[n-k+1] \|}{k} \leq \epsilon \ .$$

We will call a system bounded fair *if the lag is constrained to stay within some compact set.*

The function $\| \cdot \|$ is an arbitrary norm. Bounded fairness implies deterministic fairness because if $\ell[n]$ is in a compact set, then $\| \ell[n] \|$ is bounded by some finite C and by the triangle inequality we can satisfy the definition of deterministic fairness:

$$\frac{\| \ell[n] - \ell[n-k+1] \|}{k} \leq \frac{\| \ell[n] \|}{k} + \frac{\| \ell[n-k+1] \|}{k} \leq \frac{2C}{k} \leq \epsilon$$

for $k \geq 2C/\epsilon$. Most of the deterministic cases we look at will be bounded fair, but the second-price infinitely-repeated auction in Chapter 7 seems to be deterministically fair but not bounded fair. We will show in the next chapter that a deterministic mechanism that satisfies a drift condition is bounded fair. We can also extend this definition to a stochastic setting by simply bounding the expected value of the norm of the difference between lag values.

Definition 3.3 (Stochastic Fairness) *A scheduling mechanism is* stochastically fair *if for every $\epsilon > 0$ there exists a $k_0 > 0$ such that for all $n > n_0$ for a finite n_0 and for all $k \geq k_0$:*

$$\mathbf{E}\left[\frac{\| \ell[n] - \ell[n-k+1] \|}{k}\right] \leq \epsilon \ .$$

This is a rather weak definition; it does not place any constraints on any other characteristics of the lag. For instance, the variance of the lag may grow to ∞ with n. Stronger notions of stochastic fairness can be defined only when we know more about the statistics of the lag. Later we will show that if a mechanism satisfies a stochastic drift condition then it is *bounded stochastically fair.* This means that $\mathbf{E}[\| \ell[n] \|] \leq C < \infty$ which is again satisfied when $\ell[n]$ lies in a compact set, but also includes cases when $\ell[n]$ may take values in the entire state space. As before, a bounded stochastically fair mechanism is stochastically fair:

$$\mathbf{E}\left[\frac{\| \ell[n] - \ell[n-k+1] \|}{k}\right] \leq \mathbf{E}\left[\frac{\| \ell[n] \|}{k}\right] + \mathbf{E}\left[\frac{\| \ell[n-k+1] \|}{k}\right] \leq \frac{2C}{k} \leq \epsilon$$

for $k \geq 2C/\epsilon$. This guarantees us that the total allocation to a user over a long time window is close to the accumulated requests of the user during the window in expectation.

If the scheduling rule we use is a static rule as in Section 3.1 (also called a stationary rule) that only depends on the current lag vector, we can simply map each point in $\mathbb{R}^N$ to a specific schedule $s \in \mathcal{S}$. This gives us a simple interpretation of the model and scheduling rule. We can picture the lag vector moving from point to point in $\mathbb{R}^N$; at each point, there is a fixed value of $\mathbf{s}[n]$ that is used to compute the next value of the lag. In addition to the geometric interpretation, this structure can also be seen as a randomized state machine, or Markov chain in some cases, which will allow us to establish the stability results in Chapter 4. The problem is now to choose a specific static scheduling rule in order to keep the lag bounded. We will generally denote scheduling rules by $Q(\cdot)$, where $Q(\cdot)$ maps lag vectors to schedules, or $Q : \mathbb{R}^N \to \mathcal{S}$. Using a specific scheduling rule, the lag evolution equation becomes

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - B[n+1]Q(\ell[n]) ,$$

and becomes a nonlinear dynamic equation driven by the input $\mathbf{r}[n]$.

3.2.2 Relation to other Models

This model is similar to models considered in [65, 52, 30]. However, it differs from each of them in that it maintains a simpler geometric interpretation. The model in [52] is identical in flavor, and as stated contains more complexity. However, there are a few important differences. Our model allows for sparse constellations with arbitrary geometry, while [52] requires that the control constellations take on integer values, and be restricted to polytopes (more precisely, the integer subset of a polytope), or the product of these polytopes and integer matrices.

The model in [65] selects controls with a fixed matrix B (same as in [52]). The model in [65] also allows for random processing times at the queues. This situation is captured by our dynamic model when, for example, the processing times are geometrically distributed, which is modeled by multiplying B with a random matrix M as in [66].

The model in [30] is a subset of [65]. It is a model with no nontrivial routing (simply a bank of processors). However, it does have a characteristic not present in the model of [52]. As in [65], the model is a constrained queueing model, where the status of a queue (scheduled or not) may influence whether or not other queues may be scheduled. This makes it impossible at times to define non-idling schedules. However, it is still possible to show that these systems are stabilizable, as [65] does for constrained queueing systems and as we establish for our general stochastic model later in this chapter.

3.3 Quantizing Schedulers

For most of the rest of the thesis, we will restrict ourselves to a specific class of scheduling rules $Q(\cdot)$ we call *quantizing scheduling rules.* A quantizing scheduling rule is defined in terms of a norm; in Section 4.2 we discuss how we might extend the class of scheduling rules to include other measures of distance. We define a *quantizing scheduler* as an instance of the dynamic model from the previous section, where the static allocation mechanism is a minimum distance quantizer and the output of the quantizer is used as the feedback. As before, we can view the overall scheduler in terms of the state trajectories. However, the dynamics of the system become much simpler. We can simplify the deterministic model even further and view the problem of determining the stability of a scheduler as an instance of the canonical problem presented in Chapter 1.

The model we use for a quantizing scheduler is defined with respect to the same dynamic equation as before:

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - B[n+1]\mathbf{s}[n] \ ,$$

where $\mathbf{s}[n]$ is chosen from a schedule constellation $\mathcal{S}$, a finite set of points in $\mathbb{R}^N$. Our overall goal is to keep the lag small, in order to maintain an average resource allocation close to the requests $\mathbf{r}[n]$. However, at time n, we only have access to $\ell[n]$ in order to make our decision. We present one particular method of making the decision below, but it requires knowledge of the statistics of $\mathbf{r}[n]$ and $B[n]$ to compute the expectation. We may also use a similar rule that chooses $\mathbf{s}[n]$ based only on past, known values of $\mathbf{r}[n]$; this is used in cases when $B[n]$ does not change. We discuss this more in the next chapter.

However, here we assume full knowledge of the statistics of $\mathbf{r}[n]$ and $B[n]$. If our goal is to keep the lag small, an obvious choice of schedule is one that minimizes some measure of the size of the lag at the next round $\ell[n+1]$, given the lag at the current round $\ell[n]$. If the size measure that we use corresponds with a metric or norm, we have a quantizing scheduler. We choose a schedule at round n that minimizes a measure of the size of the expected lag:

$$\begin{aligned} \mathbf{s}[n] &\in \arg\min_{s\in\mathcal{S}} \mathbf{E}\left[d(\ell[n] + \mathbf{r}[n+1], B[n+1]s) \mid \ell[n]\right] \\ &\in \arg\min_{s\in\mathcal{S}} \mathbf{E}\left[d(\ell_s[n+1], 0) \mid \ell[n]\right] \end{aligned} \tag{3.3}$$

The function $d(\ell + \mathbf{r}, \mathbf{s})$ is a metric that tells us the distance between $\mathbf{s}$ and $\ell + \mathbf{r}$, just as we used in Section 3.1. In some cases we may also consider a scheduling rule that chooses a schedule when $B[n+1]$ is known:

$$\mathbf{s}[n] \ \in \ \arg\min_{s\in\mathcal{S}} d(\ell[n], B[n+1]s) \ .$$

This rule is used for example in queueing contexts when we must decide which jobs to process based on the current number in queue, not including the new arrivals, $\mathbf{r}[n+1]$. It may also be useful when we do not know the statistics of $\mathbf{r}[n]$ and so can not compute the expectation above. We do not explicitly present proofs of stability for mechanisms using this type of scheduling rule, but the proofs we present can be easily modified to accommodate them. The actual scheduling rule $Q(\cdot)$ that we use to make the choice is said to be *consistent* with a metric $d(\cdot)$ if $Q(\cdot)$ always chooses a schedule that minimizes the measure in (3.3). A common metric is distance under the p-norm, or

$$d(a, b) = \| \ a - b \ \|_p = \left(\sum_{i=1}^{N} |a_i - b_i|^p \right)^{\frac{1}{p}} \ .$$

When $p = 2$, we have the familiar 2-norm. Others that are important are $p = 1$ and $p = \infty$ which give us the sum of the components in the absolute difference and the maximum of the absolute difference. In general, the metric $d(\mathbf{x}, \mathbf{y})$ will be a function satisfying the following defining properties:

- $d(\mathbf{x}, \mathbf{y}) \geq 0$, and $d(\mathbf{x}, \mathbf{y}) = 0$ only when $\mathbf{x} = \mathbf{y}$;
- $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$; and
- $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$ for all $\mathbf{z}$.

We will concentrate mainly on metrics induced by a norm $\| \cdot \|$, or $d(\mathbf{x}, \mathbf{y}) = \| \mathbf{x} - \mathbf{y} \|$ but will relax this at times. Again, we say a scheduling rule $Q(\cdot)$ is *consistent* with a norm $\| \cdot \|$ if it is consistent with the corresponding induced metric. In the case of a norm, we have the scaling property $\| \alpha\mathbf{x} \| = |\alpha| \| \mathbf{x} \|$ and the triangle inequality $\| \mathbf{x} + \mathbf{y} \| \leq \| \mathbf{x} \| + \| \mathbf{y} \|$. One property of norms that we make use of is the convexity property of the sub-level sets of a norm (see Appendix A). A *sub-level set* is simply the collection of all vectors whose norm is no larger than some maximum value. For a norm $\| \cdot \|$, a sub-level set is the set of points $\{v \mid \| v \| \leq D\}$ for some $D \geq 0$. The following establishes that sub-level sets are convex.

Theorem 3.1 (Convexity of Norm Sub-level Sets) *Given a norm $\| \cdot \|$, the sub-level set $D_c = \{x \mid \| x \| \leq c\}$ is a convex set.*
Proof: First, note that we only have to show it for D_1, because the scaling property of the norm tells us it is then true for all other D_c. Assume that D_1 is not convex. Then, there exist two points x_1 and x_2 such that $x_1 \in D_1$ and $x_2 \in D_1$ but such that $x_3 = \alpha x_1 + (1-\alpha)x_2 \notin D_1$. Thus, we know that $\| x_3 \| > 1$. But this means that

$$\| \alpha x_1 + (1-\alpha)x_2 \| = \| x_3 \| > 1 \geq \| \alpha x_1 \| + \| (1-\alpha)x_2 \| ,$$

which violates the triangle inequality. ■

The set of norms can be characterized as the set of functions whose sub-level sets are linearly scaled versions of a convex set symmetric about the origin. Another property that is important is whether or not the norm is *smooth.*

Definition 3.4 (Smooth Norm) *A norm $\| \cdot \|$ is* smooth *if the surface of any sub-level set of the norm has bounded curvature everywhere. Equivalently, we can say that the gradient of a smooth norm is continuous.*

A p-norm, as defined above, can be either smooth or not. We define *corner norms* as all norms that are not smooth. The class of corner norms includes the ∞-norm, the 1-norm, any norm whose sub-level sets are polytopes, and many others. The p-norms for which p is an integer not equal to 1 are smooth norms. The essential property of a corner norm is that the surface of the sub-level set has at least one corner, or a point with infinite curvature.

When we talk about a quantizing scheduler, we need to specify more than a distance metric $d(\cdot)$. Depending on the constellation and lag, there may be several schedules that are equally attractive. A single metric may give rise to a number of possible scheduling rules, depending on the way that ties are broken. There are times when we will make a distinction between a scheduler based on a particular metric, and a specific scheduling rule that is consistent with that metric. Most of our stability results will deal with all scheduling rules consistent with a specific metric, although there are pairs of scheduling rules that are both consistent with a particular metric, where one stabilizes a model and the other does not. We give an example of this later, in Section 3.5.1.

In our subsequent discussions, we will investigate such scheduling rules, with varying choices for the characteristics of the model. The simplest models have $B[n] = I$ and $\mathbf{r}[n] = \bar{\mathbf{r}}$, a constant vector, with a simple constellation $\mathcal{S}$. The more complex models have arbitrary constellations and time-varying $B[n]$ and $\mathbf{r}[n]$.

Our goal in the next chapter is to investigate the stability behavior of a scheduler model under various choices of $d(\cdot)$, concentrating on understanding the behavior through a geometric interpretation. We show in Section 3.5 how a number of existing scheduling algorithms can be modeled this way, and will now show that a variety of scheduling problems (along with specific algorithms) fall within our framework, and thus can be interpreted as dynamic systems with discrete feedback control mechanisms.

3.4 General Applications

In addition to the queueing applications we discuss in the next section, our model is applicable in other contexts. The first obvious context is in modeling vector Σ-Δ encoders. The model as we have stated it so far is exactly equivalent to a first-order vector Σ-Δ encoder, with an arbitrary vector quantizer in the feedback loop. Generally, these encoders operate on continuous state spaces, and have inputs $\mathbf{r}[n]$ that vary with time. The matrix sequence $B[n]$ is typically equal to the identity matrix, but could possibly be used to model external effects. A vector Σ-Δ encoder model requires the aspects of our model that differentiate it from other queueing models and flow models [52]. The encoder requires both a continuous state space with continuously varying input operating in discrete time and a quantizer that has a finite number of possible outputs. The fact that deterministic models with time-varying inputs can be shown to be stabilizable by any smooth quantizing scheduler implies that vector first-order Σ-Δ encoders whose inputs lie within the convex hull of the quantization points are also stabilizable if the distortion metric is a smooth norm. More interesting, perhaps, is the fact that vector Σ-Δ encoders can be unstable not only in the sense that is currently understood for higher-order encoders in the literature, but also for first-order vector encoders that use a non-smooth distortion measure. No existing results that we know of have exposed this potential instability.

Our model is also similar to the storage model [53], but restricts the release rule to be a quantizing rule. Inputs arrive over time, and accumulate. The accumulated inputs serve as the state which is then used to decide when to release some of the accumulation. The model here looks at the release rule as a quantizing rule and establishes appropriate stability results. Again, this differs from the traditional model in which the release rule is either a constant, or a continuous control variable.

Another setting in which the general model might make sense would be in a chemical manufacturing plant, where the availability of reagents for various processes dictates which processes will be run and for how long each day. If reagents arrive at a time-varying rate then each day we need to decide how to use the available reagents. Each process can be associated with a constellation point, and determines an amount of a number of different reagents that is consumed to sustain the process for the day. The lag of the system is the available amount of each reagent. This applies most directly if we assume there is always enough of each reagent to feed all of the processes or if we assume that we can run a process partially if some reagent is limited.

3.5 Queueing Networks

In this section, we describe a number of specific contexts in which we show how our framework relates to existing work. We look at queueing models of resource allocation systems, and present another view of a number of different models that unifies some of their basic characteristics. Our model lacks some of the complexity of many of the existing models in order to make the connections more transparent, but still results in an interesting synthesis of a number of separate models. The queueing models we consider are multi-class queueing networks (specifically input-queued switches), constrained queueing networks (specifically packet radio networks), and fair queueing systems.

The study of the stability of multi-class queueing networks under various scheduling policies is a very active area of research [38, 4, 13, 52]. This area is related to specific problems such as input-queued switch scheduling [30, 51, 15, 36] and the packet radio routing problem [66, 64] which fall in the constrained queueing network framework [65]. We will also look in some detail at the basic generalized processor sharing (GPS) fair queueing model as formulated in [57].

All of these models fit within our general dynamic scheduling model as a result of the underlying queueing structure. The basic dynamics of a typical queueing network model are exactly those of our model: work arrives at the queues from internal and external sources according to some stochastic process, and is processed according to a scheduling rule and then continues to the next appropriate server or exits the network, in the process reducing the amount of work remaining. We will concentrate here on a simple discrete-time queueing model in which the work arrives as integer-sized jobs according to some arrival process, and is routed through the network according to a routing matrix.

One subtlety of this model when used for a queueing network is that in queueing contexts the number of packets scheduled during a round is constrained to be no more than the number waiting in queue. Our quantizing scheduler does not explicitly make this constraint — it would require the constellation to depend on the current state. However, because the queue lengths are integral and the number of jobs taken by a particular queue is either 0 or 1 in most of the examples we give here, the constraint is satisfied by the scheduler due to the structure of the feasible states. If a queue has no packets waiting, then it is clear that we will schedule none. If a queue has any packets remaining, there is always at least one packet and so the queue length is larger than the possible schedule. The one exception is the fair queueing example at the end of this section. The fair queueing model changes its behavior when queues are empty, and so its discussion is more complex.

This constraint also might not hold for more complicated queueing networks, such as ones that schedule either 0 or 3 packets each round. If there are 2 packets in queue then we may choose to schedule 3, leaving us with a queue length of -1 in our unconstrained model. This case does not happen in the examples below, but we do consider a related system in Section 6.5.

There are N queues, or buffers, which receive external arrivals at the beginning of each round, as well as arrivals from other servers within the network. There are M $(\geq N)$ servers which can be configured in a number of different ways. We view these configurations in terms of the number of jobs taken from each of the buffers, or the corresponding schedule. The set of these schedules, as before, is denoted $\mathcal{S}$, a set of N-dimensional vectors. The servers process the appropriate number of jobs and then send them to another server or out of the network. After the desired schedule $\mathbf{s}[n]$ is chosen, the actual number of jobs processed and their routing through the network is specified by the *routing matrix* $B[n]$. In the simplest case, each server processes exactly one job every round, and $B[n]$ is simply a static routing matrix whose i-th diagonal element equals 1, and the i, j-th element equals -1 if server j sends output jobs to server i. We can also model more complex situations, where the availability of a server may depend on which other servers are activated, and the routing matrix $B[n]$ may be stochastic, reflecting characteristics of varying packet lengths, server speeds, or routing connections. Specifically, this model can be used to represent discrete-time versions of the models mentioned above as we will show subsequently.

A scheduling rule is used to determine the appropriate configuration at the beginning of each round. If we denote the number of arriving jobs during round n by $\mathbf{r}[n]$ and the number of jobs processed from each buffer (the schedule) during round n by $\mathbf{s}[n]$, then the amount of waiting work $\ell[n]$, or queue length is governed by an equation equivalent to the lag evolution equation, Equation 3.1.

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - B[n+1]\mathbf{s}[n] \ .$$

The work comes at an average rate we call $\bar{\mathbf{r}}$. As we will see in the next chapter, we need only consider average rates such that $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$; a network with some other set of arrival rates would not be stabilizable. We call this set of average rates the set of *feasible arrival rates.* This model has the ability to incorporate many system behaviors into the schedule constellation and routing matrix. Parallel processing overhead, multiple links, and differently-sized links can all be captured in the constellation. If we allow the constellation to vary over time, we can model even more complex situations. We can even track changing reservations over time if we need to. Given a model of the reservation changes, we can modify the constellation configuration to track it.

We start by considering the input-queued packet switching problem as considered in [30, 51, 15]. We show that the scheduling algorithms that have been proposed fall within our quantizing scheduler paradigm. From this we move to reentrant multi-class networks [38], where we discuss how some of the scheduling rules used in the literature fit within our quantizing scheduler model.

Another problem that can be understood using the model we have developed is the constrained queueing model in [65]. Although the constrained queueing model allows more general dynamics than our model, a restricted (but useful) class of discrete-time constrained queueing systems can be represented well by our model. We show how the two models are related and discuss how our model might be extended to include a larger subset of the class of constrained queueing systems.

Finally, we look at how fair queueing algorithms relate to our model. To do this, we develop an extended version of our basic dynamic model that involves two interrelated dynamic equations. We show how modeling the fair queueing system as a shared resource with a time-varying schedule constellation allows us to relate our notion of stability with the notion of fairness.

3.5.1 Input-Queued Packet Switching

We start with a simple scheduling context, called an input-queued packet switch. This turns out to be a special case of a constrained queueing system; we will consider more general constrained queueing networks in Section 3.5.3.

The input-queued packet switching stability problem (for an $N \times N$ switch) can be stated as follows [15, 51]. Packets arrive at a switch from N sources; each packet wants to get to one of N destinations. During each transmission time, the switch can only make connections from source to destination such that no source is connected to more than one destination, and no destination is connected to more than one source; i.e. a one-to-one mapping between the sources and destinations. Given a scheduling algorithm that chooses connections at each transmission time as a function of the queue lengths, do the queue lengths stay bounded if the average arrival rate at each input does not exceed one packet per transmission time and the average arrival rate of packets for each output (across all inputs) does not exceed one packet per transmission time?

If each input has only a single queue, i.e. all arrivals at a particular input are serviced FIFO, the throughput is limited to 58% [32] of the maximum under certain arrival process assumptions. In other words, the average arrival rate of packets arriving at each input, as well as the total rate of packets destined for each output must be less than 0.58 packets per transmission time to ensure stability. This head-of-line blocking problem is well-known, and motivated the move to a more complex structure where each input has a set of N virtual output queues. This way, there are N^2 total queues, one for each input-output connection. Packets that arrive at input i and are destined for output j are placed in queue (i, j). At each round some subset of these queues must be chosen to send packets to the corresponding outputs without using either the same input or the same output. We will restrict ourselves from now on to the input queued switches with virtual output queues. Many of the results are based on computing matchings between the inputs and outputs. A matching is simply a collection of connections between inputs and outputs that does not connect more than one input to a single output or more than one output to a single input. The schedule constellation includes the permutation matrices (row and column permutation of the identity matrix) where a 1 at position $S_{ij}[n]$ indicates that input i be connected to output j during round n. A schedule does not have to be full rank, so it may have fewer than N values equal to 1. Thus the constellation includes all lower rank permutation matrices as well. The matrix of queue lengths at the beginning of round n is $L[n]$. We can model the dynamics of this system as we have already done:

$$L[n+1] = L[n] - S[n] + R[n+1] \ ,$$

where $R[n+1]$ is an integer matrix of arrivals to the queues during round n. When computing norms below, we treat the matrices as vectors, so the norm is computed for an 'unwrapped' vector of length N^2.

- Maximum weighted matching – choose the schedule (permutation matrix) $S[n]$ such that $\langle L[n], S[n] \rangle$ is maximum, where $\langle \cdot, \cdot \rangle$ is the Euclidean inner product.
- Maximum size matching – choose the schedule (permutation matrix) $S[n]$ such that $\langle \text{sgn}(L[n]), S[n] \rangle$ is maximum, where $\text{sgn}(L[n])$ is equal to 1 where $L[n] > 0$, and equal to 0 where $L[n] = 0$.
- Maximal matching – choose a schedule (permutation matrix) $S[n]$ such that there are no more possible input-output connections that can be made without losing the one-to-one mapping between inputs and outputs.
- Maximal weighted matching – a maximal matching that is constructed by "greedily" selecting connections with the largest weight (queue length or credit) one at a time.

The basic results for the input-queued packet switching stability problem with virtual output queues can be summarized as follows:

- A scheduling algorithm which uses the *maximum weighted matching* between the queue length matrix and the schedule at each round can guarantee full throughput [15].
- A scheduling algorithm which uses a *maximum size matching* between the queue length matrix and the schedule at each round appears to guarantee 100% throughput in simulation, but specific examples show that there are admissible arrival rates that cause instability [51].
- A scheduling algorithm which uses a *maximal matching* between the queue length matrix and the schedule at each round is known to guarantee only 50% throughput [15, 36], and a *maximal weighted matching* appears to guarantee 100% throughput in simulation [30].

The first result also is related to the stabilizing schedule in [66], where the scheduling rule is a maximum weighted matching. The second result is known to be not a theoretical guarantee, in the respect that there are admissible arrival rates that cause instability [51]. The third result has been established a few times, but no counterexamples (i.e. admissible arrival rates that cause instability) have been found to the simulation results when a maximal weighted matching is used.

What we show here is that the first two matchings are equivalent to using a simple geometric rule to choose the schedule if we view the schedules and queue length matrices as points in $\mathbb{R}^{N^2}$, and that a specific case of the third matching is as well.

In all three cases, the quantizing scheduler that we consider does not match the form we presented earlier. As we discussed then, there are many ways to approach the problem of making the scheduling choice. In this application, we do not assume that we know the statistics of the incoming packet streams. It is also reasonable to assume that the statistics may not be of much use, as many packet streams are bursty and vary greatly over time. We are also constrained because we must choose a schedule that does not schedule a queue that is empty. For these reasons, instead of choosing the schedule $S[n]$ based on a prediction of the next lag, we simply choose the schedule that results in the largest reduction in the size of the current lag, i.e. minimizes $d(L[n] - S[n])$.

Maximum Weighted Matching

The maximum weighted matching scheduling algorithm can be written as

$$S[n] = \arg\max_{S \in \mathcal{S}} \langle L[n], S \rangle \ .$$

However, consider what $S[n]$ is obtained by computing the following (with $\| \cdot \|_2$ being the 2-norm of the matrix when written as a vector):

$$\begin{aligned} S[n] &= \arg\min_{S \in \mathcal{S}} \| \ L[n] - S \ \|_2^2 \\ &= \arg\min_{S \in \mathcal{S}} \| \ L[n] \ \|_2^2 - 2\langle L[n], S \rangle + \| \ S \ \|_2^2 \\ &= \arg\min_{S \in \mathcal{S}} -2\langle L[n], S \rangle + \| \ S \ \|_2^2 \\ &= \arg\max_{S \in \mathcal{S}} \langle L[n], S \rangle - \frac{1}{2} \| \ S \ \|_2^2 \ . \end{aligned}$$

Finally, we realize that all nonzero elements of $L[n]$ are greater than or equal to 1, so the second term in the last line just enforces the constraint that if $L_{ij}[n] = 0$, then we will choose $S_{ij}[n] = 0$ as well. Otherwise, we would unnecessarily be increasing the second term while not affecting the first term.

This tells us that the maximum weighted matching scheduling algorithm along with the constraint that when $L_{ij}[n] = 0$, $S_{ij}[n] = 0$ is equivalent to choosing the schedule that minimizes the squared distance (or equivalently the distance) between the matrix of queue lengths and the schedule.

Maximum Size Matching

We can also reason similarly about the maximum size matching scheduling algorithm. Consider the following rule:

$$\begin{aligned} S[n] &= \arg\min_{S \in \mathcal{S}} \| L[n] - S \|_1 \\ &= \arg\min_{S \in \mathcal{S}} \sum_{i,j} |L_{ij}[n] - S_{ij}| \ . \end{aligned}$$

Again, it is clear that we choose $S[n]$ so that as many nonzero elements of $L[n]$ as possible line up with 1's in S. Again, the constraint that when $L_{ij}[n] = 0$, $S_{ij}[n] = 0$ follows from the statement; otherwise, we would increase the sum. This schedule is equivalent to the one that would be chosen in the maximum size matching algorithm. However, the maximum size matching does not necessarily guarantee stability because it is based on a corner norm. Our stability results from Chapter 4 tell us that corner norms can cause problems even for deterministic queueing models. A 3×3 counterexample is given in [51] that indicates that, for certain feasible input rates, the queue lengths will grow without bound. The problem with the maximum size matching algorithm as given in [51] is that when there is a choice to be made between more than one possible maximum size matching, the choice is made probabilistically. This can be seen as an artificial restriction on how often a particular choice can be made when two schedules are equally attractive, shrinking the effective convex hull of the average schedule constellation. Instead, if we make a more intelligent choice between multiple options we can achieve stability, at least empirically. For example, if we were to choose the maximum size matching that had the maximum weight, the system would stay stable for the 3×3 example. To our knowledge, no counterexample has been published that establishes that any scheduling rule consistent with the 1-norm is not stabilizable for some set of feasible arrival rates; in another context, static buffer priority policies as used in reentrant multiclass queueing networks are consistent with the 1-norm, and have been shown to maintain stability in those networks in some cases but not in others. We discuss this more in Section 3.5.2.

Maximal Weighted Matching

Finally, we consider a specific version of the maximal matching scheduling algorithm as given in [30]. The maximal matching scheduling algorithm does not specify a unique schedule; rather, it constrains the possible schedules. We consider a specific maximal matching algorithm which we call the greedy algorithm: place a 1 in S at the location of the largest element of $L[n]$; after the 1 is placed, cross out the corresponding row and column of $L[n]$ and repeat until $L[n]$ is completely crossed out or only 0's remain. It is shown in [30] that this produces a maximal matching S. This *maximal weighted matching* is also consistent with a simple geometric rule:

$$\begin{aligned} S[n] &= \arg\min_{S \in \mathcal{S}} \| L[n] - S \|_\infty \\ &= \arg\min_{S \in \mathcal{S}} \max_{i,j} |L_{ij}[n] - S_{ij}| \ . \end{aligned}$$

From this, it is clear that the greedy rule will reduce the maximum of the absolute difference between $L[n]$ and S as far as possible. As such, the greedy rule is consistent with a quantizing scheduler that uses the ∞-norm. The iterative part of the greedy rule specifies how the other elements in S are to be chosen. The ∞-norm does not because only the largest elements of $L[n]$ can affect the ∞-norm; the rest can be specified arbitrarily.

Just as with the maximum size matching, we can use the same 3×3 example to cause instability in the deterministic model when using a scheduling rule consistent with the ∞-norm. Consider a deterministic model with arrival rate matrix

$$\bar{\mathbf{R}} = \begin{bmatrix} .49 & .49 & 0 \\ .49 & 0 & .49 \\ 0 & 0 & 0 \end{bmatrix} .$$

If we start with $L_{ij}[0] = 0$ for all i, j then the switch configuration

$$S[n] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

minimizes the ∞-norm of $L[n] - S[n]$ at every round. However, it is clear that three entries of $L[n]$ grow unbounded under this scheduling rule. This counterexample holds for a large range of arrival rates, but breaks down if the initial condition is perturbed slightly (start with $L_{12}[0] = \epsilon > 0$). The problem with this example is that the actual schedule that we are using is

$$S[n] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

because many of the queues are always empty. This schedule leaves parts of the switch idle even though there is work they could be doing. If we use the above rule, and augment it by enforcing the constraint that the schedule must be a maximal schedule, as the greedy rule in [30] does, then empirically the ∞-norm scheduling rule maintains stability.

As with the maximum size matching above, the unstable example above deals only one scheduling rule that is consistent with the ∞-norm scheduler. We are not aware of any published work that establishes that all mechanisms are unstable which use a scheduling rule consistent with this ∞-norm scheduler when the arrival rates are feasible.

Discussion

We have shown that a number of existing scheduling algorithms that have been used in the input-queued switch scheduling problem are consistent with norm-based quantizing scheduling algorithms. We have also established that the stability of a model when using a norm-based algorithm is not simply a function of the norm itself. In particular cases such as the 1-norm, stability may be dependent on the method by which ambiguities in the scheduling rule are resolved. There is no existing empirical or theoretical evidence that all scheduling rules consistent with the 1-norm or ∞-norm result in unstable mechanisms. It may be that there are consistent scheduling rules which guarantee the stability of these input-queued switching mechanisms for all feasible input rates.

It is interesting to note that we do not have this ambiguity problem for smooth norms; stability is guaranteed based only on the geometric properties of the norm for deterministic versions of this switch model. However, the computation needed for a smooth-norm based scheduling algorithm is generally more complex than the computation required for the 1-norm or ∞-norm, which drives the interest in these corner norms.

3.5.2 Reentrant Multi-class Queueing Networks

The work on reentrant multi-class queueing networks such as in [38, 13] presents an interesting illustration of the complexity of the stability problem. The reentrant model consists of a collection of R servers and N buffers. Each buffer i must be served by a particular server $\sigma(i)$. After a job from buffer i is served, it is routed to some other buffer; i.e. there is a fixed order in which each job must visit the buffers. The added complexity of these networks comes in because more than one buffer can contend for a single server and the server must decide between buffers when making scheduling decisions.

The reentrant queueing model can be captured within our framework by modeling each queue (possibly more than one at each server) as a user seeking to use a resource. We can construct a constellation $\mathcal{S}$ that consists of only the configurations possible, i.e. does not allow more than one queue at a server to receive service simultaneously. By selecting $B[n] = \bar{B}$ as an appropriate routing matrix, we can also model the connections between queues.

If we restrict ourselves to the discrete-time queueing model we consider in this thesis, it can be easily seen that any non-idling scheduling rule which only uses the emptiness (whether or not jobs are waiting) of the buffers in a network in order to make a decision is consistent with the 1-norm based quantizing scheduler. To see this, we note that the 1-norm simply tells us to schedule as many buffers as possible. Because the buffer choice at each station is independent of the choice at any other station (unlike the situation we consider in Section 3.5.3), this corresponds to a simple rule: schedule one of the nonempty buffers at each station; if all the buffers are empty at a station, leave the station idle. This rule leaves open the choice between multiple nonempty buffers. The scheduling policies that are static buffer priority (SBP) policies, such as last-buffer-first-served (LBFS) and first-buffer-first-served (FBFS), are ways of making this choice explicit by ranking all of the buffers in the system and using strict priority to choose between nonempty buffers. The LBFS and FBFS policies have been shown to be stable [42], while other SBP policies can lead to instability in certain example networks [13, 37]. This tells us that the 1-norm is not enough to guarantee stability, as will also be apparent when we discuss the difference between smooth and non-smooth norms in the next chapter. These instability results actually tell us that the 1-norm can be unstable in a queueing context, not just in the simple examples we presented earlier. There are many other rules that are consistent with the 1-norm for these networks; SBP policies are an easily-characterized subset. If we instead use a squared 2-norm based quantizing scheduler, we know by the stochastic stability results in Chapter 4 that we will guarantee stability for all stabilizable arrival rates.

3.5.3 Constrained Queueing

Another scheduling problem, which includes the previous input-queued packet switching problem as a special case, is the *constrained queueing* system in [66] (a precursor to [65]). In this case, at each round N servers (edges in a graph) are used to process customers of J different classes in L queues (nodes of a graph). The customers are queued by class at each of the nodes. Customers enter the system at any node, and leave at some subset of the nodes as determined by the customer's class. The set of possible activation vectors for the classes $\{\mathbf{e}_j\}$, which specify which servers serve class j at each round, are constrained so that the sum of the vectors across all classes j lies within a set $\mathcal{S}$. This models a situation in which only certain subsets of the servers can be activated at any given round. At the beginning of each round, the vectors $\{\mathbf{e}_j\}$ must be chosen in such a way as to keep the total number of jobs in the system bounded; i.e. route the customers through the network.

Reentrant networks are a special case of constrained queueing networks; we can model each server in a reentrant network as a collection of equal virtual servers, one for each input buffer. Then, by constraining only one of these virtual servers to operate at a time, we can place reentrant networks in the constrained queueing context.

We start by considering the case when there is only a single class of customers.[2] According to the model in [66], and defining $\mathbf{x}[n]$ as the vector of queue lengths at round n, the dynamics of the queue are

$$\mathbf{x}[n] = \mathbf{x}[n-1] + RM[n]\mathbf{e}[n] + \mathbf{a}[n] \ ,$$

where $\mathbf{a}[n]$ is the vector of arrivals at time n, $\mathbf{e}[n]$ is the schedule (activation vector), R is the $L \times N$ routing matrix that is defined by the interconnection graph of the servers and queues, and $M[n]$ is a diagonal matrix with a 1 in location $M_{ii}[n]$ if server i finishes processing a customer during round n. The matrix R plays the same role as $-B[n]$ for this queueing model.

In order to compare the actual scheduling algorithm used in [66] with a simple geometric rule, we first define $\mathbf{d}[n]$ as

$$\mathbf{d}[n] = -(RM[n])^{\mathrm{T}}\mathbf{x}[n] \ ,$$

where R and M are as above. In [66], with $\bar{\mathbf{d}}[n] = \mathbf{E}[\mathbf{d}[n] \mid \mathbf{x}[n-1]]$ the schedule $\mathbf{e}[n]$ is chosen such that

$$\mathbf{e}[n] = \arg\max_{\mathbf{e}\in\mathcal{S}} \bar{\mathbf{d}}^{\mathrm{T}}[n-1]\mathbf{e} \ .$$

If we use a minimum 2-norm scheduling rule as follows

$$\mathbf{e}'[n] = \arg\min_{\mathbf{e}\in\mathcal{S}} \mathbf{E}\left[\| \ \mathbf{x}[n-1] + RM[n]\mathbf{e} \ \|_2^2 \mid \mathbf{x}[n-1]\right] \ ,$$

and define $\bar{M} = \mathbf{E}[M[n]]$ we have the following

$$\begin{aligned}
\mathbf{e}'[n] &= \arg\max_{\mathbf{e}\in\mathcal{S}} \ -\langle \mathbf{x}[n-1], R\bar{M}\mathbf{e}\rangle - \frac{1}{2}\,\mathbf{E}\left[\| \ RM[n]\mathbf{e} \ \|_2^2\right] \\
&= \arg\max_{\mathbf{e}\in\mathcal{S}} \ -(\mathbf{x}^{\mathrm{T}}[n-1]R\bar{M})\mathbf{e} - \frac{1}{2}\,\mathbf{E}\left[\| \ RM[n]\mathbf{e} \ \|_2^2\right] \\
&= \arg\max_{\mathbf{e}\in\mathcal{S}} \ \bar{\mathbf{d}}^{\mathrm{T}}[n-1]\mathbf{e} - \frac{1}{2}\,\mathbf{E}\left[\| \ RM[n]\mathbf{e} \ \|_2^2\right] \\
&= \arg\max_{\mathbf{e}\in\mathcal{S}} \ \bar{\mathbf{d}}^{\mathrm{T}}[n-1]\mathbf{e} - D \ .
\end{aligned}$$

This choice of $\mathbf{e}'[n]$ differs from $\mathbf{e}[n]$ only in the bounded additional term. Qualitatively, the minimum 2-norm scheduling rule uses the same quantity to determine the schedule, the inner product between the queue length vector and the expected schedule (after multiplication with R and $M[n]$. The stability of this 2-norm scheduling rule follows from Theorem 4.8, and the choice of schedule $\mathbf{e}'[n]$ will be the same as $\mathbf{e}[n]$ except for a small portion of possible queue length vectors.

This model is an interesting generalization of the traditional queueing network model because it allows dependencies to be introduced between servers. In fact, we can also model the input-queued switch and reentrant queueing networks in this context. This brings up an interesting question: what classes of scheduling rules guarantee stability (bounded queue length) for the general constrained queueing model?

[2]The same model works for multiple classes; we just have many more queues, and constraints between the queues so that a single server only services one class at a time. It is easily seen that minimizing the squared distance of this augmented model simply entails looking at the maximum length for any class queue at a node. If we choose a node, we will always subtract from the longest queue to minimize the squared norm.

The scheduling rule that corresponds with the minimum 2-norm quantization rule guarantees stability for all $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$, by Theorem 4.8. This scheduling rule also reduces to a non-idling policy when the queueing network is unconstrained. This idea has also been presented in [65], where it is shown that a rule related to the 2-norm quantization rule guarantees stability for the same set of $\bar{\mathbf{r}}$. Our previous results extend this result (in a discrete-time context) to a larger set of geometric scheduling problems instead of just queueing network scheduling problems.

Constrained queueing networks have not been studied in great detail. Through our general dynamic model they can be seen as a special case (at least in discrete-time) of models with scheduling constellations that are an arbitrary collection of points.

3.5.4 Fair Queueing, and Credit or Potential Based Algorithms

Another class of scheduling algorithms, which we will call lag-based algorithms, have been used for maintaining a fair allocation of a resource among a number of competing users. The basic problem is that a resource with a number of possible configurations must be shared among a number of users according to *fairness coefficients* $\{\phi_i\}$ that are assigned to the users. The difference between this and the previous model is that a subset of the users may be idle during each round. The allocation that each user gets when the user is actively using the resource must be proportional to the ratio between the user's fairness coefficient and the sum of the coefficients of all currently competing users. This is not possible in general for a resource having a limited number of configurations, so we must try to match the desired allocations on average, over a long period of time. The PGPS algorithm in [57], the credit-based scheduling algorithm in [30], and the more general class of RPS algorithms in [63] can all be modeled using a modification of our dynamic model for situations in which the job processing times are constant. The p-fair algorithm in [3] is also an example of a lag-based algorithm; however, it lacks the arrival dynamics of the more general queueing model used in the other works.

The algorithm framework we look at here, which includes the models used in [57, 30, 63], extends our basic dynamic model. We include an additional dynamic equation that is used to model the actual queue length evolution. Instead of identifying the lag with the queue length, or buffered packets, we use the lag as a measure of credit that accumulates independently from the arriving packets themselves. In this way, we introduce a mechanism for determining priority between competing users.

The most important point to be understood is that the fairness coefficients do not govern the long-term bandwidth allocations, but the delay characteristics of the system, except when the users are submitting packets at a total rate larger than the maximum rate that the resource can handle. This is because as long as the total average arrival rate of the packets for all users is feasible (the system is stabilizable for that arrival rate) and the scheduling algorithm is non-idling, the system will be able to handle the overall arrival rate. The fairness coefficients determine the *relative* bandwidth that the users receive when competing with each other. We call a user *backlogged* if the user has packets waiting to be transmitted. If a backlogged user has a large fairness coefficient, the user will send the backlogged packets more quickly because he will be able to use the resource more often than users with small coefficients. This will allow this user's packets to travel through the resource with less delay. The method of initializing credit when a user becomes backlogged can also affect the delay or latency properties. If we give a large amount of initial credit the user can burst his first few packets through, and the resource appears more responsive to the user.

We restrict our scheduling problem to that of scheduling a number of users on a resource in a sequence of equally-sized rounds. In other words, all jobs/packets are the same size. There are two

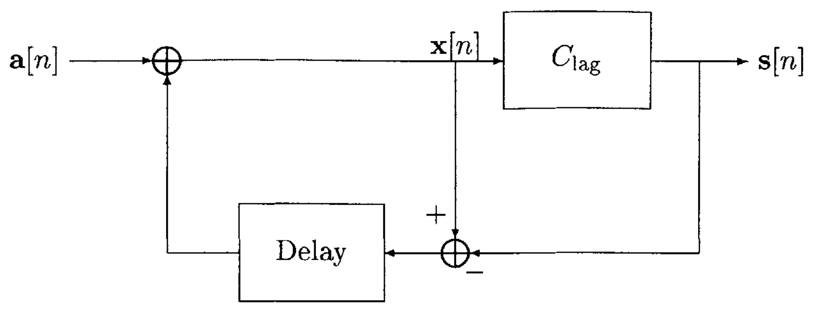


Figure 3-3: Block diagram of the state evolution equation.

equations that describe the dynamics of the system, as in [30]. The lag evolution equation is

$$\ell[n+1] = \ell[n] + \mathbf{r}[n+1] - \mathbf{s}[n] \ .$$

We call $\ell_i[n]$ the *lag* of user i. The input $\mathbf{r}[n]$ measures the rate at which the users are currently accumulating lag (or credit), and $\mathbf{s}[n]$ is the schedule chosen at round n. The state evolution equation captures the dynamics of the queue lengths themselves:

$$\mathbf{x}[n+1] = \mathbf{x}[n] + \mathbf{a}[n+1] - \mathbf{s}[n] \ .$$

The queue lengths are $\mathbf{x}[n]$ and the arrivals are denoted $\mathbf{a}[n]$. At each round, the schedule $\mathbf{s}[n]$ is chosen such that

$$\mathbf{s}[n] = \arg\min_{s \in \mathcal{S}} d(\ell[n], s) \ .$$

We also look only at the schedules that do not schedule more jobs than are currently in the queue, i.e. $s_i[n] \leq \ell_i[n]$. Figure 3-3 shows a block diagram of this system. The major difference between this and the earlier diagram, Figure 3-1, is the complexity of the scheduler C_{lag}. We are now dealing with a scheduler that has memory. A block diagram of C_{lag} (Figure 3-4) itself looks very similar to the previous diagram, but with $\mathbf{x}[n]$ affecting $\mathbf{r}[n]$ and $\ell[n]$ through the set of backlogged users. If $x_i[n] = 0$, user i is not backlogged and no credit accumulates during the period user i is not backlogged; user i is ignored when making scheduling decisions. For backlogged users, the credit accumulates at a constant rate equal to the ratio between the fairness coefficient ϕ_i and ϕ, the sum of the fairness coefficients of all backlogged users.

Backlogged Users

Let us assume for now that the users are constantly backlogged, or that there are always packets available to be scheduled ($x_i[n]$ is always greater than 0 for all users i). In this case, we can ignore the state evolution equation because it only affects the scheduling decisions when a state variable is equal to 0. In this case $r_i[n] = \phi_i$ and we can write

$$\ell_i[n] = \ell_i[n-1] + \phi_i - s_i[n-1] \ .$$

We will assume that $\sum_i \phi_i = 1$, or that the processing capacity of the resource is equal to 1. If the resource is a single queueing server, or communication link, $\mathbf{s}[n]$ might be chosen by setting $s_i[n] = 1$ for the user with the largest value of $\ell_i[n]$. As this is repeated, it is clear that on average each user is allocated the resource a fraction ϕ_i of the time. However, there is considerable latitude

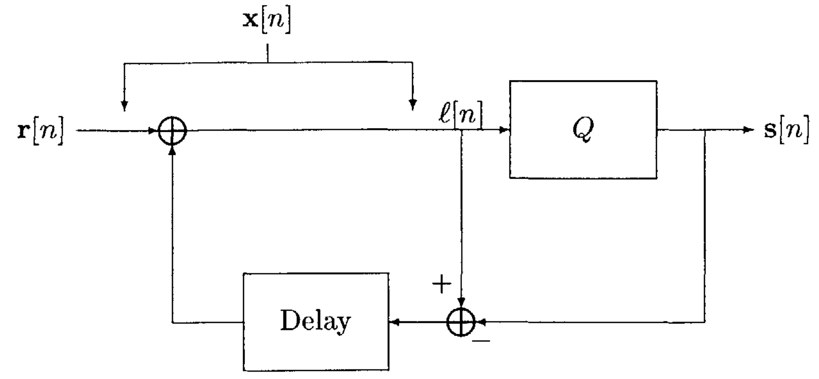


Figure 3-4: Diagram of the lag evolution equation block, C_{lag}.

in choosing $\mathbf{s}[n]$. As we saw before, we can choose $d(\cdot)$ to be one of a number of choices. The RPS model in [63] (which includes PGPS) uses a scheduling method that is equivalent to dividing $\ell_i[n]$ by ϕ_i for each i, then finding the largest element in the resulting vector and setting the corresponding element in $\mathbf{s}[n]$ to 1. This rule can also be written with $d(\cdot,\cdot)$ as a weighted function

$$d(\boldsymbol{\ell},\mathbf{s}) = \max_i \frac{\ell_i - s_i}{\phi_i} .$$

The credit-based scheduler in [30] is also similar; the algorithm uses a greedy maximal matching as described above, which is consistent with

$$d(\boldsymbol{\ell},\mathbf{s}) = \| \boldsymbol{\ell} - \mathbf{s} \|_\infty .$$

Again, for the input-queued switch, as long as all of the queues are backlogged, the bandwidth of the switch is allocated fairly among the users, assuming the credit stays bounded. Bounded credit is guaranteed if we use a maximum weighted matching, and empirically observed if we use a greedy maximal matching for all admissible arrival rates.

Queueing Dynamics

We have shown that our extended dynamic model represents the dynamics of the PGPS and credit-based input-queued switching algorithms when all queues are backlogged. What happens when a queue goes empty, or an empty queue receives an arrival? The question of how to allocate credit to idle queues and to newly backlogged queues in the link scheduling problem has been called a philosophical one ([59]), and has been approached in many different ways. One of the earliest and simplest lag-based algorithms is the VirtualClock algorithm [70]. In this basic mechanism, each user simply accumulates lag (credit) whether or not the user has packets in queue. If a user is idle for a while, then sends a burst of traffic, the user will have accumulated a large amount of credit while others have used theirs to transmit, and the user may monopolize the link for a period of time until its extra credit has been used. Although this may seem fair on a large scale — users are guaranteed a certain portion of the resource, no matter what — it is not generally good behavior because bursty traffic at the input remains bursty at the output, and fairness for each user is only guaranteed over large time windows. In fact, a user that does not send packets at a rate equal to its fairness coefficient will eventually accumulate an insurmountable surplus of credit that will allow it to monopolize the resource whenever it has packets to send.

A solution to this problem in [70] is to simply not accumulate credit when the queue is empty (realized by resetting the credit to zero whenever an empty queue receives a packet). This method may seem to violate the fairness guarantees, but we can see that this does not happen on a large scale, and by construction, it disallows the monopolization problem and all users share according to their fairness coefficients when all have packets. If this method *were* to violate a user's fairness coefficient by not sending traffic fast enough, then we would expect the user's queue to eventually become backlogged. As long as the queue is backlogged, however, the user is guaranteed a minimum rate appropriate to its fairness coefficient, so this solution does not affect the large-scale fairness guarantees, and provides better smaller-scale guarantees by eliminating the ability of a user to monopolize the link.

Another simple method of dealing with this problem is used in [30], where credit is allowed to build up while a user is idle up to a certain bound. An artificial limit is placed on the lag when a user is idle. This controls the bursty users through smoothing the traffic somewhat by only allowing a user to save up a limited amount of credit with which to send bursts through. After this amount is used up, the user will have to wait for credit to accumulate at its fixed rate. This method still guarantees long-term fairness. By allowing idle users to build up some credit, it allows newly backlogged users to begin transmitting a few packets quickly before smoothing out the traffic.

The PGPS algorithm in [57] uses essentially the same credit accumulation method as in [30], with a few differences, if we only consider the case when all packets are the same size. We define the lag for a user as the product of the user's fairness coefficient and the difference between the current time and the time at which the next packet would be finished if the communication link were able to time share with an infinitely fine resolution (vanishingly short rounds). As long as a queue is not empty and does not send packets, the lag builds up at a constant rate. This rate is proportional to the actual fairness coefficient ϕ_i for the backlogged user. However, if only a subset of the users have nonempty queues, the rates are scaled by the same factor so that the sum of the rates is equal to 1. If the set of backlogged users is B, then the actual rate $r_i[n]$ is

$$r_i[n] = \frac{\phi_i}{\sum_{j \in B} \phi_j} .$$

We set $r_i[n] = 0$ for all $i \notin B$. We set $\ell_i[n] = -1$ for all $i \notin B$. During each round, we choose $\mathbf{s}[n]$ based only on the queues that are backlogged during the round. The choice of $\mathbf{s}[n]$ is also made in a different manner. Instead of simply picking the largest $\ell_i[n]$, we first scale $\ell_i[n]$ by $1/r_i[n]$, then pick the largest of the resulting numbers; in other words, we use a metric

$$d(\ell, \mathbf{s}) = \max_i \frac{\ell_i - s_i}{r_i[n]} .$$

For a non-backlogged user i, $r_i[n] = 0$ and $\ell_i = -1$, so $\ell_i / r_i[n] = \infty$ and we will never choose to give the resource to i.

To see that this is equivalent to the PGPS algorithm, consider the two inputs to the lag evolution equation: $\mathbf{r}[n]$ and $\mathbf{s}[n]$. The input $\mathbf{r}[n]$ is an analog of the rate of change of virtual time in the PGPS model, simply scaled by ϕ. Because virtual time is the same for all users, when we scale it by $1/\phi$ in order to allocate the link, it does not affect the choice — the same user will have the maximum lag if we multiply the same number with every user's lag. Also, $\mathbf{s}[n]$ is the negative of the timestamp that is put on each packet, or the time it would finish processing in a fluid GPS server. If all packets *are the same size,* this timestamp simply increments by $1/\phi$ each time a new packet arrives. This corresponds with decrementing the lag by 1 after every packet that is serviced (the next packet's timestamp is simply the old stamp plus $1/\phi$). So, if we scale the lag by $1/\phi$, it is equal to virtual time minus the next packet's timestamp. Thus, choosing the user with the maximum scaled lag is equivalent to choosing the user with the minimum timestamp, as in the PGPS algorithm.

We can picture the PGPS algorithm simply as follows. During each period in which the same set B of users have packets waiting, the queue services them according to their fairness coefficients as if the idle users did not exist. When a user leaves this set B, its remaining credit, or lag, is redistributed by normalizing the lags of the remaining users. When a user enters B, it begins with lag equal to -1.

This specific choice of the initial lag points towards the more general question of what happens when we vary the initial conditions. That is what the RPS model [63] essentially entails. At the beginning of each busy period for each user, a system potential function is consulted and used as the initial value for the user's potential (an affine transformation of our lag).

We have been able to extend our original model to include a more dynamic model in which the scheduler adapts to the set of currently active, or backlogged, users. The large-scale fairness guarantees are kept while allowing flexibility in choosing the actual scheduling rule. The scheduling rules described above, especially the class of rules in [63], let us tailor the short time-scale behavior of the allocation mechanism to have other desired properties (as described by the fairness coefficients in this example).

We can also extend this model to include arbitrary length packets as follows. If we consider round n as corresponding to t_n, the time of the n-th departure from the link, and if $p[n]$ is the length of the packet serviced during round n, then the lag evolution equations become:

$$\ell_i[n] \quad = \quad \ell_i[n-1] - p[n]s_i[n-1] + p[n-1]r_i[n] \ .$$

The rules for updating $\mathbf{r}[n]$ and $\ell[n]$ when a queue goes empty or becomes backlogged are now more complicated, but possible to write down. This equation still fits within our dynamic model; it now has stochastic elements. We discuss this in some more detail in the concluding chapter.

Chapter 4

Stability

Our focus in this chapter is on developing stability results that apply to the general dynamic model we have just presented. The model we use here is similar in flavor to ones in other work [52, 65], but is different enough that the stability results already published do not directly apply. Although the theory of stochastic stability for Markov chains that evolve over general (non-countable) state spaces has been looked at in detail [53], our specific model has not been analyzed to the best of our knowledge.

Our results fall into two categories: stability of deterministic models and stability of stochastic models. We adopt our notions of stability in the stochastic setting from the literature on stochastic stability [53, 41, 40, 46, 25]. As in [65], the analytical results concentrate mainly on determining the class of models which are stabilizable under different types of norms. Just like in [65], stabilizability will be shown constructively, by exhibiting a quantizing scheduling rule that stabilizes a given model. For our general model we will be able to show stabilizability (ultimate boundedness for deterministic models and various levels of stochastic stability for stochastic models) when the average input rate lies within the confines of the schedule constellation. The class of norms for which these guarantees hold is either the class of smooth norms for deterministic models, or the squared 2-norm for stochastic models. Our coverage of the stochastic stability of our mechanisms consists mainly of the application of existing results to our model. We also establish that input rates outside of this region result in instability for all scheduling rules for deterministic models.

For the deterministic models with a feasible input rate, we show that a sufficient condition for stability is that the scheduling norm be smooth; we also look at some examples of non-smooth norms which illustrate that all models are not always stabilizable under specific norms.

In Section 4.1 we have a definition of stability and stability results for the case when the scheduler is deterministic, showing that the scheduler is stabilizable for all average input rates within a feasible region using any quantizing scheduling algorithm based on a smooth norm. Then, we present example systems to show that under non-smooth norms, stability is not guaranteed in general, and that deterministic stability does not imply stochastic stability.

Once we know that a large class of mechanisms are stable under a variety of norms, we can use this to help us choose a norm that results in some desired lag behavior. We discuss a few examples in Section 4.2 that show how the choice of norm can influence the behavior of the lag.

We then investigate the general stochastic model in Section 4.3 and show how existing stochastic stability results can be applied to our model.

4.1 Deterministic Stability

We begin with the deterministic model because of the ease with which the notions of stability can be defined and visualized. A natural definition of stability for a deterministic dynamic system is that all state trajectories tend towards a bounded region

One of the main reasons we care about stabilizability, or that the lag stay bounded, is that it guarantees the time-average allocation to a particular user converges to the requested allocation for the user. Theorem 4.3 tells us that under any given smooth norm, the lag of a deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ stays bounded within a finite region, and thus the system allocates the resource fairly among the users. The rest of this section is devoted to proving this fact.

We start with defining deterministic stability. A model is deterministically stable under a particular scheduling rule if there exists some compact set such that the lag vector remains within the set after the first time it enters it.

Definition 4.1 (Stability of a Deterministic Model) *Given a deterministic dynamic scheduler model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ and a scheduling rule $Q : \mathbb{R}^N \to \mathcal{S}$, we say that the model is* deterministically stable *under rule Q if for all $\ell[0]$ there exists a round n_0 and a compact set $\mathcal{B} \subset \mathbb{R}^N$ such that $n \geq n_0$, $\ell[n] \in \mathcal{B}$.*

We can connect stability with fairness as follows.

Theorem 4.1 *A deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ along with a scheduling rule $Q(\cdot)$ is a bounded fair mechanism if and only if it is deterministically stable.*
Proof: This follows easily from Definitions 3.2 and 4.1. ∎

Of course, this means that a deterministically stable mechanism is also deterministically fair. However, this does not imply that a mechanism that is not deterministically stable is not fair. An example of a mechanism that is fair but not stable (although only demonstrated empirically) is the second-price infinite horizon repeated auction from Chapter 7. We also want to define what it means for a model to be stabilizable. A deterministic model is *stabilizable* if there exists a sequence of schedules $\mathbf{s}[0], \mathbf{s}[1], \mathbf{s}[2], \ldots$ such that the lag vector is bounded.

Definition 4.2 (Stabilizability of a Deterministic Model) *Given a deterministic dynamic scheduler model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ we say that the model is* deterministically stabilizable *if for any $\ell[0]$ there exists a sequence of schedules $\mathbf{s}[0], \mathbf{s}[1], \mathbf{s}[2], \ldots$, a round n_0 and a compact set $\mathcal{B} \subset \mathbb{R}^N$ such that for $n \geq n_0$, $\ell[n] \in \mathcal{B}$.*

Another notation that we will use is $\bar{B} \cdot \mathcal{S}$. This is defined to be the set of all vectors $\bar{B}s$, for $s \in \mathcal{S}$. An object that is very important in our stability discussions is the convex hull of the constellation $\mathrm{CH}(\bar{B} \cdot \mathcal{S})$, or the set of all possible convex combinations of points in $\bar{B} \cdot \mathcal{S}$ (see Appendix A for the definition of the convex hull and its interior). A deterministic model is not stabilizable if $\bar{\mathbf{r}}$ is not inside of the convex hull of $\bar{B} \cdot \mathcal{S}$. This follows from the fact that there is no way to combine the elements of $\mathcal{S}$ over time to obtain an average value of $\bar{\mathbf{r}}$ and so the difference between the cumulative allocations and the reservations grows unbounded. So, from now on, unless specifically stated otherwise, we will generally make the assumption that $\bar{\mathbf{r}}$ is inside the convex hull of $\bar{B} \cdot \mathcal{S}$.

Theorem 4.2 (Stabilizability of a Deterministic Model) *A deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ is not stabilizable if $\bar{\mathbf{r}} \notin \mathrm{ch}(\bar{B} \cdot \mathcal{S})$.*
Proof: When $\bar{\mathbf{r}} \notin \mathrm{CH}(\bar{B} \cdot \mathcal{S})$ there is at least one parameter $\alpha(s)$ such that $\alpha(s) < 0$ or $\alpha(s) > 1$

for any set of parameters $\alpha(s)$ such that $\sum_{s\in\bar{B}\cdot\mathcal{S}}\alpha(s)s = \mathbf{r}$. But, for any scheduling sequence $\mathbf{s}[n]$, we can write

$$\begin{aligned}
\ell[n+1] &= \ell[0] + \sum_{m=0}^{n}(\bar{\mathbf{r}} - \bar{B}\mathbf{s}[m]) \\
&= \ell[0] + \sum_{m=0}^{n}\Big(\sum_{s\in\bar{B}\cdot\mathcal{S}}\alpha(s)s - \bar{B}\mathbf{s}[m]\Big) \\
&= \ell[0] + \sum_{s\in\bar{B}\cdot\mathcal{S}}(n+1)\cdot(\alpha(s)s - \tilde{\alpha}_n(s)s) \ ,
\end{aligned}$$

where we define $\tilde{\alpha}_n(s)$ to be the fraction of rounds up to round n that $\bar{B}\mathbf{s}[n] = s$, or

$$\tilde{\alpha}_n(s) = \frac{1}{n+1}\cdot\sum_{j:\mathbf{s}[j]=s,0\leq j\leq n} 1 \ .$$

It is obvious that $0 \leq \tilde{\alpha}_n(s) \leq 1$ for any n. For later contradiction, assume that there exists a finite n_0 such that $\|\ \ell[n]\ \|_2 \leq b$ for $n \geq n_0$. Then, we have

$$\begin{aligned}
\frac{\ell[n+1]-\ell[0]}{n+1} = \mathbf{e}_n &= \sum_{s\in\bar{B}\cdot\mathcal{S}}(\alpha(s)s - \tilde{\alpha}_n(s)s) \\
\sum_{s\in\bar{B}\cdot\mathcal{S}}\alpha(s)s - \mathbf{e}_n &= \sum_{s\in\bar{B}\cdot\mathcal{S}}\tilde{\alpha}_n(s)s \\
\bar{\mathbf{r}} - \mathbf{e}_n &= \sum_{s\in\bar{B}\cdot\mathcal{S}}\tilde{\alpha}_n(s)s \ .
\end{aligned}$$

Because $\bar{\mathbf{r}} \notin \mathrm{CH}(\bar{B}\cdot\mathcal{S})$ and $\mathrm{CH}(\bar{B}\cdot\mathcal{S})$ is closed, we know that there exists a neighborhood of radius $\delta_{\bar{\mathbf{r}}} > 0$ around $\bar{\mathbf{r}}$ which is also not in $\mathrm{CH}(\bar{B}\cdot\mathcal{S})$. If we assume that $\|\ \ell[n]\ \|_2 \leq b$ for all $n \geq n_0$, then we can drive $\|\ \mathbf{e}_n\ \|_2$ arbitrarily small as n increases. Because $\delta_{\bar{\mathbf{r}}} > 0$ we know that there is a finite value n_1 for which $\|\ \mathbf{e}_n\ \|_2 \leq \delta_{\bar{\mathbf{r}}}$ and thus $\bar{\mathbf{r}} - \mathbf{e}_n \notin \mathrm{CH}(\bar{B}\cdot\mathcal{S})$ for all $n \geq n_1$. Thus, the system is not stabilizable if $\bar{\mathbf{r}} \notin \mathrm{CH}(\bar{B}\cdot\mathcal{S})$.

Finally, we need to show that the model is not stabilizable if $\bar{\mathbf{r}}$ is on the surface of the convex hull, i.e. $\bar{\mathbf{r}} \in \mathrm{CH}(\bar{B}\cdot\mathcal{S}) - \mathrm{ch}(\bar{B}\cdot\mathcal{S})$. To do this we realize that when $\bar{\mathbf{r}}$ is on the surface of the convex hull there is some vector z such that

$$z^{\mathrm{T}}\bar{\mathbf{r}} = \max_{\mathbf{r}\in\mathrm{CH}(\bar{B}\cdot\mathcal{S})} z^{\mathrm{T}}\mathbf{r} \ .$$

If we pick a starting position $\ell[0]$ such that $z^{\mathrm{T}}\ell[0] = \alpha > z^{\mathrm{T}}\bar{\mathbf{r}}$ then we know that $z^{\mathrm{T}}\ell[n] \geq \alpha$ for all $n > 0$. This follows by induction. If it is true for $\ell[n-1]$ then we have that

$$\begin{aligned}
z^{\mathrm{T}}\ell[n] &= z^{\mathrm{T}}(\ell[n-1] + \bar{\mathbf{r}} - \mathbf{s}[n-1]) = z^{\mathrm{T}}\ell[n-1] + z^{\mathrm{T}}\bar{\mathbf{r}} - z^{\mathrm{T}}\mathbf{s}[n-1]) \\
&\geq z^{\mathrm{T}}\ell[n-1] \geq \alpha \ .
\end{aligned}$$

So, if we were to assume the existence of a compact set $\mathcal{B}$ which $\ell[n]$ enters and stays within for all time after some point, we could just choose $\ell[0]$ such that $z^{\mathrm{T}}\ell[0] > \max_{b\in\mathcal{B}} z^{\mathrm{T}}b$, and we know that $\ell[n]$ will never enter $\mathcal{B}$, contradicting our assumption. ■

The converse of this theorem, that a deterministic model is stabilizable if $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B}\cdot\mathcal{S})$, is demonstrated in the next section by exhibiting a class of scheduling rules that stabilize such a deterministic model.

4.1.1 Drift Condition

A common technique that we will use to prove deterministic stability makes use of Lyapunov functions. One way to establish deterministic stability for our model is to exhibit a positive-valued function $V(\ell)$ called a Lyapunov function that satisfies the drift condition given below. We will call a function $V(\ell)$ *level-bounded* if the sub-level set $\{\ell \mid V(\ell) \leq a\}$ is bounded for every a. The Lyapunov functions we talk about below are all norms, which are level-bounded by definition. From [53], if we can show that $V(\ell[n+1]) - V(\ell[n]) \leq -\epsilon$ outside of some compact set of values of $\ell[n]$ and if $V(\ell[n]) < \infty$ for all $\ell[n]$ within the set, then it is clear that $V(\ell[n])$ must stay bounded below some value and thus $\ell[n]$ is bounded below some finite constant for all n. We call this the *deterministic drift condition.*

Definition 4.3 (Deterministic Drift Condition) *The sequence $\ell[n]$ defined by a deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ and a scheduling rule $Q(\cdot)$ is said to satisfy the* deterministic drift condition *if there exists a positive- and finite-valued level-bounded function $V(\cdot)$, a compact* drift set *$\mathcal{C} \subset \mathbb{R}^N$, and a constant $M < \infty$ such that*

$$V(\ell[n+1]) - V(\ell[n]) \leq -\epsilon \tag{4.1}$$

for $\epsilon > 0$, all $\ell[n] \notin \mathcal{C}$ and

$$V(\ell[n+1]) - V(\ell[n]) \leq M < \infty \tag{4.2}$$

for $\ell[n] \in \mathcal{C}$.

By finite-valued, we mean that $V(\ell)$ is finite when ℓ is finite. The second condition (4.2) will always be true for the Lyapunov functions we choose, and so we will generally concentrate only on showing condition (4.1). By Theorem 11.2.1 of [53], this drift condition ensures that the state $\ell[n]$ in our deterministic model remains within a bounded set.

4.1.2 Smooth Norms

Having defined stability for deterministic models and given a sufficient condition for stability we proceed to look at specific scheduling rules. We show that any quantizing scheduling rule based on a smooth norm $\|\cdot\|$ guarantees stability for a deterministic model such that $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$.

In particular, we will investigate the *stability region* for a given constellation $\bar{B} \cdot \mathcal{S}$ and smooth norm $\|\cdot\|$. The stability region is simply the set of vectors $\bar{\mathbf{r}}$ such that the model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ is deterministically stable under the quantizing scheduling rule that uses the metric $d(x, y) = \| x - y \|$. We show now that the stability region for a deterministic model using an arbitrary smooth norm is equal to $\text{ch}(\bar{B} \cdot \mathcal{S})$, just as with the 2-norm.

In order to show stability for a particular model and scheduling rule combination, we will show that the deterministic drift condition is satisfied for any deterministic model when $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$. We define the Lyapunov function $V(\cdot)$ as

$$V(\ell) = \| \ell \| .$$

We will also define $\ell^+[n] = \ell[n] + \bar{\mathbf{r}}$. This notation makes the following work simpler. With this definition, we also have

$$\begin{aligned} V(\ell[n]) &= \| \ell^+[n] - \bar{\mathbf{r}} \| \\ V(\ell[n+1]) &= \| \ell^+[n+1] - \bar{\mathbf{r}} \| = \| \ell^+[n] - \bar{B}\mathbf{s}[n] \| . \end{aligned}$$

By the definition of the quantizing scheduling rule, $\bar{B}\mathbf{s}[n]$ is the closest point in $\bar{B} \cdot \mathcal{S}$ to $\ell^+[n]$. From the definition of $V(\cdot)$, we only have to show that $\ell^+[n]$ is closer to $\bar{B}\mathbf{s}[n]$ than to $\bar{\mathbf{r}}$ for all $\ell[n]$ outside some finite region $\mathcal{C}$ in order to establish the drift condition.

To prove this stability result we show that for any $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$ there is a set $\mathcal{C}$ such that if $\ell[n] \notin \mathcal{C}$ then

$$|| \ \ell^+[n] - \bar{\mathbf{r}} \ || - || \ \ell^+[n] - \bar{B}\mathbf{s}[n] \ || \leq -\epsilon \tag{4.3}$$

and we have satisfied the deterministic drift condition. After this we will introduce the idea of a *stability set*, which is the collection of possible values of $\bar{\mathbf{r}}$ for which (4.3) holds for all $\ell[n]$ that are at least some distance C (according to $|| \cdot ||$) away from the constellation $\mathcal{S}$.[1] We will show that the stability set for an arbitrary smooth norm and constellation grows towards $\text{ch}(\bar{B} \cdot \mathcal{S})$ as C grows to ∞. However, we can also define a stability set for a corner norm; when we do this we see that the same property does not hold. The stability set may reach a maximum that is smaller than $\text{ch}(\bar{B} \cdot \mathcal{S})$.

We begin by stating the theorem that establishes stability under smooth norms. The proof works by constructing a compact set outside of which (4.3) holds. We use $\mathcal{D}(u, C) = \{x \mid || \ x - u \ || \leq C\}$ to denote the set of vectors that are within distance C of a point u; then the set $\mathcal{D}(0, C)$ is a sub-level set of the norm $|| \cdot ||$.

Theorem 4.3 (Deterministic Stability of Smooth Norms) *A deterministic dynamic scheduler model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ is deterministically stable under scheduling rule $Q(\cdot)$ for all $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$ if $Q(\cdot)$ is consistent with a smooth norm $|| \cdot ||$.*
Proof: See Appendix, Section B.3. ■

The drift set $\mathcal{C}$ for which the drift condition holds is equal to $\mathcal{D}(0, C)$ for some value C. This means that for all $\ell[n] \notin \mathcal{D}(0, C)$ the drift condition is satisfied. This result tells us that for any deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ we are free to choose a scheduling rule consistent with any smooth norm without worrying about whether the system will be stable. We examine how the choice of norm might be important and how it can influence the performance of a system when we discuss Σ-Δ encoders in Section 4.2.

Theorem 4.3 can tell us whether or not a deterministic model is stable, but it does not give us any way of constructing a minimal set of vectors ℓ^+ outside of which the negative drift condition (4.1) holds. There is an easy way to characterize this minimal set geometrically as a superset of the Voronoi region for $\bar{\mathbf{r}}$ in an augmented constellation $\mathcal{S}_{\bar{\mathbf{r}}} = \mathcal{S} \cup \{\bar{\mathbf{r}}\}$ given the particular norm. The Voronoi region for a point x within a constellation $\mathcal{X}$ is defined as the set of points that are closer to x than any other point in $\mathcal{X}$. If we use an arbitrary norm $|| \cdot ||$ to make the distance comparisons, it is clear that any point ℓ some distance outside the Voronoi region corresponding to $\bar{\mathbf{r}}$ is closer to a point in $\mathcal{S}$ than to $\bar{\mathbf{r}}$. The minimal set then would correspond to the set of points ℓ such that $\min_{s \in \mathcal{S}} || \ \ell^+ - s \ || - || \ \ell^+ - \bar{\mathbf{r}} \ || \leq -\epsilon < 0$. We can think of this minimal set as a superset of the Voronoi region for $\bar{\mathbf{r}}$ extended by an amount proportional to ϵ. This corresponds to satisfying the drift condition (4.1).

A sample trajectory for a simple deterministic model using the 2-norm with $\bar{B} = I$, $\bar{\mathbf{r}} = (.7, .3)$ and $\mathcal{S}$ as a hexagon is shown in Figure 4-1. It is clear that the trajectory moves towards the Voronoi region for $\bar{\mathbf{r}}$, then remains close by as time goes forward. In general, as is exemplified by this figure, the Voronoi region is much smaller than the drift set determined in the proof of Theorem 4.3.

[1]Being a distance C from a constellation means that the closest point in the constellation, $\mathbf{s}[n]$ in this case, is at distance C.

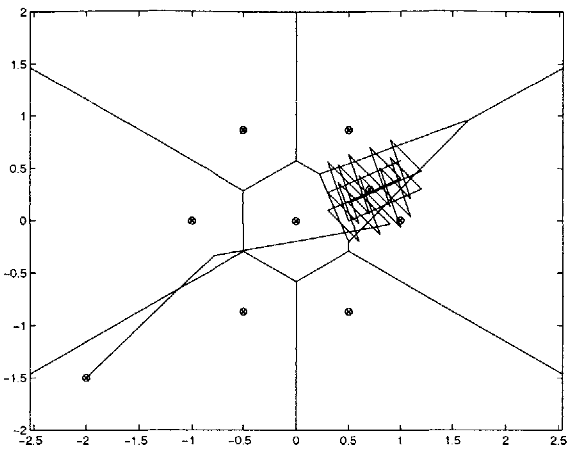


Figure 4-1: Voronoi regions for $\mathcal{S}_{\bar{\mathbf{r}}}$ and a sample trajectory when $\bar{\mathbf{r}} = (.7, .3)$.

The set of ℓ^+ outside of which the drift condition holds in Theorem 4.3 can be thought of as the set $\mathcal{D}(\bar{\mathbf{r}}, C)$ that completely contains this minimal set (Voronoi region) for the smallest value of C. Another way of interpreting Theorem 4.3 is that it tells us the Voronoi region with respect to an arbitrary smooth norm for any constellation point in the interior of the convex hull of the constellation is bounded. We will see later that it is this property that can be violated when we use non-smooth norms, which makes our drift-based proof inapplicable.

Theorem 4.3 also leads to the following result.

Theorem 4.4 (Stabilizability of Deterministic Models) *A deterministic model* $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ *is stabilizable if and only if* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$.
Proof: We have already shown the reverse direction with Theorem 4.2, and we now know that the forward direction is also true because any scheduling rule induced by a smooth norm stabilizes such a deterministic model. ∎

We can extend this theorem to allow time-varying $\mathbf{r}[n]$ as long as $\mathbf{r}[n]$ stays within a subset of $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$.

4.1.3 Time-Varying $\mathbf{r}[n]$

It follows from the previous proof that if we extend our deterministic model to allow $\mathbf{r}[n]$ to vary (deterministically or stochastically) over time such that $\mathbf{r}[n] \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ for all n while keeping $B[n] = \bar{B}$ constant, we can still guarantee the stability of all deterministic models under all smooth norms, with the condition that $\mathbf{r}[n]$ lie in a closed subset $\mathbf{R}$ of $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$. We use the same scheduling rule as before, choosing the schedule $\mathbf{s}[n]$ that minimizes $\| \ell[n] + \bar{\mathbf{r}} - \bar{B}\mathbf{s}[n] \|$. However, we can choose $\bar{\mathbf{r}}$ now to be any vector in $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$. If $\bar{\mathbf{r}}$ is not equal to the average value of $\mathbf{r}[n]$, the performance may suffer as the lag will drift somewhat outside the constellation, so a good estimate of the average value is useful. As long as $\mathbf{R}$ is closed then there is no subsequence $\tilde{\mathbf{r}}[m]$ of $\mathbf{r}[n]$ such that $\lim_m \tilde{\mathbf{r}}[m] \notin \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ and we can always use the technique of Theorem 4.3 to construct a compact drift set C for each possible value of $\mathbf{r}[n]$. This subsequence property basically means that $\mathbf{r}[n]$ must stay some finite distance away from the surface of the convex hull for all n, and can not approach the surface asymptotically.

Theorem 4.5 (Deterministic Stability with Time-Varying Input) *A deterministic dynamic scheduler model* $(\bar{B}, \mathbf{r}[n], \mathcal{S})$ *is deterministically stable under scheduling rule* $Q(\cdot)$ *for all* $\mathbf{r} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ *if* $Q(\cdot)$ *is consistent with a smooth norm* $\| \cdot \|$ *and if* $\mathbf{r}[n] \in \mathbf{R} \subset \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ *where* $\mathbf{R}$ *is closed.*
Proof: This follows from Theorem 4.3. We define $\mathbf{R}$ as the set of possible values of $\mathbf{r}[n]$. We use $V(\ell[n]) = \| \ell[n] \|$ and $\ell^+[n] = \ell[n] + \bar{\mathbf{r}}$ as before, but now we can use an arbitrary $\bar{\mathbf{r}} \in \mathbf{R}$. We need to guarantee that

$$V(\ell[n+1]) - V(\ell[n]) \leq -\epsilon \ .$$

We can rewrite this as

$$\begin{aligned} \| \ell^+[n+1] - \bar{\mathbf{r}} \| - \| \ell^+[n] - \bar{\mathbf{r}} \| &\leq -\epsilon \\ \| \ell^+[n] + \mathbf{r}[n+1] - \bar{B}\mathbf{s}[n] - \bar{\mathbf{r}} \| - \| \ell^+[n] + \mathbf{r}[n+1] - (\bar{\mathbf{r}} + \mathbf{r}[n+1]) \| &\leq -\epsilon \\ \| \tilde{\ell}[n] - \bar{B}\mathbf{s}[n] \| - \| \tilde{\ell}[n] - \mathbf{r}[n+1] \| &\leq -\epsilon \ . \end{aligned}$$

By the proof of Theorem 4.3, we know that this last statement is true for $\tilde{\ell}[n]$ outside of the set $\mathcal{D}(\mathbf{r}[n+1], C)$ for large enough C. We can easily translate this into a drift set $\mathcal{C} = \mathcal{D}(0, C)$ for $\ell[n]$. Finally, we define the overall drift set $\mathcal{C}'$ as the union of these corresponding sets for each value of $\mathbf{r} \in \mathbf{R}$. This union is simply equal to $\mathcal{D}(0, C)$ for the largest value of C over all possible values $\mathbf{r} \in \mathbf{R}$.

This means that at each round the drift condition (4.1) is satisfied. The condition that $\mathbf{R}$ be a closed subset of $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$ is required in the statement of the theorem because the proof of Theorem 4.3 depends on the distance between $\bar{\mathbf{r}}$ and $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$ being some finite positive value. If $\mathbf{R}$ were open, we could we let $\mathbf{r}[n]$ get arbitrarily close to the convex hull without ever reaching it and violate this property. Then, we may get behavior as described in the example below, where the bound on the lag is not independent of the initial condition. ■

An example of the unstable behavior when $\mathbf{R}$ is not closed is seen when $\mathbf{r}[n] = 1 - e^{-n}$, $B[n] = I$ and $\mathcal{S} = \{-1, 1\}$. For this scalar case, the distortion metric is not important, so we will just use the ∞-norm. As long as $\ell[n] > 0$, we will always choose $\mathbf{s}[n] = 1$. In this case, the lag evolution equation becomes:

$$\begin{aligned} \ell[n+1] &= \ell[n] + \mathbf{r}[n+1] - \mathbf{s}[n] = \ell[n] + 1 - e^{-(n+1)} - 1 \\ &= \ell[n] - e^{-(n+1)} \\ &= \ell[0] - \frac{1}{e} \cdot \frac{1 - e^{-(n+1)}}{1 - e^{-1}} \ . \end{aligned}$$

So, $\lim_{n \to \infty} \ell[n] = \ell[0] - 1/(e-1)$. As long as $\ell[0] > 1/(e-1)$ we will have $\ell[n] > \ell[0] - 1/(e-1)$ for all n. This means that there is no bound D such that $|\ell[n]| < D$ for large n and all initial conditions because for any D we can pick $\ell[0] > D + 1/(e-1)$ and get a contradiction.

This extension to time-varying $\mathbf{r}[n]$ is interesting because it captures a couple of important scenarios. First, a Σ-Δ encoder whose quantization constellation is chosen such that the input signal always lies within the convex hull of the constellation fits this model, and as such we know that the encoder is stable using any smooth norm. Second, we can also capture a subclass of our general stochastic model this way if we restrict the input $\mathbf{r}[n]$ to be a random process that only takes values within the interior of the convex hull as assumed in the theorem. Thus, we have a form of deterministic stability for a subclass of the stochastic model.

4.1.4 Stability Sets

We now move to an alternative view of the stability question for smooth norms. Drawing from the proof of Theorem 4.3 we can define the *stability set* of vectors $\bar{\mathbf{r}}$ such that we are guaranteed stability for a given smooth norm and specific drift set $\mathcal{C}$. Theorem 4.5 tells us that the stability set grows larger and eventually includes all closed subsets of $\mathrm{ch}(\bar{B}\cdot\mathcal{S})$ as we increase the size of $\mathcal{C}$. By our definition of the stability set, we can construct it for any norm, smooth or otherwise. We can use the stability set for a corner norm to illustrate why we can not guarantee stability for corner norms with the techniques used so far. When we use a corner norm, the stability set may not grow to include all closed subsets of $\mathrm{ch}(\bar{B}\cdot\mathcal{S})$, and may leave out particular regions of $\mathrm{ch}(\bar{B}\cdot\mathcal{S})$ no matter how large we grow $\mathcal{C}$.

Let $\mathcal{L}^{+}(C)$ denote the set of vectors such that the closest constellation point is at most distance C away, or all $\ell^{+}[n]$ such that $\|\ \ell^{+}[n]-\bar{B}\mathbf{s}[n]\ \| \leq C$, for some positive constant C. By this definition $\mathcal{L}(C)$ is the set of points that are within distance C of some point in the constellation $\mathcal{S}$, or

$$\mathcal{L}^{+}(C)=\bigcup_{s\in\mathcal{S}}\mathcal{D}(s,C)\ .$$

We will eventually use $\mathcal{L}^{+}(C)$ as an upper bound on the drift set $\mathcal{C}$ in order to analyze the stability set. We will denote the surface of $\mathcal{L}^{+}(C)$, or the set of vectors $\ell^{+}[n]$ such that $\|\ \ell^{+}[n]-\bar{B}\mathbf{s}[n]\ \|=C$ as $\mathcal{L}^{o}(C)$. In order to use Theorems 4.3 and 4.5, we need to know that for any $\mathcal{C}=\mathcal{D}(\bar{\mathbf{r}},C)$, there is a C^{m} such that for all $D>C^{m}$, $\mathcal{C}\subseteq\mathcal{L}^{+}(D)$.

Lemma 4.1 *There exists some positive C^{m} such that for a given C, all $D>C^{m}$ and $\bar{\mathbf{r}}\in\mathrm{ch}(\bar{B}\cdot\mathcal{S})$, $\mathcal{D}(\bar{\mathbf{r}},C)\subset\mathcal{L}^{+}(D)$.*
Proof: By the definition of $\mathcal{L}^{+}(\cdot)$, we have $\mathcal{L}^{+}(C^{m})\subset\mathcal{L}^{+}(D)$ if $C^{m}<D$. So, we only have to show that there exists a C^{m} such that $\mathcal{D}(\bar{\mathbf{r}},C)\subseteq\mathcal{L}^{+}(C^{m})$.

This last statement follows from the triangle inequality. For any point $p\in\mathcal{D}(\bar{\mathbf{r}},C)$, we would like to show that there exists a C^{m} such that $p\in\mathcal{L}^{+}(C^{m})$. We know that $\mathcal{D}(s,C^{m})\subseteq\mathcal{L}^{+}(C^{m})$ for any $s\in\mathcal{S}$ by definition, so we will just show that $p\in\mathcal{D}(s,C^{m})$ for some C^{m}.

$$\|\ p-s\ \|\leq\|\ p-\bar{\mathbf{r}}\ \|+\|\ \bar{\mathbf{r}}-s\ \|\leq C+\|\ \bar{\mathbf{r}}-s\ \|\ .$$

If we define $C^{m}=C+\max_{s\in\mathcal{S}}\|\ \bar{\mathbf{r}}-s\ \|$ then we know that $p\in\mathcal{D}(s,C^{m})\subseteq\mathcal{L}^{+}(C^{m})$ whenever $p\in\mathcal{D}(\bar{\mathbf{r}},C)$. ∎

Because the form of the drift sets determined in Theorems 4.3 and 4.5 is $\mathcal{C}=\mathcal{D}(0,C)$ and thus the set of vectors ℓ^{+} outside of which the drift condition is satisfied is $\mathcal{D}(\bar{\mathbf{r}},C)$, this lemma tells us that we can always choose C^{m} large enough to include this set in some $\mathcal{L}^{+}(C^{m})$.

Given a set $\mathcal{L}^{+}(C)$ we define the *stability set* $\mathcal{R}(C)$ as containing all vectors $\bar{\mathbf{r}}$ such that $\bar{\mathbf{r}}\in\mathrm{ch}(\bar{B}\cdot\mathcal{S})$ and

$$\|\ \ell^{+}[n]-\bar{\mathbf{r}}\ \|\geq\|\ \ell^{+}[n]-\bar{B}\mathbf{s}[n]\ \|-\epsilon \tag{4.4}$$

for some $\epsilon>0$ and all $\ell^{+}[n]\notin\mathcal{L}^{+}(C)$. This equation tells us that the deterministic drift condition (4.1) holds for $\bar{\mathbf{r}}\in\mathcal{R}(C)$ and thus gives us deterministic stability. This development leads to a sufficient stability condition as follows.

Theorem 4.6 (Stability Region) *A deterministic dynamic scheduler model $(\bar{B},\bar{\mathbf{r}},\mathcal{S})$ is deterministically stable under scheduling rule $Q(\cdot)$ if $Q(\cdot)$ is consistent with a (possibly non-smooth)*

norm $\| \cdot \|$ *and* $\bar{\mathbf{r}} \in \mathcal{R}(D)$ *for some finite* D.
Proof: This follows because the definition of the stability set $\mathcal{R}(D)$ implies that the deterministic drift condition is satisfied for all ℓ outside of a compact set $\mathcal{L}^+(D)$ for any $\bar{\mathbf{r}} \in \mathcal{R}(D)$. ∎

Theorem 4.6 is only useful if $\mathcal{R}(D)$ is not empty. We now show that the stability region $\mathcal{R}(D)$ for smooth norms tends towards the interior of the convex hull of the schedule constellation as D grows to ∞. More specifically, we will show that for any given $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ there exists a finite D such that $\bar{\mathbf{r}} \in \mathcal{R}(D)$.

Lemma 4.2 (Stability of any $\bar{\mathbf{r}}$) *For any deterministic model* $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ *with* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$*, there exists a positive finite number* $C_{\max}$ *such that* $\bar{\mathbf{r}} \in \mathcal{R}(D)$ *for* $D \geq C_{\max}$ *where* $\mathcal{R}(\cdot)$ *is defined with respect to a smooth norm* $\| \cdot \|$.
Proof: From Theorem 4.3 we know that there exists a set $\mathcal{D}(\bar{\mathbf{r}}, C)$ such that for $\ell^+ \notin \mathcal{D}(\bar{\mathbf{r}}, C)$ the deterministic drift condition is satisfied. From Lemma 4.1 we can choose C^m so that $\mathcal{D}(\bar{\mathbf{r}}, C) \subset \mathcal{L}^+(D)$ for all $D > C^m$. Thus the condition (4.4) holds and $\bar{\mathbf{r}} \in \mathcal{R}(D)$ for $D > C_m$. ∎

Given Theorem 4.6 and Lemma 4.2, we can state the following result.

Lemma 4.3 (Stability Region for Smooth Norms) *The stability region* $\mathcal{R}(D) \subseteq \mathrm{ch}(\bar{B} \cdot \mathcal{S})$*, and* $\lim_{D\to\infty} \mathcal{R}(D) = \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ *under scheduling rule* $Q(\cdot)$ *if* $Q(\cdot)$ *is consistent with a smooth norm* $\| \cdot \|$.
Proof: Follows from the previous lemma. ∎

An interesting fact that comes out of Lemma 4.2 is that $\mathcal{R}(D) \subseteq \mathcal{R}(C)$ when $D \leq C$. We give this as another lemma that applies even when the norm $\| \cdot \|$ is not smooth.

Lemma 4.4 (Stability Sets are Nondecreasing) *For a (possibly non-smooth) norm* $\| \cdot \|$*, when* $D \leq C$ *we have* $\mathcal{R}(D) \subseteq \mathcal{R}(C)$.
Proof: We know that $\mathcal{L}^+(D) \subseteq \mathcal{L}^+(C)$, so $\mathcal{L}^c(C) \subseteq \mathcal{L}^c(D)$ where $\mathcal{L}^c(C)$ is the complement of $\mathcal{L}^+(C)$. If $\bar{\mathbf{r}} \in \mathcal{R}(D)$ then

$$\| \ell - \bar{\mathbf{r}} \| \geq \| \ell - \bar{B}\mathbf{s} \| - \epsilon$$

for all $\ell \in \mathcal{L}^c(D)$. But this also means that this inequality holds for all $\ell \in \mathcal{L}^c(C)$. Which in turn implies that $\bar{\mathbf{r}} \in \mathcal{R}(C)$. ∎

However, Theorem 4.3 tells us that for every C there is some $D > C$ such that $\mathcal{R}(C) \subset \mathcal{R}(D)$ because eventually all $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ must be included in $\mathcal{R}(C)$ for some large enough C. We will see this in the next example.

Consider the schedule constellation $\mathcal{S}$ in Figure 4-2, and the norm $\| \cdot \|_a$ with level curves shown in Figure 4-3. Figures 4-4 and 4-5 show the stability set (as the shaded region) $\mathcal{R}(C)$ for $C = 2/3$ and $C = 2$ respectively for the constellation and norm shown in Figures 4-2 and 4-3. From the figures, we see that $\mathcal{R}(2/3)$ is smaller than $\mathcal{R}(2)$. Intuitively, as we increase C, the difference between $\mathcal{R}(C)$ and $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$ will grow smaller. If this is true, then for a given $\bar{\mathbf{r}}$ we can pick C large enough so that $\bar{\mathbf{r}} \in \mathcal{R}(C)$.

If $\| \cdot \|$ is a corner norm then this result does not hold and $\mathcal{R}(\infty) \subset \mathrm{CH}(\bar{B} \cdot \mathcal{S})$; we are left with a set of possible vectors $\bar{\mathbf{r}}$ for which the model is not guaranteed stable. Figures 4-6–4-8 show a corner norm $\| \cdot \|_b$, and the resulting sets $\mathcal{R}(2/3)$ and $\mathcal{R}(2)$ as before for the same constellation from Figure 4-2. In this case, the shaded set in Figure 4-8 is $\mathcal{R}(2) = \mathcal{R}(\infty)$. As we increase C, there is a certain point beyond which the set $\mathcal{R}(C)$ does not increase in size with increasing C.

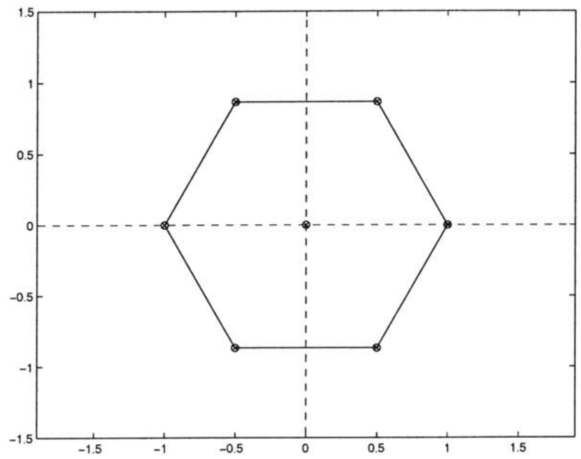


Figure 4-2: Schedule constellation $\mathcal{S}$ in the plane. Each schedule is marked with a $\otimes$.

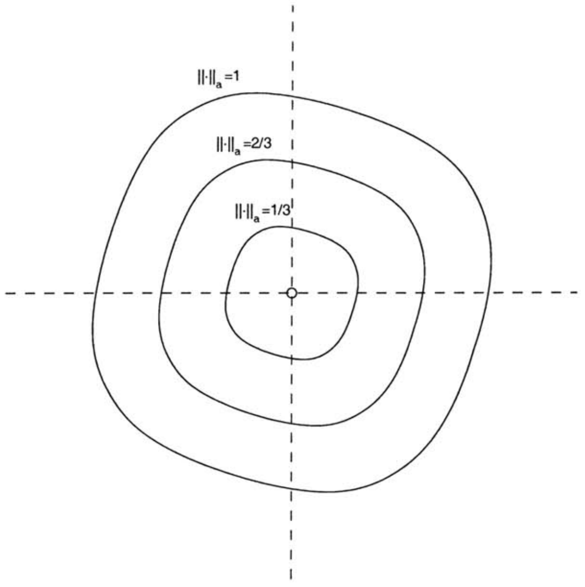


Figure 4-3: Level curves of the norm $\|\cdot\|_a$.

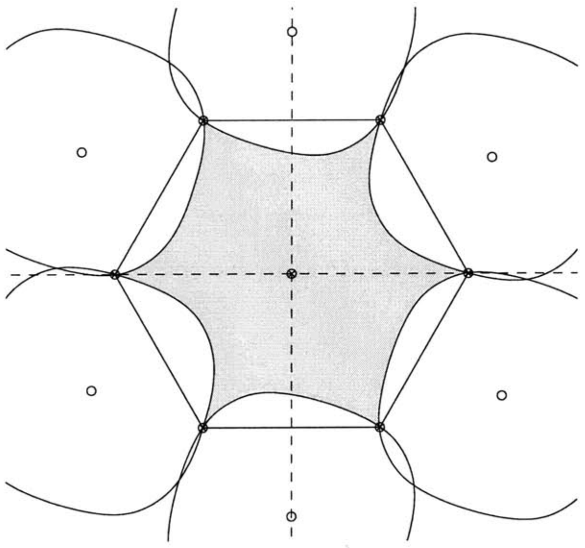


Figure 4-4: Set $\mathcal{R}(2/3)$ using the constellation and norm from Figures 4-2 and 4-3.

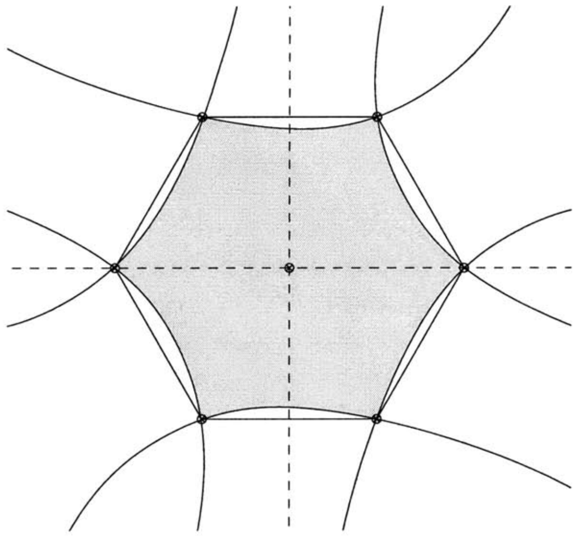


Figure 4-5: Set $\mathcal{R}(2)$ using the constellation and norm from Figures 4-2 and 4-3.

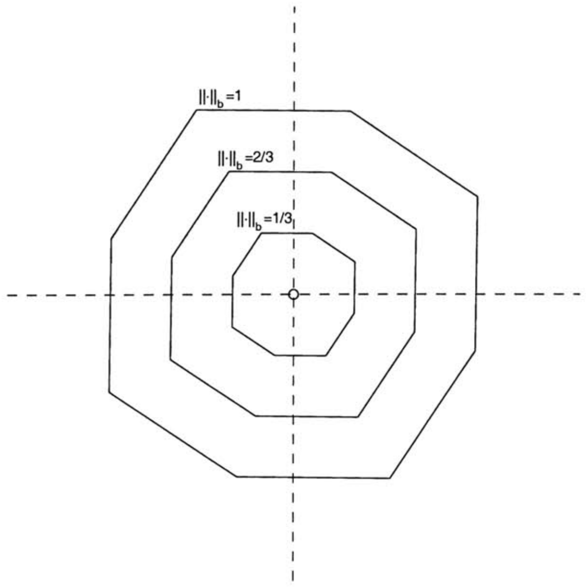


Figure 4-6: Level curves of the norm $\| \cdot \|_b$.

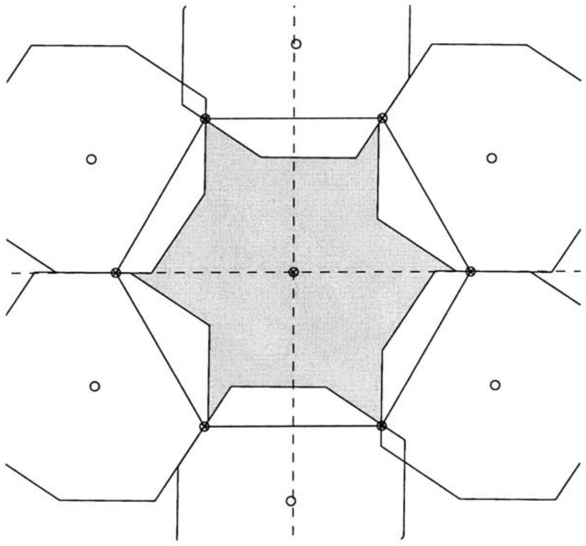


Figure 4-7: Set $\mathcal{R}(2/3)$ using the constellation and norm from Figures 4-2 and 4-6.

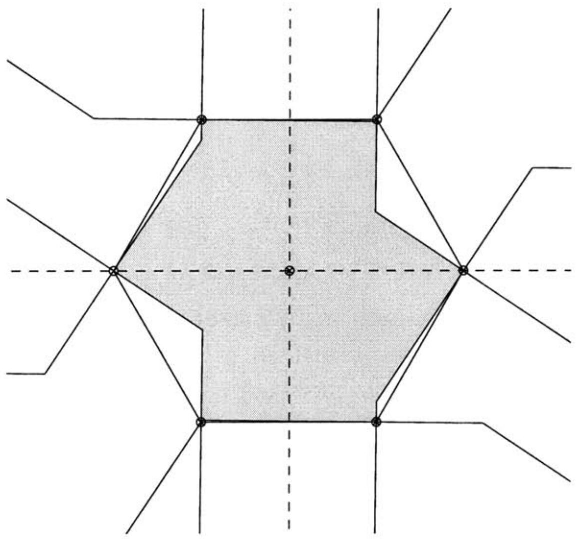


Figure 4-8: Set $\mathcal{R}(2)$ using the constellation and norm from Figures 4-2 and 4-6. This set $\mathcal{R}(2)$ is equal to $\mathcal{R}(c)$ for all $c > 2$ as well, including $\mathcal{R}(\infty)$.

The counterexample to the stability of the maximum size matching algorithm in [51] also falls in this category. It exhibits a specific vector $\bar{\mathbf{r}}$ for which $\| \cdot \|_1$ does not guarantee stability in the deterministic (or stochastic) model. However, we do not know that a model with a scheduling rule consistent with a corner norm is unstable for $\bar{\mathbf{r}} \notin \mathcal{R}(\infty)$, only that we can not guarantee stability in the sense we have defined. This is important, and a simple example is convincing. Consider a schedule constellation $\mathcal{S} = \{(0,0),(1,0),(0,1)\}$ (assume $\bar{B} = I$). The ∞-norm stabilizes the dynamic system for any $\bar{\mathbf{r}} \in \operatorname{ch}(\mathcal{S})$. This can be seen because we can define a scheduling rule consistent with the ∞-norm which is exactly equal to the rule consistent with the 2-norm (a smooth norm) for this geometry.[2] However, if we follow the same construction as before, it is clear that the set $\mathcal{R}(\infty) = \emptyset$. This tells us that the approach we used to prove stability using smooth norms can be very weak when applied to corner norms.

Despite the possible weakness of this method, it is important to realize that when the stability set $\mathcal{R}(C)$ does intersect with $\operatorname{ch}(\bar{B} \cdot \mathcal{S})$ as in Figure 4-8, Theorem 4.6 guarantees stability for a corner norm when $\bar{\mathbf{r}}$ is constrained to lie within a corresponding $\mathcal{R}(C)$. For certain corner norms and constellations, we can guarantee stability for all $\bar{\mathbf{r}} \in \operatorname{ch}(\bar{B} \cdot \mathcal{S})$. For example, if we add the point $(1,1)$ to the constellation discussed in the previous paragraph, the set $\mathcal{R}(C) = \operatorname{ch}(\bar{B} \cdot \mathcal{S})$ when C is greater than 1.

4.1.5 Corner Norms

An important question now is whether or not non-smooth norms guarantee stability of our dynamic model when they are used to determine the scheduling rule. These non-smooth norms can be more useful because they allow the scheduling rule to be implemented more simply. For example, implementing the scheduling rule which uses the ∞-norm requires a number of comparisons that grows linearly with the size of the switch in an input-queued switch, while the 2-norm necessitates a higher polynomial complexity. Linear complexity allows the scheduling rule to be built and scaled in hardware. A norm whose sub-level sets are polytopes can be computed with a fixed number (perhaps large or growing faster than $O(N)$) of linear operations and comparisons on the current lag.

The goal of this section is to illustrate simple counterexamples showing that using a non-smooth norm in a static scheduling rule, specifically the ∞- and 1-norms, does not guarantee the stability of the stochastic model even when $\bar{\mathbf{r}} \in \operatorname{ch}(\bar{B} \cdot \mathcal{S})$. Determining the behavior of these dynamic models under non-smooth scheduling rules in general is difficult, so we concentrate on a few examples.

We will concentrate here on the ∞-norm and the 1-norm, as they are simply described and commonly used as measures of distance. It is obvious that they both are non-smooth; the set of vectors v such that $\| v \|_p = 1$ in two dimensions is a square centered around the origin for $p = \infty$ and a diamond for $p = 1$. The first example below shows that, unlike the smooth norms considered previously, the ∞-norm does not guarantee the deterministic stability of all stabilizable deterministic models. The second illustrates the fact that a scheduling rule based on the 1-norm may stabilize a deterministic model while not stabilizing a corresponding stochastic model.

The 1-norm is also interesting as it relates to queueing networks. As we discussed in the previous chapter, static buffer priority policies used for scheduling reentrant multi-class queueing networks are consistent with the 1-norm. There are counterexamples in the literature [13, 37] that show the instability of specific networks when using static buffer priority policies. There is also a similar counterexample in [51] for the input-queued switching problem.

[2]The ∞-norm leaves ambiguities in the scheduling rule, where there is more than one optimal schedule over large regions. For these cases, we can always choose to make the same choice as we would with the 2-norm.

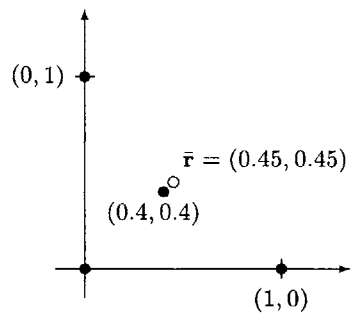


Figure 4-9: Example of unstable deterministic model.

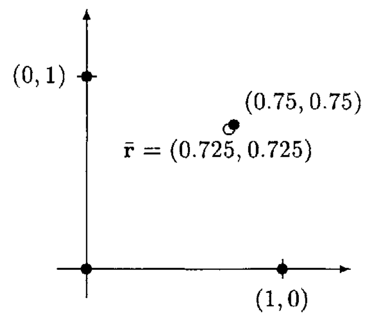


Figure 4-10: Example of stable deterministic model with corresponding unstable stochastic model.

The class of corner norms is much larger than just these two examples. For instance, it includes all norms whose level sets are polytopes, defined by a collection of linear inequalities. It also includes other mostly-smooth norms that happen to have points of unbounded curvature on the surface of their sub-level sets.

Unstable Deterministic Model

Consider a situation in which we have two users of one machine. The machine can process the users' data at a rate of 1 MB/second individually. However, the machine can also process both data streams simultaneously, but can only process each one at a rate of 0.4 MB/second because of the overhead of switching between the two. The machine switches between configurations (one of the above three, or idle) once every second. Assume that user 1 has a data rate of r_1, and user 2 has a data rate of r_2. We can plot the possible machine configurations as shown in Figure 4-9, along with a possible data rate vector $\mathbf{r} = (r_1, r_2) = (0.45, 0.45)$. Even though $\mathbf{r}$ is easily schedulable and thus the system is stabilizable for this input (simply alternate between the two users and share every so often), a scheduler that uses the ∞-norm to choose the configuration may not necessarily be stable for this case depending on the initial amount of data in the system.

A reasonable choice for a scheduler might be to choose the configuration at each switching time n that will result in the smallest remaining work for both users, $\ell[n+1]$. In other words, minimize the maximum amount of work remaining at the next switching time. This is equivalent to using $d(\ell + \mathbf{r}, \mathbf{s}) = \| \ell + \mathbf{r} - \mathbf{s} \|_\infty$. If we try this, however, the sharing configuration is chosen at each time because it maintains the smallest maximum remaining work. Because this configuration can not keep up with the incoming data rates, the remaining work grows unbounded.

However, if we instead use $d(\ell + \mathbf{r}, \mathbf{s}) = \| \ell + \mathbf{r} - \mathbf{s} \|_2$, the same thing happens for a while, but eventually one of the other configurations is chosen and the remaining work decreases. In fact, no matter what $\mathbf{r}$ is chosen (as long as it is within the convex hull of the available configurations), this choice of $d(\cdot)$ does not allow the remaining work, $\ell[n]$, to grow without bound.

This example illustrates that corner norms, or at least the ∞-norm, require careful attention if they are to be used. However, the example we give here is a contrived one; a small change in $\mathbf{r}$ will give us a stabilized model. In higher dimensions, unstable behavior can appear over a nontrivial set of reservation vectors $\bar{\mathbf{r}}$ as in the example given in [51].

Stable Deterministic Model, Unstable Stochastic Model

Another interesting problem that arises when we deal with corner norms is that stability of the deterministic model does not guarantee stability of a corresponding stochastic model.

We will use the ∞-norm again to perform the scheduling. The possible schedules are

$$\mathcal{S} = \{(0,0),(1,0),(0,1),(3/4,3/4)\}\ ,$$

shown in Figure 4-10. This system is similar to the previous example, but can work for both users somewhat independently. Arrivals occur as one of the following possibilities:

$$\mathbf{r}[n] = \begin{cases} (1,1) & \text{with probability } 0.65 \\ (1,0) & \text{w. p. } 0.075 \\ (0,1) & \text{w. p. } 0.075 \\ (0,0) & \text{w. p. } 0.2 \end{cases}$$

This corresponds with an average rate of $\mathbf{E}[\mathbf{r}[n]] = \bar{\mathbf{r}} = (0.725, 0.725)$ which is clearly within ch$(\mathcal{S})$.

We know the deterministic model $(I, \bar{\mathbf{r}}, \mathcal{S})$ is stable for these two cases using the same stability set construction used in the previous section for smooth norms. We can define a Lyapunov function based on the 1-norm and show that it always decreases for lag values outside a bounded region for this particular $\bar{\mathbf{r}}$.

Now we turn to the stochastic model $(I, \mathbf{r}[n], \mathcal{S})$ and show that this model is unstable when we use the scheduling rule

$$Q(\ell[n]) = \arg\min_{s\in\mathcal{S}} \| \ell[n] - s \|_1\ .$$

If there is more than one minimizing schedule s, we will pick one of them arbitrarily. Assume that $\ell[n]$ is on the line with unit slope and is equal to (a,a) for a large (this is not needed, any a positive or negative will do, but this assumption simplifies the analysis). The schedule $\mathbf{s}[n]$ is chosen to be $(3/4, 3/4)$ and the possible values for $\ell[n+1]$ are:

$$\ell[n+1] = \begin{cases} (a+1/4, a+1/4) & \text{with probability } 0.65 \\ (a+1/4, a-3/4) & \text{w. p. } 0.075 \\ (a-3/4, a+1/4) & \text{w. p. } 0.075 \\ (a-3/4, a-3/4) & \text{w. p. } 0.2 \end{cases}$$

We note that if $\ell[n+1] = (a-3/4, a+1/4)$ or $(a+1/4, a-3/4)$ then $\mathbf{s}[n+1] = (0,1)$ or $(1,0)$ respectively and we have the same possibilities for $\ell[n+2]$ as for $\ell[n+1]$. So being in one of these two states leads to the same trajectory as being in state (a,a). We can rewrite the possibilities equivalently as:

$$\ell[n+1] = \begin{cases} (a+1/4, a+1/4) & \text{with probability } 0.65 \\ (a,a) & \text{w. p. } 0.15 \\ (a-3/4, a-3/4) & \text{w. p. } 0.2 \end{cases}$$

But this equivalent to a Markov chain where a state a transitions to state $a+1/4$ with probability 0.65, remains at a with probability 0.15 and moves to $a-3/4$ with probability 0.2. The average drift from round n to $n+1$ is $+0.0125$. This means that on average the Markov chain moves to the right, or to larger values and the system is unstable.

It is important to remember that this unstable behavior is dependent on the initial condition. There are many initial conditions for which the mechanism is stable using the same scheduling rule and model.

4.1.6 Necessary Condition for Stability

Our earlier stability results can not be extended to cover corner norms. And the previous section gave two counterexamples that showed that corner norms do not always guarantee stability. We now give a condition on the constellation and norm that is necessary for a deterministic model to be stable for all $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$ under the quantizing scheduling rule, $Q(\cdot)$, that corresponds with the norm for both smooth and corner norms. The statement and proof of the condition is very simple. An allocation region for schedule s is said to be *linearly bounded* if there is no nonzero vector r and initial condition v_0 such that the affine sequence of points $v_\alpha = v_0 + \alpha r$ satisfies $Q(v_\alpha) = s$ for all $\alpha \geq 0$. This essentially means that there is no straight line starting somewhere in the allocation region and going to infinity completely within the region.

Theorem 4.7 (Necessary Condition for Deterministic Stability) *If a schedule s in $\text{ch}(\bar{B} \cdot \mathcal{S})$ (not on the surface) does not have a linearly bounded allocation region given the scheduling rule $Q(\cdot)$ then the deterministic model $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ is not stable for some choice of $\bar{\mathbf{r}}$.*
Proof: If this is not true, then we can pick $\ell[0] = v_0$ and $\bar{\mathbf{r}} = s + \epsilon r$ for $\epsilon > 0$ small enough that $\bar{\mathbf{r}} \in \text{ch}(\bar{B} \cdot \mathcal{S})$. With these choices $\ell[n] = \ell[0] + n\epsilon r$ and $\ell[n] = v_{\epsilon n}$, driving $\ell[n]$ arbitarily large as n increases. ■

We know that the condition is not sufficient because the maximum size matching scheduling algorithm used in the input-queued switching problem satisfies this property (trivially because it has no schedules that lie inside the convex hull), but can be unstable for some values of $\bar{\mathbf{r}}$. The second corner norm example given earlier in this section also has no interior schedules, but can still be unstable.

This necessary condition does allow us to make some conclusions about the stability of specific deterministic models by only looking at the allocation regions. For example, in the unstable deterministic model presented in Section 4.1.5, we can see that the allocation region for the schedule $(0.4, 0.4)$ includes (at least) the points $s + \alpha r = (0.4, 0.4) + \alpha \cdot (1, 1)$ for $\alpha \geq 0$. If $\bar{\mathbf{r}} = (0.45, 0.45) = s + 0.05r$, which is still inside the convex hull of the schedule constellation, the lag vector grows unbounded and we violate the above condition.

As another example, Figures 4-12 and 4-13 show quantization regions for the constellation shown in Figure 4-11 when we use the 1-norm and the 2-norm, respectively, to make scheduling decisions. The constellation spaces 8 points in the plane fairly evenly in order to have good quantization error performance over the region they occupy. However, we can see in Figure 4-12 that the quantization regions for the center points are unbounded along the horizontal axis. This violates our necessary condition and tells us that the deterministic model with this constellation is not stable for some values of $\bar{\mathbf{r}}$ when the 1-norm is used to make scheduling decisions. If we use any input $\mathbf{r}[n] = \rho = (c, 0)$ for $1 > c > 1/2$, the lag $\ell[n]$ will be equal to $\ell[0] + (n \cdot (c - 1/2), 0)$, which grows without bound. In fact even if $\mathbf{r}[n]$ is a random signal with mean $\rho = (c, 0)$ and small variance around that mean, the lag will grow large, although it is not clear without more detailed analysis whether the lag grows to infinity with any probability.

4.2 Controlling the Lag Behavior

The stability result just presented for deterministic models allows us to choose a norm on which to base our scheduler from a large class. In fact, we can relax this class to include other norm-like functions that are not necessarily norms, but which have geometric properties similar to norms. An interesting application of this result is in controlling the properties of the lag vector by varying the

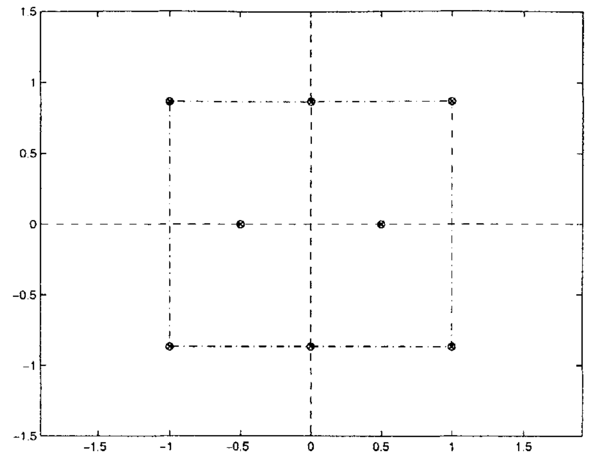


Figure 4-11: Example quantization constellation.

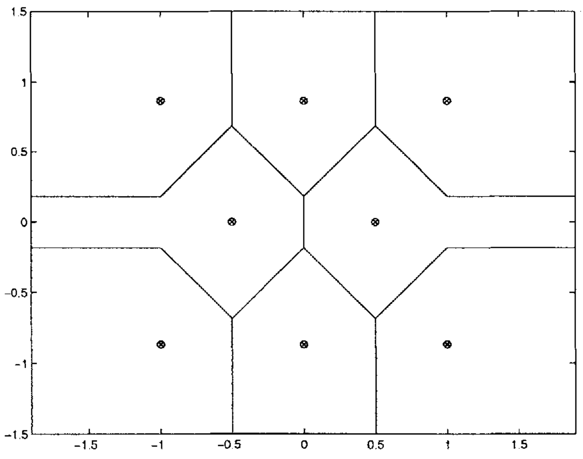


Figure 4-12: Quantization regions for the example constellation under the 1-norm.

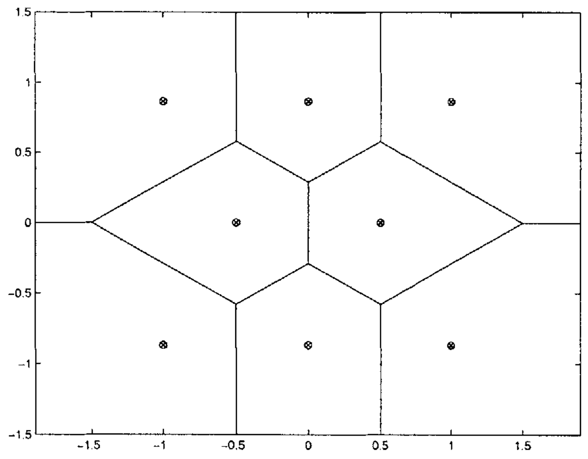


Figure 4-13: Quantization regions for the example constellation under the 2-norm.

norm that the scheduler is based on. If the lag vector remains within (or very near) the constellation its behavior is governed by the internal structure of the constellation and the distance metric used. Thus there are two ways we can affect the behavior of the lag. First, we can design the constellation to perhaps reduce the variation of the lag or bias it towards one user or another. In the following chapters we consider a specific constellation design problem that reduces the maximum value the lag can take. The other method we might use to affect the behavior of the lag is to choose an appropriate distance metric which results in favorable lag behavior.

We see an example of this in [57]; the PGPS algorithm makes scheduling decisions by doing the equivalent of scaling each component of the lag vector by the inverse of the corresponding priority and then scheduling the user with the largest resulting scaled lag. This method, instead of equalizing the worst-case lag across all of the users, equalizes the weighted worst-case lag, which can be seen to be the time delay rather than packet delay. By doing this [57] gives worst-case guarantees on the delay rather than the lag. These guarantees do not affect the throughput, or fairness of the scheduler, but they do affect the delay properties that individual users experience. By choosing the priorities appropriately the scheduler can guarantee desired delay properties. This method is equivalent to using a diagonal matrix M to weight the lag inputs to the scheduling rule. We can generalize this for our model if we view the matrix and a quantizing scheduling rule as a combined single rule. Our model allows the scheduling rule to be defined in terms of any norm, and guarantees deterministic stability if the norm is smooth and in many cases for non-smooth norms. By choosing the geometry of the level sets appropriately, we can specify how lag vectors in different directions are weighted and through this control how large the elements of the lag vector are relative to each other on average.

To illustrate this, we discuss three cases below. We extend our consideration of distance metrics to include functions that do not necessarily satisfy all of the properties of a metric. The functions will be positive and equal to 0 at the origin, but may violate any of the other properties. We will still use the term distance metric in this section, but in a non-technical sense. Our observations will mostly be based on empirical results, and we will discuss a few cases in which we can successfully choose distance metrics to modify the behavior of the lag as well as one instance illustrating that this method is not always useful.

Consider first a function $f(\cdot)$ that does not necessarily satisfy the metric symmetry property ($f(x-y) \neq f(y-x)$), but that satisfies a scaling property ($f(\alpha x) = \alpha f(x)$ for $\alpha \geq 0$) and has smooth convex sublevel sets. We will call these functions *norm-like*. The proof of Theorem 4.3 does not depend on the symmetry property, and so any smooth norm-like function can also be used as a distance metric in a quantizing scheduler without sacrificing deterministic stability. To illustrate when using a norm-like function might be useful, consider the following example for $N = 2$.

Assume we are feeding a signal $\mathbf{r}[n]$ into a Σ-Δ encoder where $\mathbf{r}[n]$ can be modeled as $\bar{\mathbf{r}} + \mathbf{w}[n]$ where $\bar{\mathbf{r}}$ is a fixed vector and $\mathbf{w}[n]$ is a sequence of independent zero mean random variables. Assume also that we have different requirements for each of the components of the error $\mathbf{e}[n] = \ell[n] - \mathbf{s}[n]$. We would like $e_1[n]$ to have a smaller variation when it is negative than when it is positive, and thus will concentrate on reducing the variation of $e_2[n]$ only when we can make $e_1[n]$ more positive. Define the sublevel sets of $f(x)$ to be concentric equilateral triangular regions (with corners rounded for smoothness) with one point directed to the right and the origin at the center of mass of the triangle. From this definition, we are weighting three directions preferentially: $(1,0)$, $(-\sqrt(3)/2, 1/2)$, and $(-\sqrt(3)/2, -1/2)$. These directions correspond to $e_2[n]$ having a small variation when $e_1[n]$ is positive and $e_2[n]$ having a large variation when $e_1[n]$ is negative. This is best illustrated with a picture; Figure 4-14 shows a representative plot of how the error is distributed when we define $\mathbf{w}[n]$ to be a vector of two independent uniform random variables. The constellation used is the same as

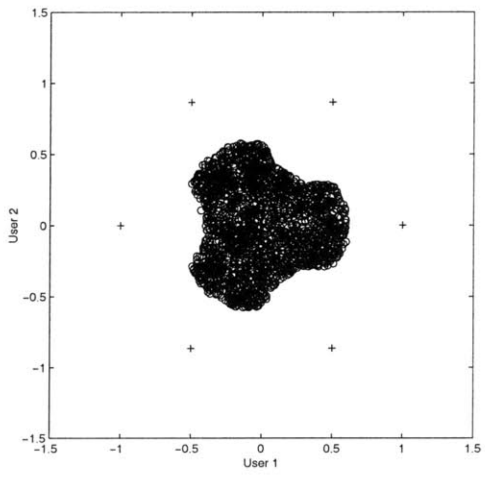


Figure 4-14: Error distribution for triangular norm-like function and hexagonal constellation.

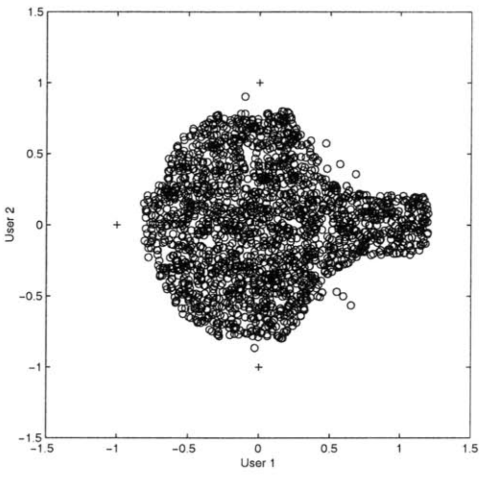


Figure 4-15: Error distribution for triangular norm-like function with diamond constellation.

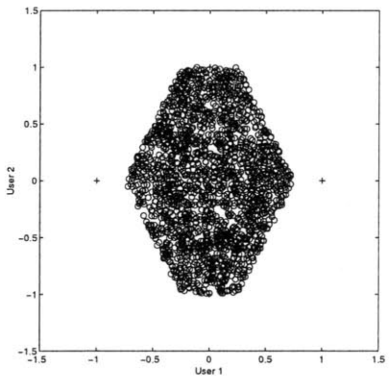


Figure 4-16: Error distribution for weighted norm function and diamond constellation.

in Figure 4-11. Each point in the plot is $\mathbf{e}[20]$ for an independent realization of $\mathbf{w}[n]$. Because $\mathbf{w}[n]$ is uniform and remains within the constellation, the resulting error distribution is bounded. The same error-shaping behavior can be seen for other constellations as well. For example, Figure 4-15 shows the behavior for a diamond constellation.

A simpler example might entail weighting the two error components differently, by using a diagonally-weighted 2-norm as the distance metric. If M is a diagonal matrix, and we use a function $f(\boldsymbol{\ell} - \mathbf{s}) = \| M(\boldsymbol{\ell} - \mathbf{s}) \|_2$ we see that the element of $\boldsymbol{\ell} - \mathbf{s}$ with index i such that $\ell_i - s_i$ corresponds with the largest element of M is weighted more highly than other elements. When minimizing the norm of this weighted vector, a schedule $\mathbf{s}$ which reduces $\ell_i - s_i$ at the expense of other elements will be preferable to a schedule that reduces all elements of $\boldsymbol{\ell} - \mathbf{s}$ equally. We plot error $\mathbf{e}[20]$ as before when the constellation $\mathcal{S} = \{(1,0), (0,1), (-1,0), (0,-1)\}$ in Figure 4-16. The elements of $\mathbf{r}[n]$ are uniform random variables, distributed in $[-0.5, 0.5]$. We use a weighting matrix M with the elements 2 and 1 on the diagonal. The average value of $|\boldsymbol{\ell}[30]|$ is $(0.44, 0.3)$, showing a reduction in the variation in the error for user 2 when compared with the average value of (0.4, 0.4) for the unweighted case. This behavior, however, depends on the constellation. If we use $\mathcal{S} = \{(-1,-1),(1,-1),(-1,1),(1,1)\}$ the diagonal weighting matrix M has no effect on the error. This is clear when we observe that the decision regions induced by the weighted norm for this constellation are identical for any matrix M.

Another example of a metric that can allow us to control the lag behavior of a system is a *skewed* norm. Skewed norms are a subset of norm-like functions. A skewed norm $f(\cdot)$ is defined as a function whose level set $d_f(C) = \{x \mid f(x) = C\}$ is equal to a translated level set $d(C) + Cx_0 = \{x \mid \| x - Cx_0 \| = C\}$ for some norm $\| \cdot \|$ and x_0 such that $\| x_0 \| < 1$. Computing the size of a vector using a skewed norm is more complex than with a real norm because the equation $\| x - Cx_0 \| = C$ must be solved for C. A skewed norm can be visualized as a norm whose center has been translated so that the sub-level sets are not symmetric around the origin. Even though the skewed norm is not a real norm, we will call it a norm because the geometry of its sub-level sets is the same as for the corresponding norm $\| \cdot \|$ except for a translation. Because of this, the sets $\mathcal{L}^+(C)$ for a skewed norm can be constructed from those of the corresponding norm by the same translation. Thus, although we do not show it in detail, we can reprise our earlier stability results for skewed norms and show that any skewed norm which corresponds with a smooth norm results in a deterministically stable model when used in the quantizing scheduling rule. For now, we will assume that the norm used is the 2-norm. In this case we can compute the value of $C = f(x)$ by solving the following quadratic equation.

$$\begin{aligned} \| x - Cx_0 \|_2^2 &= C^2 \\ \| x \|_2^2 - 2Cx^{\mathrm{T}}x_0 + C^2\| x_0 \|_2^2 &= C^2 . \end{aligned}$$

By choosing x_0, we can change shape of the decision regions and thus control the characteristics of the error.

As an example, consider a scalar system ($N = 1$). The only decision regions we can have for a scalar system are intervals containing each point in the constellation. Typically, when we use a norm as the distance metric the intervals are centered around the constellation points. Instead, if we skew the norm in one direction the intervals are no longer centered around the points. This tells us that the error distribution will similarly be shifted relative to the constellation points. Using skewed norms we can bias the error towards positive or negative values for the different error components. This works for larger N as well. We use a scalar system as an example because we can actually compute the probability distribution of the error $\mathbf{e}[n]$ for a simple two point constellation if we can model the input $\mathbf{r}[n]$ as a fixed mean plus white noise. At each round, the distribution of the lag $\ell[n+1]$ can be computed by convolving the distribution of the previous error $\mathbf{e}[n]$ with the distribution of the new input $\mathbf{r}[n+1]$. If we assume all $\mathbf{r}[n]$ are identically distributed, we can calculate these convolutions to see if the lag converges to a steady-state distribution. For example, when $\mathbf{r}[n]$ is a uniform random variable in the range $[-0.8, 0.8]$ and the constellation $\mathcal{S} = \{-1, 1\}$ with a scalar quantizer whose threshold is at 0.6 this iteration converges and the steady-state distribution is shown in Figure 4-17. The distribution is centered around 0.6, and so we have effectively biased the lag towards positive values (the convergent distribution when the threshold is at 0 is symmetric around 0 and the same shape). In fact, the error distribution is uniform on the interval $[-0.4, 1.6]$.

4.3 Stochastic Stability

We now look at a more general stochastic model with random $B[n]$ and $\mathbf{r}[n]$. Our goals are to establish that classes of stochastic mechanisms are stabilizable in some sense, and that these

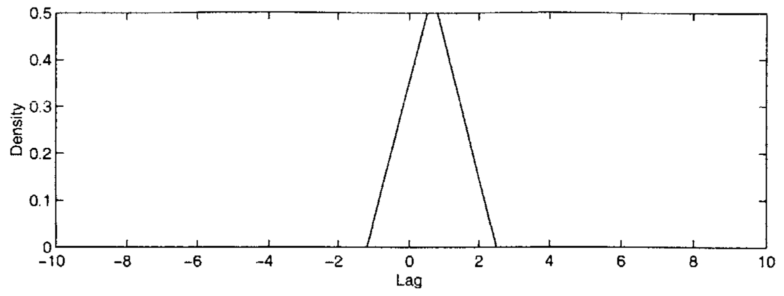


Figure 4-17: Lag distribution function for biased scalar quantizing scheduler.

mechanisms are fair. It will be clear that, just as in the deterministic case, the mechanisms are fair as we defined in Chapter 3 when a Lyapunov function of the lag exists that satisfies the stochastic drift condition 4.4. Our development here (apart from the bounded input models discussed first) depends mainly on the stability results for real-valued strict supermartingales in [46] and [25]. This section serves mainly as an illustration of how the more complex question of stochastic stability might be approached for our model.

We first discuss what the drift condition alone implies about the behavior of the lag. If we know that a Lyapunov function satisfies the stochastic drift condition for a mechanism, then we can model the evolution of the values of the Lyapunov function as a strict supermartingale. This allows us to make conclusions about the long-term behavior of the lag vector. Some of our conclusions are based on the assumption that the probability distribution of $\ell[n+1] - \ell[n]$ satisfy certain assumptions, which restricts the set of sequences $\mathbf{r}[n]$ and $B[n]$ to which the conclusions apply to those with appropriately constrained probability distributions. For example, a one-dimensional Σ-Δ encoder was analyzed in [35] and shown to have favorable properties such as a steady state distribution for the quantization error (and thus the lag) when the input $\mathbf{r}[n]$ is a sequence of independent and identically distributed Gaussian random variables ($B[n] = I$) and the distortion metric is the squared 2-norm. We use the more general stability results (which do not assume that the process of interest is Markovian) in [25] in order to establish a fairly strong level of stochastic stability.

We then show that the drift condition is satisfied for the specific case when the scheduling rule is consistent with the squared 2-norm for any white sequences $\mathbf{r}[n]$ and $B[n]$ with finite second moments. This result can be extended to a larger class of sequences for which the moments of $\mathbf{r}[n]$ and $B[n]$ are bounded conditioned on the past.

4.3.1 Bounded Input Stochastic Models

Before we get to the drift condition, we consider stochastic models with fixed $B[n] = \bar{B}$, and $\mathbf{r}[n] \in \mathbf{R} \subset \mathrm{ch}(\bar{B} \cdot \mathcal{S})$. We allow $\mathbf{r}[n]$ to vary arbitrarily within a closed subset of $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$. This case was considered in Section 4.1.3 and shown to be deterministically stable; this means that $\ell[n]$ remains bounded within a fixed region after an initial transient period has elapsed. This case covers a number of different interesting situations, including the Σ-Δ encoder for which the quantization constellation is chosen so that the range of the input signal is contained within the convex hull of the constellation. If we know that the input signal stays within the convex hull of the quantization points, then the lag will remain deterministically bounded. This class of stochastic models is deterministically stable and thus both deterministically and stochastically fair.

4.3.2 Stochastic Drift Condition

The definition of the *stochastic drift condition* is similar to that of the deterministic drift condition except that it is stated as an expectation of an appropriate Lyapunov function. We define it as follows, using the notation $\mathbf{E}_\ell[V(\ell[n+1])]$ to mean the expected value of $V(\ell[n+1])$ given that $\ell[n] = \ell$. All conditional expectations and relations between random variables are considered to hold with probability 1 unless otherwise specified. We also assume that $\mathbf{E}_\ell[V(\ell[n+1])]$ is finite for any finite value $\ell[n] = \ell$.

Definition 4.4 (Stochastic Drift Condition) *The process $\ell[n]$ defined by a model $(B[n], \mathbf{r}[n], \mathcal{S})$ and a scheduling rule $Q(\cdot)$ is said to satisfy the* stochastic drift condition *if there exists a positive- and finite-valued level-bounded function $V(\cdot)$, a compact set $\mathcal{C} \subset \mathbb{R}^N$, and a constant $M < \infty$ such that*

$$\mathbf{E}_\ell[V(\ell[n+1])] - V(\ell[n]) \leq -\epsilon \tag{4.5}$$

for $\epsilon > 0$, all $\ell[n] \notin \mathcal{C}$ and

$$\mathbf{E}_\ell[V(\ell[n+1])] - V(\ell[n]) \leq M < \infty \tag{4.6}$$

for $\ell[n] \in \mathcal{C}$.

It is possible to reach some conclusions about the stability of a model given only that it satisfies our stochastic drift condition. For example, if the sequence $\ell[n]$ is a Markov sequence (i.e. when $\mathbf{r}[n]$ and $B[n]$ are white) a weak notion of stability called *non-evanescence* is guaranteed when a model satisfies a relaxed version of the stochastic drift condition (see Theorem 9.4.1 in [53]). Non-evanescence guarantees that the probability that a sample path of the chain $\ell[n]$ goes to ∞ is 0. This means that $\ell[n]$ visits at least one compact subset of $\mathbb{R}^N$ infinitely often, but does not constrain the length of the times in between visits. Non-evanescence also does not guarantee that the expectation of the lag is bounded.

In order to bound the expectation of the lag, we note that Equation 4.5 also implies that the quantity $V(\ell[n])$ is a *strict supermartingale* before $\ell[n]$ enters $\mathcal{C}$. A supermartingale [24] is a sequence of random variables V_n such that

$$\mathbf{E}[V_{n+1} \mid V_i, i \leq n] \leq V_n \ .$$

A strict supermartingale is a supermartingale such that there is an $\epsilon > 0$ such that for all n and i

$$\mathbf{E}[V_{n+1} \mid V_i, i \leq n] \leq V_n - \epsilon \ .$$

It is immediately clear that a strict supermartingale is a supermartingale.

The drift condition we have defined is the same as Condition C1 used in [25]. In [25], it is shown that if a sequence of random variables satisfies this drift condition and if the increments $|V_{n+1} - V_n|$ are *exponential type* (Condition C2), then each element V_n is also exponential type. A random variable V_n that is exponential type essentially means that the variable has a finite exponential moment, or that $\mathbf{E}[\alpha^{V_n}] \leq A < \infty$ for some $\alpha > 0$. This in turn implies that the tail of the probability distribution of V_n is asymptotically bounded below an exponential.

This assumption on the distribution of the increments is not nearly as restrictive as the one required for deterministic stability. This assumption is satisfied, for example, when $\mathbf{r}[n]$ and $B[n]$ are constrained to lie within bounded regions. This is a looser constraint than for deterministic stability, for which we required $\mathbf{r}[n]$ to lie within $\mathrm{ch}(\bar{B}\cdot\mathcal{S})$ and $B[n]$ to be constant. The assumption here even allows the region of support of the probability distribution of $|V_{n+1}-V_n|$ to be unbounded, as long as the distribution itself has subexponential tails.

Under these conditions, the results from [25] tell us the following about the sequence $\{V_i\}$ given an initial value V_0 of exponential type (we define a as the smallest number such that $V(\ell) \leq a$ for all $\ell \in \mathcal{C}$):

- Every V_k is exponential type;
- Starting at V_0, the time τ when V_τ is first below a is exponential type; and
- The probability that $V_k > b$ is exponentially decreasing in b.[3]

All these together tell us that $V(\ell[k])$ does not travel far above a. This means that $\ell[k]$ is bound closely to the compact set $\{\ell \mid V(\ell) \leq a\}$. Specifically, all of the polynomial moments of $V(\ell[k])$ are finite for all k.

This is a fairly strong form of stability, as it tells us that the probability distributions for the elements $V(\ell[k])$ maintain nice properties (finite moments and exponential tails whose rate of decay has a finite lower bound independent of k) for any value of k. Thus, if we can show that the sequence of values $V(\ell[k])$ satisfies conditions C1 and C2 above for some mechanism, we know that the mechanism will be very well behaved.

4.3.3 Fairness

In particular, the previous properties of the sequence $V(\ell[k])$ may also give us guarantees on the fairness of a mechanism. As we have discussed, bounded lag implies deterministic fairness; for the stochastic model we have just seen that the probability that the lag leaves a bounded region defined as the set of points ℓ such that $V(\ell) \leq b$ decreases exponentially with the size of the region. This tight stochastic boundedness implies that, for many choices of $V(\cdot)$, the fact that the polynomial moments of $V(\ell[k])$ are finite implies that $\mathbf{E}[\| \ell[k] \|]$ is also finite. This means that a mechanism that satisfies conditions C1 and C2 with such a function $V(\cdot)$ is also fair. As a simple example, if conditions C1 and C2 hold and $V(\cdot)$ is defined to be a norm then the mechanism is stochastically fair by definition.

4.3.4 The 2-Norm

Our stability results require that the mechanism satisfy the stochastic drift condition above. We show here that the condition (4.5–4.6) is satisfied for a common scheduling rule. A stochastic model with independent white processes $\mathbf{r}[n]$ and $B[n]$ satisfies the stochastic drift condition when we use the following scheduling rule. At each stage, choose the schedule $\mathbf{s}[n]$ that minimizes

$$\mathbf{E}\left[\| \ell[n] - B[n+1]\mathbf{s}[n] + \mathbf{r}[n+1] \|_2^2 \mid \ell[n]\right] .$$

Another related scheduling rule chooses $\mathbf{s}[n]$ that minimizes

$$\| \ell[n] - \bar{B}\mathbf{s}[n] \|_2^2$$

when $B[n] = \bar{B}$. This is useful for situations when we can only make decisions based on the current state $\ell[n]$, such as in queueing networks or when the statistics of $\mathbf{r}[n]$ are unknown; we can not

[3]This holds on all but a set of outcomes whose probability decreases to 0 as k grows to ∞.

schedule future arrivals before they arrive. We concentrate on the first scheduling rule, but it is clear that the drift condition also holds when the second rule is used. These are technically not quantizing schedulers as we have defined them because the distance metric is the squared distance $\| \cdot \|_2^2$ rather than the distance itself, and thus is not a norm. We discuss how we might extend the class of schedulers that satisfy the drift condition to include all smooth norms in the conclusion.

This type of result is not particularly new; it uses a quadratic Lyapunov function to satisfy a drift condition, and related results have been developed in a number of different situations, such as [66, 51, 52, 39]. The proof follows a similar path as these others, by exhibiting a Lyapunov function of the lag vector that decreases if the lag vector lies outside of a compact region. This theorem includes restricted cases of some of the previous results and extends the stability result to our geometric model. However, these previous results all apply to situations in which the state space is countable. Our result extends this class of results to a more general state space. A simple version of this result has been shown for a one-dimensional Σ-Δ encoder with IID Gaussian inputs [35], but not extended to more general models.

By viewing our model within a geometric framework, the intuition behind this theorem is simple: if we simply choose the schedule that minimizes what we expect the value of the next state vector to be, then the state vector will stay bounded. In the deterministic lag model, this corresponds to picking the schedule that is closest to the current lag vector plus the next reservation vector, or the point closest to where the lag vector is heading next. This intuition holds even when we have a stochastic model, with an arbitrary schedule constellation, as long as $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$.

Theorem 4.8 (Drift of the 2-Norm for Independent Inputs) *Given dynamics*

$$\ell[n+1] = \ell[n] - B[n+1]\mathbf{s}[n] + \mathbf{r}[n+1] \ ,$$

if we choose $\mathbf{s}[n]$ *so that*

$$\mathbf{s}[n] = \arg\min_{s \in \mathcal{S}} \mathbf{E}\left[\| \ \ell[n] - B[n+1]\mathbf{s} + \mathbf{r}[n+1] \ \|_2^2 \mid \ell[n]\right] \ ,$$

and if $B[n]$ *and* $\mathbf{r}[n]$ *are independent sequences of independent random variables with finite means and second moments such that* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$*, then* $\ell[n]$ *satisfies the stochastic drift condition (4.5–4.6).*
Proof: See Appendix, Section B.3. ■

From this, we know that the expected value of $\| \ \ell[n] \ \|_2$ is bounded because $\mathbf{E}[\| \ \ell[n] \ \|_2] < \infty$ if $\mathbf{E}[\| \ \ell[n] \ \|_2^2] < \infty$. This theorem also implies that the variances of the components of the lag are finite because $\mathbf{E}[\ell_i^2[n]] \le \mathbf{E}[\| \ \ell[n] \ \|_2^2] < \infty$. For this particular case, a stochastic model with white sequences $\mathbf{r}[n]$ and $B[n]$ and a scheduler consistent with the squared 2-norm, the mechanism is not only stochastically fair, but the lag also has a finite variance.

The constraint that $B[n]$ and $\mathbf{r}[n]$ are independent and consist of independent random variables is rather limiting. We define $\mathbf{E}_\Sigma[\cdot]$ as the expectation of the argument given the previous history of $\mathbf{r}[n]$, $B[n]$, and $\ell[n]$. In situations such as the vector Σ-Δ encoder, the input sequence will often be correlated over time; in these situations $\mathbf{E}_\Sigma[\mathbf{r}[n]]$ is not necessarily equal to $\bar{\mathbf{r}}$, because the current state may be correlated with, and thus provide some knowledge of the previous inputs. However, the only conditions that are required on $B[n]$ and $\mathbf{r}[n]$ in the proof of Theorem 4.8 are that $\mathbf{E}_\Sigma[\mathbf{r}[n]]$, $\mathbf{E}_\Sigma[B[n]]$ and $\mathbf{E}_\Sigma[\| \ B[n+1]\mathbf{s}[n] - \mathbf{r}[n+1] \ \|_2^2]$ be finite. This is equivalent to saying that the first and second moments of $\mathbf{r}[n+1]$ and $B[n+1]$ be finite, conditioned on knowledge of the previous inputs and lag values. This will be true under many circumstances, such as when both $\mathbf{r}[n]$ and $B[n]$ are finite. In light of this, we can immediately state a corollary to the previous theorem.

Corollary 4.1 (Drift of the 2-Norm) *Given dynamics*

$$\ell[n+1] = \ell[n] - B[n+1]\mathbf{s}[n] + \mathbf{r}[n+1] \ ,$$

such that the first and second moments of $\mathbf{r}[n]$ *and* $B[n]$ *are finite,* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ *and the following conditions are satisfied:*

$$\begin{aligned} |\mathbf{E}_\Sigma[\mathbf{r}[n+1]] - \bar{\mathbf{r}}| \leq R_1 < \infty \quad &, \quad |\mathbf{E}_\Sigma[\| \ \mathbf{r}[n+1] \ \|_2^2] - \mathbf{E}[\| \ \mathbf{r}[n+1] \ \|_2^2]| \leq R_2 < \infty \ , \\ |\mathbf{E}_\Sigma[B[n+1]] - \bar{B}| \leq B_1 < \infty \quad &, \quad |\mathbf{E}_\Sigma[\| \ B[n+1] \ \|_2^2] - \mathbf{E}[\| \ B[n+1] \ \|_2^2]| \leq B_2 < |\infty \ , \end{aligned}$$

if we choose $\mathbf{s}[n]$ *so that*

$$\mathbf{s}[n] = \arg\min_{s \in \mathcal{S}} \mathbf{E}\left[\| \ \ell[n] - B[n+1]\mathbf{s} + \mathbf{r}[n+1] \ \|_2^2 \mid \ell[n]\right]$$

then $\ell[n]$ *satisfies the stochastic drift condition (4.5–4.6).*

It is important to note that this corollary does not place any restrictions on the densities of $\mathbf{r}[n]$ or $B[n]$ other than the ones mentioned. Thus, the corollary applies to a large subset of models in our general stochastic framework.

This result is restrictive in another sense. Unlike the deterministic model, we can not guarantee (with this approach) the stochastic fairness of a stochastic model when the quantizing scheduler is consistent with an arbitrary smooth norm. It appears that this may be true, but we do not have a proof yet. If it were true that stochastic models are stabilizable with smooth quantizing schedulers as the deterministic models are, then we can use the same approach to controlling the lag behavior by choosing an appropriate scheduler as we did for the deterministic models in Section 4.2.

Chapter 5

Variable-Sized Channels

The previous chapter explored the large-scale stability of the dynamic model we presented in Chapter 3. In this chapter, we move to an investigation of the smaller-scale performance of the model for a particular resource allocation problem. Specifically, we consider a constellation design problem for which we try to minimize the worst-case variation of the lag.

We consider a resource allocation context in which a shared resource is made up of a collection of smaller pieces. We are most interested in the case when the number of pieces is much larger than the number of users sharing the resource, and there are no constraints on the possible allocations of the pieces to the users except that the pieces are not further divisible: specifically, any user can use any subset of pieces from the resource at any time. The scheduling algorithm we use is simply a minimum-distance quantizer that chooses the schedule which minimizes the maximum componentwise difference between the user requests and the amount of resource each user receives.

Later, in Chapter 6, we will look at a dynamic extension of the static problem considered here that fits within the class of models presented in Chapter 3. The problem that we present here is one in which we are trying to partition a shared resource into a fixed number of fixed-size pieces, such that when incoming requests for resources are considered after we have chosen the partition, we can use the pieces to approximate the unknown vector of requests as closely as possible. A simple analogy for this problem that captures many of the details is deciding the minimum number and denominations of bills that you would need to make change for a given dollar amount. It is clear that in order to make any amount of change less than a 100 dollars, the minimum number (7) of bills might have denominations 1, 2, 4, 8, 16, 32, and 64 dollars.[1] Another question you might ask is, given that you can have at most 7 bills, what is the largest amount you could make change for? Using the denominations above, you could make change for anything up to \$127. Our results from Section 5.4 give a solution to more general versions of these two problems. For the first, we solve the problem in which you need to make change for N people using M bills and such that the total change made is no more than 100 dollars. For the second question, our results tell us the largest amount of change that can be made for N people given M bills. The results also give us a characterization of all sets of bill denominations that are able to make change for any amount up to their total value.[2] A basic assumption we make in this chapter is that the resource pieces are strictly additive; i.e., if a user receives more than one piece, the set of pieces is just as useful as a single piece with the same total size.

[1]The last bill could actually be anything between 37 and 64 dollars.

[2]For example, bills with denominations 1, 2, 4 and 5 dollars can make change for anything up to 12 dollars, bills with denominations 1, 2, 3 and 6 can also, but bills with denominations 1, 3, 3 and 5 dollars can not, although the total value $(1 + 3 + 3 + 5 = 12)$ is the same.

A similar problem that has been studied in depth is the quantization constellation design problem for vector quantizers [21]. The vector quantization goal is to choose the points in a fixed-size constellation to minimize the average or worst-case error between a stochastic input value and the closest constellation point given a distribution of possible input values. This general problem is typically difficult to solve, so constraint techniques such as looking only at lattice-based quantizers that reduce the possibilities are used to make the solution easier to find. Our problem differs in that we choose a number of other parameters, the sizes of the resource pieces, that define the constellation indirectly through the process of allocating resource pieces to inputs. However, the ultimate optimization goal — minimizing some measure of error — is the same.

In Section 5.1, we describe this static resource allocation problem. Issues related to the optimization problem are discussed and the concept of a uniformly dense constellation is defined in Section 5.2. A special case of the problem in which the pieces of the resource are all equally-sized is used to illustrate the geometry of the problem in Section 5.3, and the minimum distance is determined for this case. Then in Section 5.4 we use the uniformly dense notion as a way of constraining the general problem to be easily solvable, and show how the solution relates to the case in Section 5.3. Section 5.5 presents examples of nonuniform constellations that perform better than the uniformly dense constellation that is the solution to the constrained problem in Section 5.4. Finally, we give a few possible applications for the results of the earlier sections and some extensions of the basic model in Chapter 6 by embedding our static model in a repeated resource allocation mechanism much like the ones in [3, 51, 57]. We discuss a class of resource partitions that are robust to the deletion of elements, i.e. sets of piece sizes that still yield uniformly dense constellations when some of the pieces are removed in Section 6.6.

5.1 Problem Statement

5.1.1 Static Problem

We state our problem in terms of the change-making analogy as follows: given that you may only have a total of M bills whose total value is 100 dollars, determine the denominations of the bills such that, if you are to distribute them among N people who each request a portion of the 100 dollars and collectively request the total dollar amount, then the distribution has the smallest worst-case under- or over-payment to a person. We show in Chapter 6 that this problem can be looked at in a few different ways in order to model more practical problems.

The more abstract problem definition we consider is the problem of choosing the sizes, $\mathbf{p} = (p_1, \ldots, p_M)$, of M pieces of a single, divisible resource with total size $\sum_i p_i = 1$, $p_i > 0$, so that the *optimal allocation* of these pieces to N bins with *unknown* capacities $\mathbf{r} = (r_1, \ldots, r_N)$, $\sum_j r_j = 1$, $r_i \geq 0$ minimizes the amount by which any bin is over- or underfilled for all possible $\mathbf{r}$. The collection of all possible values of $\mathbf{r}$ is denoted by $\mathbb{S}_N$. The region $\mathbb{S}_N$ is often called a *simplex*. It has a very simple description as the set of all nonnegative vectors that sum to 1, or as the set of all convex combinations of vectors equal to columns of the identity matrix. For example, $\mathbb{S}_3$ is the area inside the circumscribing triangle in the plot in Figure 5-1). Throughout this chapter and next, we will assume that $\mathbf{r} \in \mathbb{S}_N$. We refer to a vector of sizes, $\mathbf{p}$, as a *size vector*. An *allocation* is a mapping of the pieces to the bins. The set of all possible allocations can be represented by the set $\mathcal{A}$ of all possible $M \times N$ 0-1 matrices that have a single 1 in each row. We define a *schedule* with respect to an allocation $\mathbf{A} \in \mathcal{A}$ and a particular $\mathbf{p}$ as

$$\mathbf{s_A}(\mathbf{p}) = \mathbf{pA} \ ,$$

where $\mathbf{A}$ is an $M \times N$-dimensional matrix whose element A_{ij} is equal to 1 if the i-th piece is allocated to the j-th bin, and is otherwise 0. A schedule describes the total size of the pieces allocated to each of the bins. The collection of achievable schedules given some *size vector* $\mathbf{p}$ is denoted by $\mathcal{S}(\mathbf{p})$; we call this set the *schedule constellation*, or more simply, the constellation. Because we can always allocate the entire resource to any of the users, the convex hull of the constellation is always equal to $\mathbb{S}_N$. Although a constellation is completely determined by a given $\mathbf{p}$, a constellation does not determine a unique $\mathbf{p}$. The error properties and performance measures that we care about are properties of the geometry of the constellation, but the constellation may be achievable using a number of different $\mathbf{p}$ vectors.

By an optimal allocation, we mean an allocation of the pieces that minimizes the maximum amount any bin is over- or underfilled for a given $\mathbf{p}$. We call the schedule that results from the optimal allocation the *optimal schedule.* Below we will use the notation $\| \mathbf{x} \|_\infty$ to mean the maximum componentwise absolute value of the vector $\mathbf{x}$, or

$$\| \mathbf{x} \|_\infty = \max_j |x_j| \ .$$

We will use $\mathbf{A}^*(\mathbf{p}, \mathbf{r})$ to denote the optimal allocation for a given $\mathbf{p}$ and $\mathbf{r}$:

$$\mathbf{A}^*(\mathbf{p}, \mathbf{r}) = \arg \min_{\mathbf{A} \in \mathcal{A}} \| \mathbf{s}_\mathbf{A}(\mathbf{p}) - \mathbf{r} \|_\infty = \arg \min_{\mathbf{A} \in \mathcal{A}} \| \mathbf{pA} - \mathbf{r} \|_\infty \ ,$$

and the optimal schedule is

$$\mathbf{s}^*(\mathbf{p}, \mathbf{r}) = \mathbf{pA}^*(\mathbf{p}, \mathbf{r}) \ .$$

We define the maximum error for a particular choice of $\mathbf{p}$ and $\mathbf{r}$ as

$$e(\mathbf{p}, \mathbf{r}) \quad = \quad \| \mathbf{s}^*(\mathbf{p}, \mathbf{r}) - \mathbf{r} \|_\infty \ .$$

Then, the quantity we are trying to minimize with our choice of $\mathbf{p}$ is

$$E(\mathbf{p}) \quad = \quad \max_{\mathbf{r} \in \mathbb{S}_N} e(\mathbf{p}, \mathbf{r}) \ . \tag{5.1}$$

With this terminology, we can restate our objective as finding the vector of sizes $\mathbf{p}^*$ such that

$$\mathbf{p}^* = \arg \min_{\mathbf{p} \in \mathbb{S}_M} E(\mathbf{p}) \ . \tag{5.2}$$

Note that $\sum_i r_i = \sum_i p_i = 1$. One aspect of this objective that is very important in how we choose to solve the problem is that simply evaluating $E(\mathbf{p})$ is very hard to do for an arbitrary vector $\mathbf{p}$. However, for restricted classes of vectors $\mathbf{p}$, the evaluation becomes easy.

5.1.2 Dynamic Problem

Although we will discuss the dynamic model in detail later in Chapter 6, we preview the idea here. We can begin to think of the dynamic model in the following way: each morning, M packages with fixed weights $\mathbf{p}$ are to be loaded onto N trucks with maximum load ratings of $\mathbf{r}$. If the total package weight is equal to the total load rating, we would like to place the packages on the trucks in order to minimize the amount by which we exceed any truck's load rating. At the same time, over many days we would like to make sure that each truck is loaded to exactly its load rating on average. To do this, we choose the best allocation as above, and allocate the packages to the trucks. The difference between the allocation vector and the vector of load ratings $\mathbf{r}$ is measured as an error vector and used when we fill the next set of trucks. By subtracting this error from the actual sizes of the trucks before we allocate the packages, we make sure that any trucks that were overfull the previous day are less full the next. As this repeats over many days, the error propagates forward; if the error does not grow without bound, then we can say that the each truck is filled exactly to capacity on average. The simple feedback of the error guarantees that on average we will not be biased towards any particular truck. This also indicates why we assume that the vector $\mathbf{r}$ is unknown. As the error evolves over time, the effective truck sizes (after subtracting the error) to which we will be allocating also change and may be somewhat unpredictable. So, we would like our solution to the previous optimization problem to work well no matter what happens to the error.

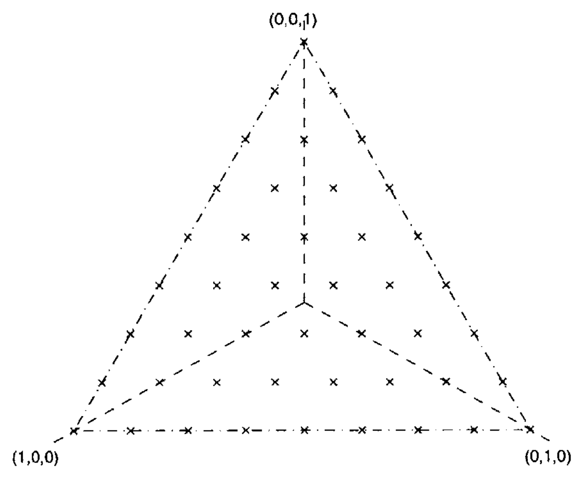
5.2 Preliminaries

It is important to realize that solving Equation (5.2) is trivial if the capacities $\mathbf{r}$ are known before we have to choose $\mathbf{p}$; we simply choose $\mathbf{p} = \mathbf{r}$. When $\mathbf{r}$ is *not* known, the choice of $\mathbf{p}$ must work well for any possible $\mathbf{r}$. Even the method of solving for $E(\mathbf{p})$ for any particular $\mathbf{p}$ is not obvious. Theorem 5.1 gives us the value of $E(\mathbf{p})$ when the elements of $\mathbf{p}$ are all equal, but for an arbitrary $\mathbf{p}$ this calculation is not as simple. We can begin to understand why it might be more or less difficult to compute $E(\mathbf{p})$ for a particular $\mathbf{p}$ by considering the schedule constellation $\mathcal{S}(\mathbf{p})$ associated with the size vector $\mathbf{p}$. The geometric properties of the constellation $\mathcal{S}(\mathbf{p})$ can help us understand why the calculation of $E(\mathbf{p})$ might be complex in general.

In solving Equation (5.2), we are essentially trying to design a vector $\mathbf{p}$ that gives rise to a schedule constellation with somewhat regularly spaced points, and thus small maximum error. This follows from the definition of $E(\mathbf{p})$, which is the maximum distance between any value of $\mathbf{r}$ and the closest point $\mathbf{s}^*(\mathbf{p}, \mathbf{r})$ in the constellation.

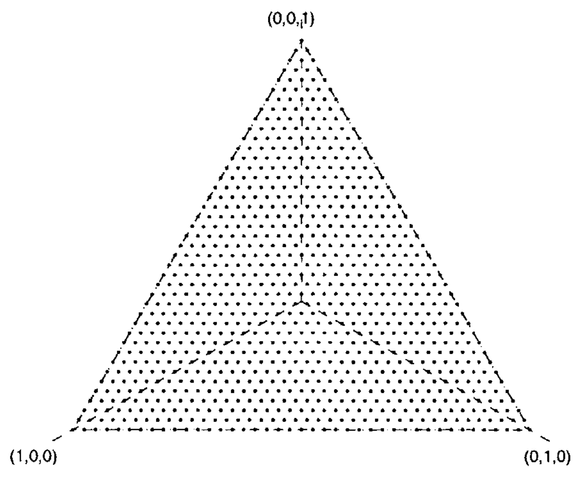
Figures 5-1–5-4 show a number of constellations that correspond to different vectors $\mathbf{p}$ for $M = 8$ pieces and $N = 3$ bins. We will be referring to these plots to illustrate ideas throughout the thesis. The plots are limited to the case when $N = 3$, to show the geometry of the constellations as clearly as possible. The plots are made so that the viewer is looking in the direction $(-1, -1, -1)$ towards the origin. The triangle indicates the boundary of the region that contains all nonnegative points that sum to 1, or all points $\mathbf{r}$ where $\sum_i r_i = 1$, and $r_i \geq 0$. For $N = 3$, this region is a 2-dimensional triangle whose vertices (labeled on the plots) lie on the axes. The points of the constellation, or elements of $\mathcal{S}$, are indicated by $\times$'s or dots, and all lie in the 2-dimensional triangle. The axes are indicated by the lines radiating out from the origin.

Figure 5-1 shows a constellation that is very regular, and corresponds to a size vector $\mathbf{p}_0$ whose entries are all equal, $p_i = 1/M$ for all i. In order to compute $E(\mathbf{p}_0)$ for this constellation, we have the advantage that the regular geometry makes the computation easier. Theorem 5.1 tells us that the maximum error in this case is $1/12$. The next constellation, shown in Figure 5-2, is also completely regular. However, it is much denser than the first, even though we use the same number of pieces ($M = 8$) to generate it. In fact, $E(\mathbf{p}_1)$ for this constellation is equal to $1/60$, smaller than the first by a factor of 5. The third constellation, Figure 5-3, is almost completely regular; it has a small number of holes. Computing $E(\mathbf{p}_2)$ involves only looking at the holes, as they can only increase $E(\cdot)$ beyond a completely regular constellation with the holes filled. It turns out that for this case, the maximum error is also $1/60$. If we were to look at the average error, we would find that this third constellation has a much smaller average error than the first two (see Section 5.5). The last constellation, Figure 5-4, has very complex geometry and is representative of constellations for most arbitrary size vectors; computing $E(\mathbf{p}_4)$ looks much more complicated. For this specific



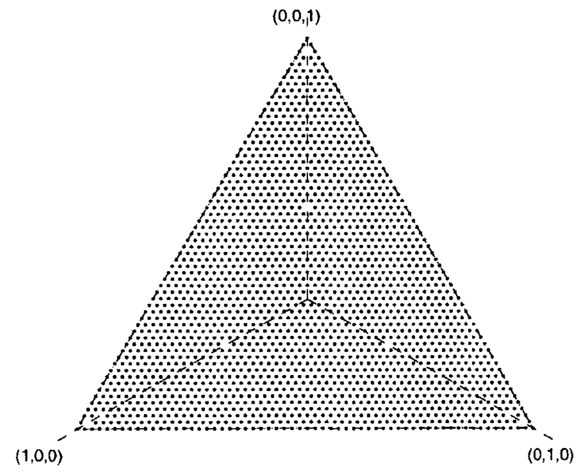
$\mathbf{p}_0 = (1/8) \cdot (1,1,1,1,1,1,1,1)$

Figure 5-1: Constellation with 8 equally-sized pieces.



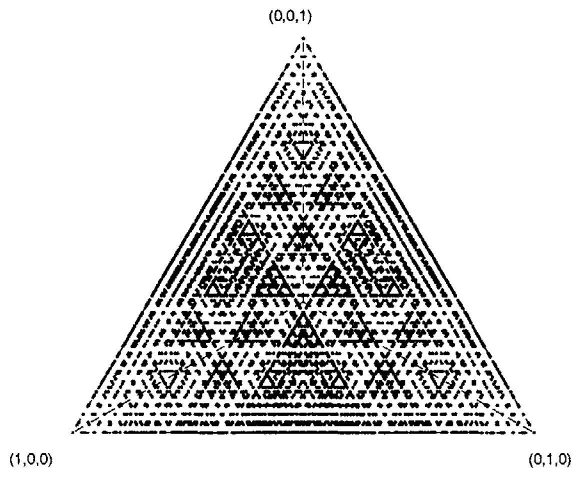
$\mathbf{p}_1 = (1/40) \cdot (1,1,2,3,4,6,9,14)$

Figure 5-2: Maximally dense uniform constellation with 8 pieces.



$\mathbf{p}_2 = (1/60) \cdot (1,2,3,4,6,9,14,21)$

Figure 5-3: Slightly more dense constellation with 8 pieces, but not quite uniformly dense.



$\mathbf{p}_3 = (1/6277) \cdot (128,164,288,432,648,972,1458,2187)$

Figure 5-4: A very nonuniform constellation more dense than any uniformly dense constellation.

example, an exhaustive examination of the constellation tells us that the maximum error is strictly less than 1/60 (and also that the average error is smaller than that of the constellation in Figure 5-3). Although this last constellation has a smaller maximum error than the previous two, it is more complicated to compute the error, and it is also appears very complicated to compute the optimal schedule given a particular value of $\mathbf{r}$. Currently, it is not clear how to compute $E(\mathbf{p})$ efficiently for an arbitrary $\mathbf{p}$ with a schedule constellation with very nonuniform geometry.

Uniformly Dense Constellations

At the end of Section 5.5, we show that the problem of determining the optimal schedule given an arbitrary size vector $\mathbf{p}$ and capacities $\mathbf{r}$ is NP-hard. Because of this and the apparent complexity of computing $E(\mathbf{p})$ for unconstrained $\mathbf{p}$, we will restrict ourselves (except for Section 5.5) to optimizing over size vectors $\mathbf{p}$ that give rise to what we call *uniformly dense*, or simply *uniform*, schedule constellations $\mathcal{S}(\mathbf{p})$. The basic property of a uniformly dense constellation is that it has completely regular geometry, much like the constellations in Figures 5-1 and 5-2. We define Γ as the set of all vectors $\mathbf{p}$ such that $\mathcal{S}(\mathbf{p})$ is a uniformly dense constellation in $\mathbb{S}_N$. In general, Γ is a subset of $\mathbb{S}_M$. It will also become clear that Γ is a finite and countable subset of $\mathbb{S}_M$, which allows the continuous optimization problem in Equation (5.2) to be restated as a discrete optimization problem when we restrict ourselves to only looking at size vectors $\mathbf{p}$ in the set Γ. This discrete optimization problem can then be solved using a very simple algorithm. Finally, even though we constrain the optimization problem, we show that we are still able to attain a minimum error $E(\mathbf{p})$ that decreases exponentially with M for any fixed N (the error associated with identically-sized resource pieces decreases with $1/M$).

A uniformly dense constellation in our scheduling context is defined to be a collection of all possible schedules $\mathbf{s}$ that satisfy the following conditions: each component of the vector $\mathbf{s}$ is a nonnegative integral multiple of some value β, and the components of $\mathbf{s}$ sum to 1. An important property of any uniformly dense constellation is that, no matter what vector $\mathbf{p}$ of length M was used to generate it, we know that there is some $\mathbf{p}'$ that consists of $M' = 1/\beta$ equally-sized elements and yields the same constellation. For example, a vector $\mathbf{p}'$ with all ($M' = 40$) elements equal to 1/40 yields the same constellation as the vector $\mathbf{p}_1$ in Figure 5-2. We call this the *emulation* property of the variable-sized pieces that create a uniformly dense constellation. We show in Section 5.4 that a set of M variable-sized pieces can generate a uniformly dense constellation requiring a larger number M' of equally-sized pieces when M is larger than N. We investigate the practical implications of this property further in Chapter 6.

One choice of $\mathbf{p}$ that produces a uniformly dense constellation is $p_i = 1/M$ (Figure 5-1), but there are others (Figure 5-2). A uniformly dense constellation has a single parameter, $E(\mathbf{p})$, which determines the density of the constellation. In Section 5.4 we discuss these uniformly dense constellations in more detail and give an algorithm that produces size vectors of length M that have maximally dense (i.e. minimal $E(\mathbf{p})$) uniform constellations. We will denote any vector that yields a maximally dense uniform constellation by $\mathbf{p}^+$.

It is not necessarily true that a vector $\mathbf{p}^+$ producing a maximally dense uniform constellation for a given M and N has the same $E(\cdot)$ as the optimal size vector $\mathbf{p}^*$ that solves Equation (5.2). In Section 5.5 we illustrate that for a specific choice of M and N and $\mathbf{p}'$, we can create a constellation that is not uniformly dense but has a smaller maximum error $E(\mathbf{p}')$ than that of the maximally dense uniform constellation, $E(\mathbf{p}^+)$ for $\mathbf{p}^+$ of length M.

The other important issue that makes uniformly dense constellations attractive is relative ease of implementation. If we are actually to implement a system that takes as input some vector $\mathbf{r}$

and outputs the optimal schedule (closest point in the constellation), along with the corresponding optimal allocation, we would like a simple algorithm to compute the optimal schedule. For the constellation in Figure 5-4, finding the optimal schedule and allocation is difficult. However, in Section 5.4.4 we show that if the $\mathbf{p}$ we use generates a uniformly dense constellation, then there is a very simple algorithm to compute the optimal schedule and allocation. In general, we are not guaranteed that there is a simple (or even polynomial-time) algorithm to calculate the optimal schedule unless the size vector $\mathbf{p}$ yields a uniformly dense constellation. Outside of this set of size vectors, solving this problem is an NP-complete problem.

5.3 Equally-Sized Pieces

Before we discuss uniformly dense constellations any further, we will look at a particular choice of $\mathbf{p}$ to gain some insight into the general problem and to develop some basic results that are useful in the subsequent discussion.

We start by considering the case when the resource is divided into M equally-sized pieces of size $1/M$. We call this set of sizes $\mathbf{p}_0 = (1/M, \ldots, 1/M)$. With these pieces, we are able to achieve any schedule $\mathbf{s}_\mathbf{A}(\cdot)$ with a granularity of $1/M$. In other words, $\mathbf{p}_0$ yields a uniformly dense constellation. We can visualize these schedules by picturing the corresponding constellation in the region $\mathbb{S}_N$. The set of achievable schedules $\mathcal{S}(\mathbf{p}_0) = \{\mathbf{s} | \mathbf{s} = \mathbf{s}_\mathbf{A}(\mathbf{p}_0), \mathbf{A} \in \mathcal{A}\}$ is a constellation of points within $\mathbb{S}_N$. We will use the notation $\mathcal{S} = \mathcal{S}(\mathbf{p}_0)$ when we are dealing with equally-sized pieces. Figure 5-1 shows this constellation when $N = 3$, and $M = 8$. Looking at this figure, we can see that no matter where $\mathbf{r}$ is within $\mathbb{S}_N$, it is close to a point in the constellation — there are no regions of the constellation where schedules are more or less dense than in other regions. We mentioned before that the computation of $E(\mathbf{p})$ is easier for equally-sized pieces, and this comes from the regularity of the constellation and the uniformity of the size vector $\mathbf{p}_0$. The following theorem establishes $E(\mathbf{p}_0) \le 1/M$ for all choices of M and N, and that $\lim_{N\to\infty} E(\mathbf{p}_0) = 1/M$.

Theorem 5.1 (Maximum Error with Equally-Sized Pieces) *If* $\mathbf{p} = \mathbf{p}_0$ *and* $\mathbf{r} \in \mathbb{S}_N$, *then*

$$E(\mathbf{p}_0) = \frac{N-1}{N} \cdot \frac{1}{M} \, .$$

Proof: See Appendix B.1. ■

This theorem establishes the value of $E(\mathbf{p}_0)$ for equally-sized pieces. This basic result is used as a point of comparison in the next section when we discuss more general size vectors $\mathbf{p}$. This maximum error implies that any bin will be over- or underfilled by at most $E(\mathbf{p}_0)$ for any N and M.

It might help to return to the change-making analogy presented earlier. Theorem 5.1 says that if you have M dollars to pay N people who collectively request M dollars, the most you would be forced to under- or over-pay any one person is $(N-1)/N$ dollars. The next section addresses the problem of how to choose the denominations of my bills so that you can pay more than M dollars with M bills but achieve the same payment guarantee. For example, if $N = M = 3$, you could use bills with the denominations 1, 1, and 2 dollars to pay off a total of 4 dollars to 3 people while still guaranteeing that you only under- or over-pay any person by at most 2/3 of a dollar. The factor of $(N-1)/N$ in $E(\mathbf{p}_0)$ comes from the geometry of the higher dimensions. We can gain some feel for it by considering what can happen if you need to give a single N-dollar bill ($M = 1$) to one of N people. If each person wants a single dollar, you will underpay $N-1$ people by a single dollar,

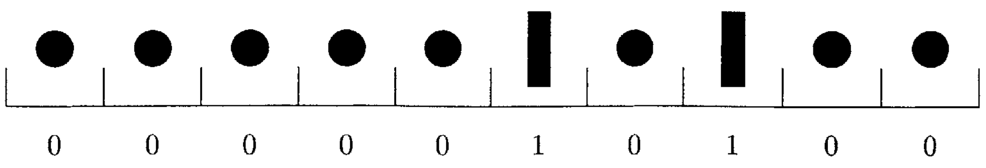


Figure 5-5: Illustration of pieces and dividers for $M = 8$, $N = 3$.

but overpay one person by $N-1$ dollars. you thus overpay one person by a fraction $(N-1)/N$ of the entire amount you have to pay out.

In order to move to discussing uniformly dense constellations in more detail, we need to establish one more fact for equally-sized pieces.

Lemma 5.1 (Number of Schedules using Equally-Sized Pieces) *If we have M equally-sized pieces, to be allocated among N bins, the number of distinct schedules is*

$$R(M,N) = \binom{M+N-1}{N-1}.$$

Proof: Consider a sequence of $M+N-1$ slots. In each slot we can put either one of the M pieces, denoted by '0', or one of $N-1$ dividers, denoted by '1'. The dividers indicate where the amount placed in one bin ends and the next begins. The number of pieces between the $(m-1)$-th and m-th dividers is the number of pieces allocated to bin m. Figure 5-5 illustrates this for a simple example. In the figure, bin 1 receives 5 pieces, bin 2 gets 1 piece, and bin 3 gets 2 pieces after we allocate the two dividers in slots 7 and 9. It is easy to see from this that we can get all possible distinct schedules of the 8 pieces to the 3 bins by choosing the positions of the dividers appropriately. In the case of general M and N, the number of possible ways of placing the $N-1$ dividers and the M pieces in the $M+N-1$ slots is simply the number of possible ways of choosing $N-1$ slots out of $M+N-1$ total slots, or $R(M,N)$. ■

This lemma will help us later on when we compare uniformly dense constellations generated by variable-sized pieces with those generated by equally-sized pieces.

Consider Figure 5-2 again. Although we have not discussed the design of uniformly dense constellations, it is clear that $\mathbf{p}_1$ yields a uniformly dense constellation. In fact, $\mathbf{p}_1$ yields the densest uniform constellation over all vectors $\mathbf{p}$ of length 8 for $N = 3$. The next few sections discuss the problem of how to design such maximally dense uniform constellations. In order to do this, we ask in Section 5.4 how to choose the piece sizes so as to increase the density of schedules while not changing the other parameters of the problem (M, N) or losing the regularity of the constellation.

5.4 Design of Uniformly Dense Constellations

As was mentioned before, the optimization problem in Equation (5.2) is not easy to solve in general. We concentrate in this section on a constrained version of the problem in which the size vectors $\mathbf{p}$ that we are allowed to choose are the ones that give rise to uniformly dense constellations. In Section 5.5, we show an example of a nonuniform constellation that performs better with respect to the measure of optimality we have chosen; however, nonuniform constellations in general require a more complicated algorithm to find the closest point in the constellation to a given point $\mathbf{r} \in \mathbb{S}_N$, and so are less practical to implement.

More specifically, in this section the three main results are Theorems 5.2, 5.3, and 5.4. They show, respectively, that: (i) we can easily compute the size vector yielding the maximally dense uniform constellation, i.e., the uniformly dense constellation with the smallest $E(\cdot)$ for given N and M; (ii) we can define a simple algorithm to generate any vector $\mathbf{p}$ yielding a uniformly dense constellation; and (iii) we can compute the optimal schedule and allocation for a given point $\mathbf{r} \in \mathbb{S}_N$ and vector $\mathbf{p}$ which yields a uniformly dense constellation using a simple algorithm. Section 5.4.2 describes an algorithm that returns a set of sizes of M pieces that yield a uniformly dense constellation, and thus can be used to emulate a set of $M' \geq M$ equally-sized pieces. Section 5.4.3 gives a complete characterization of the set Γ that comprises all possible vectors $\mathbf{p}$ yielding uniformly dense constellations. This characterization is accomplished by a generalization of the algorithm of Section 5.4.2. Section 5.4.4 shows that if the pieces are allocated from the largest piece to the smallest piece, a simple greedy algorithm can be used to determine the allocation that results in the closest schedule to any given $\mathbf{r}$.

5.4.1 Maximally Dense Uniform Constellations

We now define what we mean by a vector $\mathbf{p}^+$ that yields a maximally dense uniform constellation. The constrained version of Equation (5.2) considered now is

$$\mathbf{p}^+ = \arg\min_{\mathbf{p}\in\Gamma} E(\mathbf{p}) \ . \tag{5.3}$$

The solution to this, $\mathbf{p}^+$, is said to yield a maximally dense uniform constellation. This means that there is no other vector $\mathbf{p}$ that yields a uniformly dense constellation with a smaller value of $E(\mathbf{p})$ than $E(\mathbf{p}^+)$. The results in this section (see Lemma 5.4 and Theorem 5.3) allow us to characterize the set Γ completely for given M, N, as well as to pick $\mathbf{p}^+$ out of the set. It is important to realize that we are solving this problem for fixed values of M and N, so that the vector $\mathbf{p}^+$ that we find only applies for the specified values of M and N. However, the simple iterative structure of the algorithm that is used to generate $\mathbf{p}^+$ tells us that the M elements of $\mathbf{p}_M^+$ for M pieces are equal to the first M elements of $\mathbf{p}_K^+$ for K pieces where $K \geq M$. There is no simple correspondence between vectors $\mathbf{p}^+$ for different values of N. However, it is easy to show (though not proved here) that a size vector $\mathbf{p}^+$ designed for some N also yields a uniformly dense constellation for all smaller values of N (follows from Lemma 5.4 and Section 5.4.3).

Our definition of a uniformly dense constellation is equivalent to saying that the constellation can be generated using some number of equally-sized pieces. If we know the equivalent number of equally-sized pieces (say M') that are required to emulate a size vector $\mathbf{p}$ then Lemma 5.1 can be used to determine the number of schedules in $\mathbf{p}$'s constellation. We illustrate this with two examples.

If we consider the example of Figure 5-5 again, we can see that we can reach the same schedule if we use 5 pieces with sizes $(1/8, 1/8, 1/8, 2/8, 3/8)$ instead of 8 pieces with size $1/8$. A possible schedule using these 5 pieces is shown in Figure 5-6. Upon further inspection, it becomes clear that these five pieces and three bins can be used to achieve precisely the set of schedules that is possible with eight identical pieces and three bins. Thus 5 variable-sized pieces can be used to emulate a larger number (8) of equally-sized pieces. So, by using 5 variable-sized pieces instead of 5 equally-sized pieces, we have increased the number of distinct schedules in the constellation from $R(5,3) = 21$ to $R(8,3) = 45$.

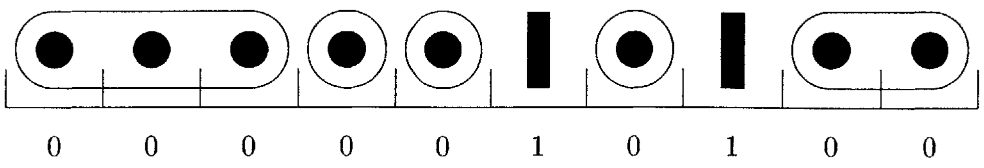


Figure 5-6: Illustration of slots and dividers for $N = 3$, using pieces with sizes $(1, 1, 1, 2, 3)$. The ovals surrounding the black dots indicate the actual pieces.

We can also consider the constellations in Figures 5-1 and 5-2. By using variable-sized pieces ($\mathbf{p}_1$), we have increased the number of distinct schedules from $|\mathcal{S}(\mathbf{p}_0)| = R(8, 3) = 45$ to $|\mathcal{S}(\mathbf{p}_1)| = R(40, 3) = 861$, and decreased the maximum error from $E(\mathbf{p}_0) = 1/12$ to $E(\mathbf{p}_1) = 1/60$. This decrease in error occurs for all values of N and M where $M > N$. In fact, the ratio between $E(\mathbf{p}^+)$ and $E(\mathbf{p}_0)$ for a given N grows exponentially as M increases (see Section 5.4.2).

The problem we look at next is how to find the size vector $\mathbf{p}^+$ that maximizes the density of the resulting constellation $\mathcal{S}(\mathbf{p}^+)$, or in other words, the size vector that allows M pieces to emulate the largest possible number of equally-sized pieces.

5.4.2 Generating Uniform Constellations

In this section, we present an algorithm that generates the maximally dense uniform constellation for given M and N. Before we get to the actual algorithm, we will establish a few restrictions that can be made on the size vectors and properties of the vectors that will motivate the algorithm and simplify the proof of the maximality of the resulting maximal size vector.

We first establish the property that only vectors $\mathbf{p}$ with rational elements can generate uniformly dense constellations, so that we may assume from now on that all vectors $\mathbf{p}$ are rationally-valued.

Lemma 5.2 (Rationality of p) *For any vector* $\mathbf{p}$ *such that* $\sum_i p_i = 1$*, all of the elements* p_i *must be rational if* $\mathbf{p}$ *generates a uniformly dense constellation.*
Proof: Assume without loss of generality that p_1 is irrational. Consider the schedule that puts p_1 in the first bin, and all the other elements of $\mathbf{p}$ in the second bin, so the second bin contains $1 - p_1$. There are no integers k and ℓ such that

$$\frac{1 - p_1}{p_1} = \frac{k\beta}{\ell\beta} = \frac{k}{\ell},$$

otherwise p_1 would be rational ($= \ell/(\ell + k)$) and so would $1 - p_1$. However, by the definition of a uniformly dense constellation, every schedule must consist of elements that are integer multiples of some common factor, so the above schedule is evidently not part of a uniformly dense constellation. Thus, any $\mathbf{p}$ with an irrational element can produce a schedule that is not in any uniformly dense constellation. ■

Given that we are only considering rationally-valued size vectors, we can define a scaled version of any vector $\mathbf{p}$ that has only integer components. We call this vector $\mathbf{P} = d \cdot \mathbf{p}$, where d is the greatest common denominator of the elements of $\mathbf{p}$; there is a one-to-one correspondence between $\mathbf{p}$ and $\mathbf{P}$. Working in terms of $\mathbf{P}$ at times will make the statement of the algorithm and the proof of its optimality simpler. We will use the notation

$$\bar{\mathbf{P}}(i) = \sum_{j=1}^{i} P_j$$

in our discussions below, and when necessary

$$\bar{\mathbf{P}}(i, N) = \sum_{j=1}^{i} P_j$$

if we need to make the number of bins, N, explicit.

We will next assume that the elements of $\mathbf{p}$ (and thus $\mathbf{P}$) are sorted, with $p_1 \leq p_2 \leq \cdots \leq p_M$ ($P_1 \leq P_2 \leq \cdots \leq P_M$). This does not restrict the possibilities, because any vector $\mathbf{p}$ produces the same constellation as the sorted version of the vector.

We use the terminology "$\mathbf{P}$ corresponds to a uniformly dense constellation" to mean that $\mathbf{p} = \mathbf{P}/\sum_j P_j$ has a uniformly dense constellation $\mathcal{S}(\mathbf{p})$.

Lemma 5.3 *For any sequence* $\mathbf{P}$ *that corresponds to a uniformly dense constellation,* P_1 *is a common divisor of all* P_i.
Proof: We show this by considering $\mathbf{p} = P/\sum_i P_i$. By the definition of a uniformly dense constellation, every schedule $\mathbf{s}$ that can be generated by some allocation of $\mathbf{p}$ must have elements $s_i = \nu_i \beta$ for integers ν_i and some constant β. The smallest that s_i can be is p_1 and because the constellation must be uniformly dense, we know that $\beta = p_1$. So, we can write $\mathbf{s} = \nu p_1$ for any allocation $\mathbf{s}$ and some integer vector ν. However, any allocation of the elements p_i to a schedule $\mathbf{s}$ must be a point in the constellation. Specifically, the allocation $\mathbf{s}(j)$ that puts element p_j in $s_1(j)$ and the other elements in $s_i(j)$, $i > 1$ must produce a vector $\mathbf{s}(j) = \nu(j)p_1$, which means that $s_1(j) = p_j = \nu_1(j)p_1$ for integer ν_1. This implies that p_j is a multiple of p_1 if $\mathbf{p}$ generates a uniformly dense constellation.

If every element of $\mathbf{p}$ is an integer multiple of p_1, then when we multiply $\mathbf{p}$ by $\sum_i P_i$ to get the integer sequence $\mathbf{P}$, every element of $\mathbf{P}$ must be an integer multiple of P_1. ∎

Lemma 5.3 lets us consider only sequences with the smallest element $P_1 = 1$, because we know that there is no other common factor across all of the elements of $\mathbf{P}$. Lemma 5.3 also tells us that $p_1 = \beta$ for any $\mathbf{p}$ that yields a uniformly dense constellation. Thus, we now have a method for comparing vectors with uniformly dense constellations. If a size vector $\mathbf{p}$ yields a uniformly dense constellation and thus can emulate $M' = 1/\beta$ equally-sized resource pieces, then the maximum error is

$$E(\mathbf{p}) = \frac{N-1}{N} \cdot \frac{1}{M'} = \frac{N-1}{N} \cdot \beta = \frac{N-1}{N} \cdot p_1 \ ,$$

and the size vector $\mathbf{p}^+$ with the smallest β yields the maximally dense uniform constellation. This is important because it implies that we can determine $\mathbf{p}^+$ if we can determine the vector $\mathbf{P}^+$ that corresponds to a uniformly dense constellation and has the largest sum, $\bar{\mathbf{P}}^+(M)$, of all possible vectors that correspond to uniformly dense constellations, for given M and N.

We now describe an algorithm that will generate the size vector that yields the provably densest uniformly dense constellation using M variable-sized pieces and N bins. The following algorithm first produces integral piece sizes that result in a scaled version of the maximally dense uniform constellation. Then $\mathbf{p}^+$ results from scaling these sizes so that their sum is equal to 1.

Algorithm 5.1 (Maximally Dense Uniform Schedule Constellations) *To generate the size vector* $\mathbf{p}^+$ *that has the maximally dense uniform constellation, given* N *and* M *(* $\mathbf{P}^+$ *below is an integer vector of length* M*):*

1. *Set* $P_1^+ = 1$.
2. *Set* $P_{i+1}^+ = f(\bar{\mathbf{P}}^+(i)) = \left\lceil \frac{1+\bar{\mathbf{P}}^+(i)}{N-1} \right\rceil$ *for* $i = 1, \ldots, M-1$.
3. *Set* $p_i^+ = \frac{P_i^+}{\sum_{j=1}^M P_j^+}$ *for* $i = 1, \ldots, M$.

This algorithm is interesting for a few reasons. First, it is very simple to perform the calculations. Each new element of $\mathbf{P}^+$ depends only on the sum of the previous elements. Second, as we shall see, this simple dependence allows the algorithm to be applied to any existing vector $\mathbf{p}$ that has a uniformly dense constellation in order to extend $\mathbf{p}$ to larger values of M. Third, as we show in Section 5.4.3, if we were to generalize the algorithm slightly (see Algorithm 5.2) so that the iterative step chooses some value of P_{i+1} such that $P_i \leq P_{i+1} \leq f(\bar{\mathbf{P}}(i))$, it would still generate a vector $\mathbf{p}$ that has a uniformly dense constellation, and in fact this generalized algorithm is able to generate all possible size vectors that have a uniformly dense constellation, i.e. all vectors in the set Γ defined earlier.

We prove that this algorithm computes the vector $\mathbf{p}^+$ as a function of M and N using the following lemmas and theorem. First, in Lemma 5.4 we will show that the integral sequence $\mathbf{P}^+$ produced in the algorithm corresponds to a uniformly dense constellation. We then show in Lemma 5.5 that $\mathbf{P}^+$ is a nondecreasing sequence of integers that is a component-wise upper bound on any other nondecreasing sequence $\mathbf{P}$ of integers corresponding to a uniformly dense constellation. In other words, if $\mathbf{p} = \mathbf{P}/\sum_i P_i$ has a uniformly dense constellation $\mathcal{S}(\mathbf{p})$, then $P_i \leq P_i^+$ for $i = 1, \ldots, M$. Finally, using these two results, we show that $\mathbf{p}^+$ has the densest uniform constellation among all other normalized sequences that have uniform constellations.

We first present a property that can be used to test whether a sequence $\mathbf{P}$ corresponds to a uniformly dense constellation.

Lemma 5.4 (Uniformly Dense Test) *The sequence* $\mathbf{P}$ *corresponds to a uniformly dense constellation if and only if* $P_k \leq P_{k+1} \leq f(\bar{\mathbf{P}}(k))$ *for all* $k = 1, \ldots, M-1$.
Proof: See Appendix B.2. ■

It follows from this lemma that $\mathbf{P}^+$ corresponds to a uniformly dense constellation, because it satisfiesthe condition in the statement of the lemma. We have shown that Algorithm 5.1 produces a sequence $\mathbf{p}^+$ that has a uniformly dense constellation. We now assert that the corresponding sequence $\mathbf{P}^+$ is an upper bound on all integral sequences whose smallest element is 1 and that correspond to uniformly dense constellations.

Lemma 5.5 *The sequence* $\mathbf{P}^+$ *of length* M *that is produced by the previous algorithm is an upper bound (element by element) on any other integral nondecreasing sequence* $\mathbf{P}$ *that corresponds to a uniformly dense constellation.*
Proof: See Appendix B.2. ■

We now present the theorem that we have been building to.

Theorem 5.2 (Maximality of $\mathbf{p}^+$) *The nondecreasing sequence* $\mathbf{p}^+$ *has the maximally dense uniform constellation over all possible nondecreasing sequences* $\mathbf{p}$ *of length* M *that have uniformly dense constellations and such that* $\sum_i p_i = 1$ *for given* N.
Proof: Lemma 5.4 tells us that $\mathbf{P}^+$ corresponds to a uniformly dense constellation, and Lemma 5.5

tells us that $\mathbf{P}^+$ is an upper bound on any length M sequence $\mathbf{P}$ that corresponds to a uniformly dense constellation such that $P_1 = 1$. Thus, $\bar{\mathbf{P}}^+(M) \geq \bar{\mathbf{P}}(M)$ for any $\mathbf{P}$ that corresponds with a uniformly dense constellation. Because $\bar{\mathbf{P}}^+(M) = M' = 1/\beta$, $\mathbf{p}^+$ has a smaller β than any other $\mathbf{p}$ that yields a uniformly dense constellation, and thus the smallest maximum error. ∎

This result is particularly interesting due to the simplicity of the algorithm that is used to compute the vector $\mathbf{p}^+$. Each element of $\mathbf{p}^+$ is generated based only on the sum of the previous elements; this also tells us what the relationship is between vectors $\mathbf{p}^+$ of differing lengths M.

A useful property of the maximal sequence $\mathbf{p}^+$ is that the increase in density over the sequence $\mathbf{p}_0$ is exponential in the number of resource pieces, M. The integers that are produced for large M and fixed N are

$$P_{i+1}^+ = \left\lceil \frac{1 + \sum_{j=1}^{i} P_j^+}{N-1} \right\rceil > \frac{\sum_{j=1}^{i} P_j^+}{N-1},$$

and so the sum

$$\begin{aligned} \bar{\mathbf{P}}^+(i+1) &> \frac{N}{N-1} \cdot \bar{\mathbf{P}}^+(i) \\ &= \left(\frac{N}{N-1}\right)^i P_1 = \left(\frac{N}{N-1}\right)^i . \end{aligned}$$

Since we know that $p_1 = 1/M' = \beta$, then $1/\beta = M' = \bar{\mathbf{P}}^+(M)$ for $\mathbf{p}^+$ is greater than $(N/(N-1))^{M-1}$. And so we have the following bound for $E(\mathbf{p}^+)$.

$$E(\mathbf{p}^+) = \frac{N-1}{N} \cdot \frac{1}{M'} < \frac{N-1}{N} \cdot \left(\frac{N-1}{N}\right)^{M-1}$$

where $M' = \bar{\mathbf{P}}^+(M)$. Compare this with the maximum error for a sequence of equally-sized pieces, $\mathbf{p}_0$,

$$E(\mathbf{p}_0) = \frac{N-1}{N} \cdot \frac{1}{M} .$$

For M larger than N we have an increase in density (a decrease in maximum error) by a factor exponential in M. This means that the density of schedules grows exponentially with M when we choose the piece sizes properly. Because the constellation is uniformly dense, the possible schedules that result are equivalent to the ones that would be possible if we were to use M' equally-sized pieces. We will see some concrete examples of this in Chapter 6. Also, as we will show in Section 5.4.4, the complexity of finding the optimal schedule and allocation is very low.

5.4.3 Characterizing Γ

We now present a slight modification of Algorithm 5.1 that can compute all possible size vectors $\mathbf{p}$ yielding uniformly dense constellations for given M and N.

Algorithm 5.2 (Uniformly Dense Schedule Constellations) *To generate a size vector* $\mathbf{p}$ *that has a uniformly dense constellation, for given* N *and* M*:*

1. *Set* $P_1 = 1$.
2. *Set* P_{i+1} *such that* $P_i \leq P_{i+1} \leq f(\bar{\mathbf{P}}(i)) = \left\lceil \frac{1 + \bar{\mathbf{P}}(i)}{N-1} \right\rceil$, *for* $i = 1, \ldots, M-1$.

3. Set $p_i = \dfrac{P_i}{\sum_j P_j}$.

As we argued before, we only need to worry about nondecreasing vectors because any other vector yields a constellation equivalent to that of the sorted (and thus nondecreasing) version of the vector. Thus, the fact that the algorithm must output a nondecreasing vector by construction does not limit the constellations that will result.

Theorem 5.3 (Characterization of Γ) *A nondecreasing size vector* $\mathbf{p}$ *is in the set* Γ *if and only if it is a possible output of Algorithm 5.2.*
Proof: It is obvious that the vector $\mathbf{p}$ output by Algorithm 5.2 satisfies the property in Lemma 5.4, and so yields a uniformly dense constellation. Because Γ is simply the set of vectors with uniformly dense constellations, we have proved the theorem. ■

We will use $\Gamma(M, N)$ to denote the vectors $\mathbf{p} \in \Gamma$ such that the length of $\mathbf{p}$ is equal to M, and it is a possible output of Algorithm 5.2 using N bins. Being able to characterize Γ in this way not only allows us to find the vector that has the maximally dense uniform constellation, but allows us to test whether or not a given vector has a uniformly dense constellation. By simply performing the test in the statement of Lemma 5.4 for each increment in the sorted vector, we can tell whether or not a vector has a uniformly dense constellation. Algorithm 5.2 also provides a way to generate vectors with uniformly dense constellations when there are other constraints on the elements of the vector. If we consider that the last element of the vector $\mathbf{p}^+$ grows exponentially with M, we may have problems practically implementing a system with such widely varying piece sizes. For example, view the pieces of the resource as individual compute servers, then our optimization problem becomes how to choose the speeds of the servers so that we can provide the finest-grain allocation to the users. It may be infeasible to choose such a wide range of server speeds, or certain speeds may not be available. In this case, we can just choose the largest speed available at each step in the algorithm, instead of using the largest possible increment.

5.4.4 Computing the Optimal Allocation and Schedule

The previous results tell us that we can create any schedule of $\bar{\mathbf{P}}(M)$ equally-sized pieces among N bins by using only M variable-sized pieces with sizes corresponding to the elements of $\mathbf{P}$. Another way to interpret this result is that we can schedule $\bar{\mathbf{P}}(M)$ equally-sized pieces using only M decisions, where each decision is made to allocate a subset of the pieces to some bin. We now present a simple algorithm that will compute the optimal allocation in this way.

If we only wanted to determine the optimal schedule for a given vector $\mathbf{r}$, we can simply use the method in Theorem 5.1 to construct the schedule closest to a given $\mathbf{r}$ by choosing I to contain the indices that correspond with the $K = \sum_i r_i - \lfloor r_i \rfloor$ largest elements of $\mathbf{r} - \lfloor \mathbf{r} \rfloor$ [11, p. 447]. This method requires $O(NK)$ steps. However, if we want to also find an allocation of the pieces that results in the optimal schedule, we need to do a little more work. The following algorithm computes the optimal schedule and algorithm in $O(NM)$ steps. At each of M stages it simply needs to find the largest element of a vector of length N, and perform a few minor operations.

The basic idea of the algorithm is that at each round, the largest remaining piece is smaller than the largest difference between the optimal schedule and the current schedule (based only on the allocations made in previous rounds), and so we can simply allocate the largest remaining piece to the user with the largest difference between the optimal schedule and the current schedule. At each step, we know that every element of the current schedule is no greater than the optimal schedule, and after the last round the total sums of both schedules are equal, implying that the two schedules are equal.

Algorithm 5.3 (Variable-Sized Scheduling Algorithm (VSSA)) *The following algorithm computes an optimal allocation* $\mathbf{A}$ *of* M *pieces with sizes* $\mathbf{p} \in \Gamma$ *given a set of bins with capacities* $\mathbf{r}$. *It also computes the optimal schedule* $\mathbf{s}$.

1. *Begin with the schedule* $\mathbf{s}$ *and allocation* $\mathbf{A}$ *initialized to zero and define working quantities* $v_k = r_k$.
2. *Initialize* $m = 1$, *and while* $\sum_k s_k < 1$ *repeat the following:*
   (a) *Choose* $k^* \in \arg\max_k v_k$.
   (b) *Set* $v_{k^*} = v_{k^*} - p_{M-m+1}$.
   (c) *Set* $s_{k^*} = s_{k^*} + p_{M-m+1}$.
   (d) *Set* $A_{(M-m+1),k^*} = 1$.
   (e) *Increment* m *by* 1.

Theorem 5.4 (Optimality of VSSA) *The VSS algorithm computes a schedule* $\mathbf{s}$ *such that* $\| \mathbf{s} - \mathbf{r} \|_\infty$ *is minimized for any* $\mathbf{r}$ *such that* $\sum_i r_i = 1$.
Proof: See Appendix B.2. ■

This scheduling algorithm gives us a way of practically determining the optimal schedule and allocation given a set of bin capacities. This means that both designing and using optimal size vectors $\mathbf{p}^+$ in practical systems is easily accomplished.

5.5 Nonuniform Constellations

Section 5.4 describes how to find the vector $\mathbf{p}^+$ that has a maximally dense uniform constellation as the vector that solves Equation (5.3), and thus is the best choice among uniformly dense constellations if we want to minimize $E(\mathbf{p})$. There are also other choices of error measures that would make sense other than $E(\mathbf{p})$, such as the average error

$$e_M(\mathbf{p}) = \frac{1}{|\mathbb{S}_N|} \cdot \int_{\mathbb{S}_N} e(\mathbf{p}, \mathbf{r}) d\mathbf{r} . \tag{5.4}$$

We could also use another metric besides $e(\cdot)$ to measure the distance between $\mathbf{r}$ and $\mathcal{S}(\mathbf{p})$. It is clear that as the density of a uniformly dense constellation increases, many of these reasonable error measures will decrease, keeping the maximally dense uniform constellation as the best choice among uniformly dense constellations.

However, the vector $\mathbf{p}^+$ is not necessarily equal to $\mathbf{p}^*$. It turns out that we can easily come up with other vectors $\mathbf{p}$ with nonuniform constellations with equal or smaller measures of error (either maximum error, or as we show first below, average error). We also give an example of a vector $\mathbf{p}$ that, although we do not show that it is equal to $\mathbf{p}^*$, outperforms $\mathbf{p}^+$ with respect to the optimization problem in Equation (5.2).

We first present an example that shows that even a simple modification to the vector $\mathbf{p}^+$ can result in a new vector that outperforms $\mathbf{p}^+$ with respect to the average error. Look at the two constellations in Figures 5-2 and 5-3. The uniform constellation in Figure 5-2 uses pieces with sizes

$$\mathbf{p}_1 = \mathbf{p}^+ = (1/40, 1/40, 2/40, 3/40, 4/40, 6/40, 9/40, 14/40)$$

and is the maximally dense uniform constellation as determined using Algorithm 5.1, and the constellation in Figure 5-3 uses pieces with sizes

$$\mathbf{p}_2 = (1/60, 2/60, 3/60, 4/60, 6/60, 9/60, 14/60, 21/60) \ .$$

We will use M_1' to denote the number of equally-sized pieces needed to emulate $\mathbf{p}_1$. By Theorem 5.1, we know that $E(\mathbf{p}_1) = (N-1)/NM_1' = 2/(3 \cdot 40) = 1/60$. We define M_2' as the number of equally-sized pieces that would be needed to generate a uniformly dense constellation equal to the constellation in Figure 5-3 with the holes filled in to make the geometry completely regular. Looking closely at the nonuniform constellation in Figure 5-3, we see that the only missing points are isolated from each other. It is easy to see that if we choose $\mathbf{r}$ to lie where one of the points is missing, the closest schedule will be distance $1/M_2' = 1/60$. If we try to maximize the error by putting $\mathbf{r}$ anywhere else, we will get $(N-1)/NM_2' = 1/90$, so $E(\mathbf{p}_2) = 1/60 = E(\mathbf{p}_1)$. However, it is obvious from the figure that the *average* error between an arbitrary $\mathbf{r}$ and the closest schedule is much smaller in the nonuniform constellation (there are only a few small areas in which it could be larger). We can get a rough idea of how much smaller by considering the number of points in the two constellations. The number of points in the uniform constellation is $N_1 = 861$ and the number of points in the nonuniform constellation is $N_2 = 1881$. This means there are more than twice as many points in the same area for the nonuniform constellation. If the nonuniform constellation had the few holes filled in, the average error would be proportional to the maximum error 1/90 (or equivalently, proportional to the density). Call this average error $\alpha/90$. The 10 holes in the constellation at most increase the average error for the 60 surrounding points to $\alpha/60$, so an approximate upper bound on the average error would be $\alpha(1/90 + (60/1881) \cdot (1/60)) \approx \alpha/85$. The average error in the uniform constellation would be approximately $\alpha/60$. Empirical studies for these two constellations show that the ratio of the average error of the nonuniform and the uniform constellations is approximately 0.67 (compared with the approximate upper bound of $60/85 \approx .7$). This indicates that $\mathbf{p}_1$ may not perform as well as other almost-regular nonuniform constellations under varying error measures.

We can even exhibit an example vector that outperforms $\mathbf{p}_1$ with respect to both the average and maximum error measures. If we use the vector $\mathbf{p}_3$ such that

$$\mathbf{p}_3 = (1/6277) \cdot (128, 164, 288, 432, 648, 972, 1458, 2187) \ ,$$

the maximum error is *smaller* than 1/60 as can be determined by an exhaustive examination of the constellation (shown in Figure 5-4). Empirically, the average error also appears to be about 1/2 the average error of the constellation of $\mathbf{p}_1$. Uniformly dense constellations do not have the smallest maximum error over all possible constellations, as the example above illustrates; there are other constellations that are much denser on average, and provide better worst-case bounds.

A drawback to the use of nonuniform constellations is that it is more complicated to determine the optimal schedule. Uniformly dense constellations can be scheduled using the simple algorithm given in Section 5.4.4, but the algorithm does not select the optimal schedule for arbitrary nonuniform constellations. In fact, as we show next, determining the optimal schedule given an arbitrary size vector $\mathbf{p}$ and capacities $\mathbf{r}$ is an NP-hard problem.

5.5.1 Difficulty of Finding the Optimal Schedule or Allocation for Nonuniform Constellations

It appears that the problem of determining the optimal schedule and allocation in a nonuniform constellation given an input vector $\mathbf{r}$ is more difficult than for a uniform constellation. In order to establish that, in fact, the problem of determining the optimal allocation is NP-hard, we show that any algorithm that computes the optimal allocation would also allow us to solve the set partition problem, which is NP-complete [12]. See [12] for background information.

The set partition problem is defined as follows. Given a set $\mathbf{A}$ of integers with sum A, does there exist a subset $\mathbf{A}'$ of $\mathbf{A}$ such that $\sum_i A'_i = (1/2) \cdot \sum_i A_i = A/2$? This question can be answered by an algorithm that computes the optimal allocation given a size vector $\mathbf{p}$ and capacities $\mathbf{r}$. All we have to do is set $p_i = A_i/A$ and $r_1 = r_2 = 1/2$ for $M = |\mathbf{A}|$ and $N = 2$. Then, our algorithm will compute the optimal allocation from which we can find the optimal schedule $\mathbf{s}$ by matrix multiplication. If $\mathbf{s} = \mathbf{r}$, then there is a partition of $\mathbf{A}$. Otherwise, there is no partition because $\mathbf{s}$ minimizes the maximum componentwise error between any schedule and $\mathbf{r}$. So, because the set partition problem is NP-hard, computing the optimal schedule must also be NP-hard.

The NP-hardness of the general allocation problem, and our ability to completely characterize the set Γ gives us a simple division of the space of possible size vectors $\mathbf{p}$. For all $\mathbf{p} \in \Gamma$, the determination of the optimal allocation and schedule is accomplished through a simple greedy algorithm. For all other $\mathbf{p} \notin \Gamma$, this determination is not as straightforward, and in the worst case requires the solution of an NP-hard decision problem.

The scheduling problem can be solved using a nondeterministic Turing machine by simply iterating through the resource pieces and trying all possible positions for each piece in parallel. Thus, the problem is NP-complete.

Even if this scheduling problem is NP-complete, are there practical situations in which a restricted version of the optimization problem can be solved exactly or approximately in polynomial time? For instance, if we only consider vectors $\mathbf{p}$ such that the maximum element of the corresponding $\mathbf{P}$ is bounded below some number B then the optimal allocation can be computed by a pseudo-polynomial algorithm [19], at least for $N = 2$. There might also be polynomial time approximation algorithms which allow us to find a schedule that is guaranteed to have an error close to the optimal error. We leave these questions for further investigation.

5.5.2 Simple Nonuniform Constellations

Although the general optimization problem with nonuniform constellations is hard to solve, it is possible to examine a few simple cases and solve directly for the best nonuniform constellation. These examples establish that $N = 2$, the uniform constellation has the smallest worst case error but for $N = 3$, uniform constellations do not necessarily have the best error characteristics.

Start by assuming that $N = 2$ and $M = 2$. In this case, we have 4 possible schedules: $(0, 1)$, $(\alpha, 1 - \alpha)$, $(1 - \alpha, \alpha)$, and $(1, 0)$. It is obvious that by choosing $\alpha = 1/3$ we will get the smallest maximum error; this is equivalent to the maximally dense uniform constellation.

Next, assume $N = 2$, and $M = 3$. Again, if we choose $\mathbf{p} = (1/7, 2/7, 4/7)$ we will get the smallest maximum error as well as the maximally dense uniform constellation. In fact, for $N = 2$ the maximally dense uniform constellation always gives the smallest maximum error for any M. This is because we are simply placing 2^M points on a line; placing them uniformly spaced gives us the smallest maximum distance between adjacent points and thus the smallest maximum error.

Now, assume $N = 3$, and $M = 2$. In this case, $\mathbf{p} = (1/2, 1/2)$ yields a uniformly dense constellation and choosing $\mathbf{p} = (1/3, 2/3)$ results in the same maximum error, $1/3$, but smaller

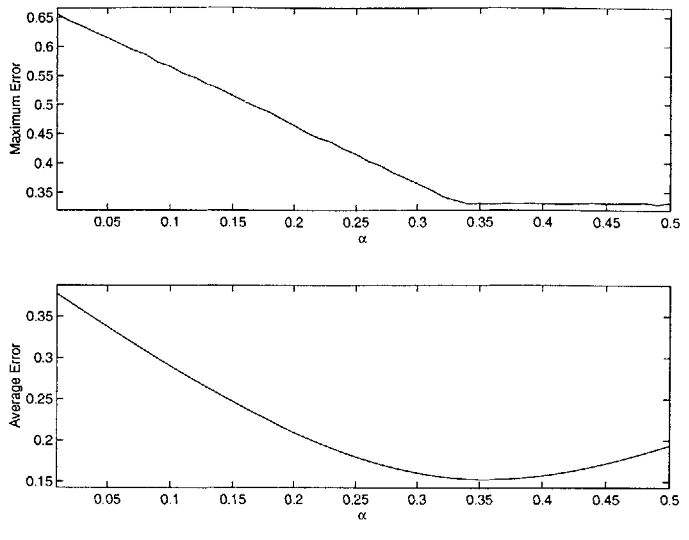


Figure 5-7: Average and maximum error for $N = 3$, $M = 2$ for a range of constellations. Approximately 5000 sample points were used to compute the error statistics. This translates to a bound of ± 0.02 on the maximum error.

average error with a nonuniform constellation. The average error turns out to be 0.16 as opposed to 0.19 for the maximally dense uniform constellation. In fact, empirical data shows that $\mathbf{p} = (0.35, 0.65)$ results in an even smaller average error for the same maximum error. A plot of the average and maximum error is given in Figure 5-7 for a fine sampling of all possible constellations. The constellations can be parameterized by a single value we will call α. A constellation corresponds to $\mathbf{p} = (\alpha, 1-\alpha)$ for $0 \le \alpha \le 0.5$. We have plotted error statistics for $\alpha = 0.01, 0.02, \ldots, 0.5$. The plots were obtained by sampling the entire space $\mathbb{S}_3$ evenly and then perturbing the sample points slightly on the plane. From the plot of average error, it is clear that the minimum error occurs near $\alpha = 0.35$.

If we repeat this for $N = 3$ and $M = 3$, we get similar results; the maximal uniformly dense constellation $\mathbf{p}^+ = (1/4, 1/4, 1/2)$ has maximum and average error measures of 1/6 and 0.1, respectively, but is outperformed by other size vectors, such as $\mathbf{p} = (1/6, 1/3, 1/2)$ which has a maximum error slightly smaller than that of $\mathbf{p}^+$ and an average error of approximately 0.077. These error measurements are reduced slightly at points near $\mathbf{p}$.

Continuing this process for larger N or M is computationally very difficult. The complexity of this process arises because the objective function, Equation (5.1), that we are trying to minimize is very difficult to evaluate. A straightforward approach as we have used involves generating large numbers of sample values of $\mathbf{r}$ and computing the worst-case error over all of the samples as a lower bound on the value of our objective function. Even just computing the worst-case error for a single sample in an arbitrary constellation is an NP-hard problem if the computation is done by first finding the closest point in the constellation (which is NP-hard), then computing the distance between the sample and the closest point.

5.6 Aside: Bits and Pieces

As we mentioned earlier, the general static allocation problem of Section 3.1 is equivalent to a vector quantization problem. The goal of a vector quantizer is to minimize the number of bits per sample needed to encode a stream of samples to a maximum level of error or distortion. In this chapter, we have the same goal, but we are restricted to dealing with what we might call a *building-block* quantizer; the output of the quantizer is formed by partitioning a set of building blocks among the elements of the output. A uniform scalar quantizer is the simplest building-block quantizer, where the output of the quantizer can be thought of as having been assembled from some subset of a number of building blocks called bits. If the input number is an integer between 0 and 1, then the blocks have sizes 2^{i-B} for $0 \leq i \leq B-1$ to give a resolution of 2^{-B}. A general vector quantizer will not have this kind of structure; the constellation points are chosen for their geometric properties directly. The problem we are solving here is how to determine the sizes of these building blocks in order to get the smallest distortion measured over all possible input values.

As an illustration of how this problem differs from a more traditional quantization problem, consider what happens when we compare the number of bits required to represent an allocation built out of these blocks with the number of bits needed to quantize the input directly using a uniform scalar quantizer for each element of the input. An allocation consists of a mapping of M resource pieces to N users. Each resource requires approximately $\log_2 N$ bits to specify the user it is allocated to, for a total of $M \log_2 N$ bits to represent the entire allocation. We compare the following situations using the resolution of the resulting quantizer, which is proportional to the worst-case quantization error. We have shown in this chapter that this number of bits gives us an upper bound on the resolution of

$$r_0(M) \approx ((N-1)/N)^{M-1}$$

if we choose the sizes of the blocks properly. If we instead want to encode the schedule directly with the same number of bits, we can gain a better resolution over a larger space of inputs by simply splitting the bits equally between the inputs. Given N inputs, we have $(M \log_2 N)/N$ bits per input. This leaves us with a resolution of

$$r_1(M) \approx 2^{-(M \log_2 N)/N} = N^{-M/N} .$$

This decreases faster than r_0 as N increases, and so gives better resolution for a given number of bits. Also, because we are splitting the bits evenly, we are accounting for a range of input values that allow each individual input to vary from 0 to 1 independently, instead of the restriction that the sum of the inputs be equal to 1. This allows us to represent inputs over a much larger range, and perhaps even increase the resolution further if we use a smarter bit allocation.

What if we keep the basic restriction that we have a limited number of building blocks? This implies that we will only have M blocks to split across the elements of the output. By giving M/N bits to each element, we can now achieve a resolution of

$$r_2(M) \approx 2^{-M/N} ,$$

for the direct encoding of the input. This is equivalent to assigning a group of blocks to be used only for a single input element, and using block sizes that are powers of 2 within each group. Because $(N-1)/N < (1/2)^{1/N}$, we now have that r_2 decreases more slowly than r_0 with M.

This still leaves us with the problem that the region of input vectors that can be represented is larger when we quantize each element independently. However, if we simply scale the sizes of the

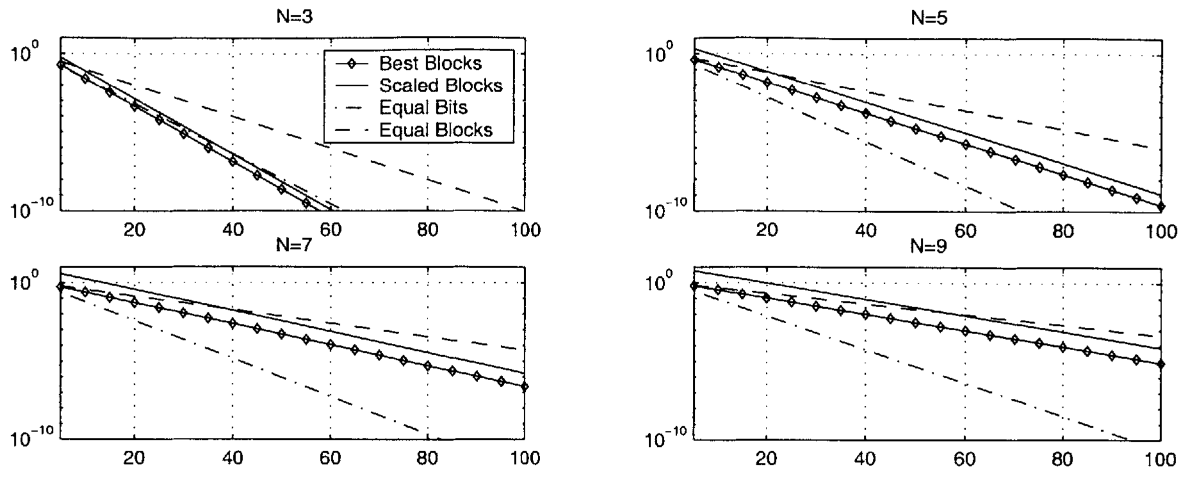


Figure 5-8: Comparison of resolution achieved for $N = 3, 5, 7, 9$ and $M = 5, \ldots, 100$.

original blocks by N, we make the resolution worse by a factor of N, but can now represent any N-dimensional input such that each element is between 0 and 1 independently. Additionally, the resolution r'_0 is still asymptotically better than r_2:

$$r'_0(M) \approx N \cdot ((N-1)/N)^{M-1} \ .$$

We show examples of the 4 resolution measures $r_0(M)$ ('Best Blocks'), $r'_0(M)$ ('Scaled Blocks'), $r_1(M)$ ('Equal Bits'), and $r_2(M)$ ('Equal Blocks') in Figure 5-8 for values $N = 3, 5, 7, 9$ and $M = 5, \ldots, 100$.

Chapter 6

Applications of the Variable-Sized Channel Results

We begin in Section 6.1 with a basic dynamic model which fits in our framework from Section 3.2, building on the scenario and results presented in Chapter 5. After a presentation of the problem and discussion of how it fits in with our model, we show how the results from Chapter 5 can be incorporated into this dynamic model and used to improve performance in a number of different ways in various applications in subsequent sections.

We present three representative applications of the previous results to scheduling problems arising in:

- distributed sensor communication;
- virtual grouping of channels in multichannel communication systems;
- batch queue servers.

All three use the same model for the dynamics of the system, which we described in Section 6.1. We then discuss in Section 6.2 how the results from the previous sections can decrease the buffering requirements in a specific shared-link distributed communication environment. In Section 6.3 we use the previous results to reduce the control complexity in a communication system with a large number of small identical channels. In Section 6.5, we show how to design a batch queueing system with multiple servers with differing service rates such that the system has a large throughput while never having any idle servers. A basic assumption we make here is that the resource pieces are strictly additive; i.e., if a user receives more than one piece, the set of pieces is just as useful as a single piece with the same total size. Finally, in Section 6.6 we discuss modifications of these applications in which failures of the resources can be handled through the use of sets of resource pieces that are robust to the removal of some number of pieces.

6.1 Dynamic Model

In order to illustrate how the earlier scheduling results can be applied to specific circumstances, we will first introduce a basic model of a dynamic scheduler. Consider a situation in which a resource comprising M interchangeable pieces (with sizes specified by a fixed vector $\mathbf{p}$) is to be repeatedly allocated to a set of N users during a sequence of allocation rounds. The desired fraction of the resource or *reservation* that the respective users want takes the place of the bin sizes $\bar{\mathbf{r}}$ in the generic problem from Section 5.1; we assume that $\bar{\mathbf{r}}$, like $\mathbf{p}$, is fixed. At the beginning of round n, a new allocation is chosen by a scheduler, and the users employ the allocated pieces for the duration of the round. The goal of the scheduler is ensure the schedule $\mathbf{s}[n]$ of pieces has a time average close to the desired schedule $\bar{\mathbf{r}}$. As in Section 3.2, we define the dynamics as

$$\ell[n+1] = \ell[n] + \bar{\mathbf{r}} - \mathbf{s}[n] \, , \tag{6.1}$$

and we define $\ell[0] = 0$. Note that $\ell[n]$ is allowed to have negative components, i.e. 'lead' components. We choose an allocation at round n that minimizes the predicted lag:

$$\begin{aligned} \mathbf{s}[n] &= \arg \min_{s \in \mathcal{S}(\mathbf{p})} \| \, \ell[n] + \bar{\mathbf{r}} - s \, \|_\infty \\ &= \arg \min_{s \in \mathcal{S}(\mathbf{p})} \| \, \ell_s[n+1] \, \|_\infty \, . \end{aligned} \tag{6.2}$$

The basic idea behind this scheduler is to try to keep the lag bounded by choosing $\mathbf{s}[n]$ to minimize the size of the lag vector at the next time step, $\ell_s[n+1]$. If the absolute value of the elements of the lag vector stay bounded below some finite number b, then the time average schedule will converge to the desired schedule $\bar{\mathbf{r}}$ as we saw in Section 3.2. We now show that using this dynamic scheduler with M equally-sized pieces results in $B \leq 1/M$ for all $\bar{\mathbf{r}}$ in a set $\bar{R} \subset \mathbb{S}_N$. We call the set of vectors $\bar{R}$ the *uniform rate set.*

Theorem 6.1 (Lag Bound for Equally-Sized Resources) *The scheduling rule in Equation (6.2), with M equally-sized resources ($\mathbf{p} = \mathbf{p}_0$) yields*

$$|\ell_i[n]| \leq \frac{N-1}{N} \cdot \frac{1}{M} < \frac{1}{M}$$

for all i, n, and $\bar{\mathbf{r}} \in \bar{R}$. The uniform rate set $\bar{R}$ is the set of vectors $\bar{\mathbf{r}}$ such that for all $\mathbf{s} \in \mathcal{S}(\mathbf{p}_0)$ there is no pair of indices i and j such that $(s_i - r_i) - (s_j - r_j) > 2/M$.
Proof: Follows from Corollary B.2 and Theorem B.6 in Appendix B.4. ∎

We can interpret the set $\bar{R}$ as a constraint that $\bar{\mathbf{r}}$ is not closer to any point whose elements sum to 1 and are multiples of β that is outside the schedule constellation $\mathcal{S}$ (these points outside $\mathcal{S}$ have at least one negative element) than it is to a point in $\mathcal{S}$ itself.

Figure 6-1 depicts $\bar{R}$ for $M = 8$, $N = 3$. As the figure shows, the set $\bar{R}$ is a large subset of $\mathbb{S}_3$, only lacking small sections near the boundaries. For large values of M, this means that $\bar{R}$ is a large portion of the total volume of $\mathbb{S}_3$. To see that $\bar{R}$ remains a large portion of the total volume for larger N, note that even for large values of N, the set of points $\bar{\mathbf{r}}$ such that $r_i \geq 1/M$ for all i is a subset of $\bar{R}$. This subset only neglects the boundary of $\mathbb{S}_N$, and as M increases the thickness of the excluded boundary decreases. So, because the set $\bar{R}$ contains this smaller subset, $\bar{R}$ is a large portion of $\mathbb{S}_N$ for large values of M and N.

It is important to realize that the restriction of $\bar{\mathbf{r}}$ to $\bar{R}$ is needed only to guarantee the simple lag bound of Theorem 6.1. In practice, when $\bar{\mathbf{r}}$ is allowed to take values anywhere in $\mathbb{S}_N$, the lag appears to satisfy the bounds

$$-\left(\frac{N-1}{N} \cdot \frac{1}{M}\right) \leq \ell_i[n] \leq \left(\frac{N-1}{N} \cdot \frac{2}{M}\right) \, .$$

This means that the benefits that we get by using variable-sized pieces (as we see next) still apply for these values of $\bar{\mathbf{r}}$, although the actual bounds may be larger than when $\bar{\mathbf{r}}$ is in the set $\bar{R}$.[1]

[1]The tightest analytical bound we have come up with so far replaces the 2 with $N-1$; however, empirically the given bound holds.

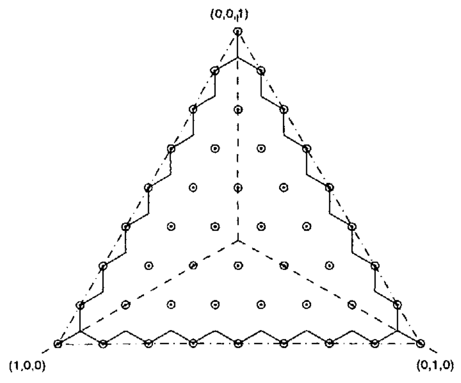


Figure 6-1: The uniform rate set for $N = 3$, $M = 8$. The circles are the points in $\mathcal{S}(\mathbf{p}_0)$, the dashed lines are the axes, the dash-dot lines indicate the region $\mathbb{S}_N$, and the solid lines indicate the boundary of the region $\bar{R}$.

We can relate these lag bounds with the earlier stability results for deterministic models in Chapter 4. In this case, we have a very tight bound on the variation of the lag. It turns out that because of the geometry of the constellation, the allocation regions achieved by using an optimization criterion based on the ∞-norm is equivalent to the regions achieved when the 2-norm is used. We would expect then, from Theorem 4.3, that the lag would be ultimately bounded. In fact, in this case the lag is bounded very closely to the convex hull of the constellation.

6.1.1 Variable-Sized Resources

Theorem 6.1 applies to equally-sized pieces. However, if we allow $\mathbf{p}$ to be any M-dimensional vector that corresponds to a uniformly dense constellation, we can choose schedules at each round to exactly emulate as many as $\bar{\mathbf{P}}^+(M)$ equally-sized pieces. We can therefore generalize Theorem 6.1 as follows.

Theorem 6.2 (Lag Bound for Variable-Sized Resources) *The scheduling rule in Equation (6.2), with M resources of sizes given by $\mathbf{p}$ that correspond to a uniformly dense constellation $\mathcal{S}(\mathbf{p})$ yields*

$$|\ell_i[n]| \leq \frac{N-1}{N} \cdot \frac{1}{M'} < \frac{1}{M'}$$

for all i, n, and $\bar{\mathbf{r}} \in \bar{R}$ where $M' = \bar{\mathbf{P}}(M)$. The uniform rate set $\bar{R}$ is the set of vectors $\bar{\mathbf{r}}$ such that for all $\mathbf{s} \in \mathcal{S}(\mathbf{p})$ there is no pair of indices i and j such that $(s_i - r_i) - (s_j - r_j) > 2/M'$.
Proof: Follows from the emulation property. ∎

Because M' is maximized when $\mathbf{p} = \mathbf{p}^+$, using $\mathbf{p}^+$ will give us the tightest lag bound. Although the preceding lag bounds have factors relating to N in them, we will ignore them in the following sections. We are more interested in the properties of the scheduler as a function of M, so we use the upper bound $1/M$ (or $1/M'$) as the actual lag bound.

6.2 Distributed Sensor Data Transmission

Now that we have the basic model set up, we will move to an application of the variable-size results from Section 5.4 as an illustration of how they might affect the design of a communication system.

We have $N > 1$ sensors, each producing data at a particular (unknown at design time) rate. We call the vector of these rates $\bar{\mathbf{r}}$. It is important to realize that all of what we present here only makes sense if this rate vector is unknown at design time, or may change with time. If it is constant and known, we can simply design channel sizes to exactly carry the right amount of data for each sensor. The data all has to be broadcast back to a central processing server over the same shared communication link. The shared communication link has a capacity of 1 Gbit/sec. Our goal is to design a partition $\mathbf{p}$ of the communication link into M independent channels that will be used for communication between the sensors and the central server. Our options are:

- M equal channels, $p_i = (1/M)$ Gbit/sec each;
- M variable-sized channels, $\sum_i p_i = 1$ Gbit/sec.

The allocation of channels to sensors is updated once every second. We assume here that the i-th sensor is producing data at a constant rate r_i, and so if we have a large enough buffer in the sensor, the sensor will be able to make use of whatever allocation of capacity it is given during each update. We will compare the cases of equal- and variable-sized pieces with respect to the variability of the capacity allocated to the sensor transmitters, and amount of total buffer space needed at the sensors. If we know that the lag is bounded by $\pm B$, then the buffer space required to handle the worst case lag for sensor i is at least $b_i = r_i + B$.[2] This happens when the lag is equal to B, or we are B Gbits behind in sending data, and the sensor produces r_i Gbits more data to send.

6.2.1 Many Equal Channels

This means that if we use M channels of equal size, the i-th sensor requires at least $r_i + (1/M)$ Gbits of buffer space because we know the lag is bounded by $1/M$. It is clear that the more channels we use (larger M), the closer we can approximate the number of bits each sensor wants to send during each second, and the less the buffer varies over time. However, we still need to buffer at least r_i bits for each sensor. Also each sensor needs to be able to send at rates in the range $r_i \pm (1/M)$ Gbits/sec.

However, as more and more equal channels are used, the complexity of assigning and tracking the allocations, and the necessity of having the sensors send on a number of channels simultaneously makes the scheduling fairly complex.

6.2.2 Many Variable-Sized Channels

When we allow the sizes of the channels to vary, we can either decrease the buffer space needed at each sensor, or reduce the complexity of scheduling. For a given number of channels, M, variable-sized channels can emulate M' equal channels, where $M' \leq \bar{\mathbf{P}}(M)$ for some $\mathbf{P}$ that corresponds to a $\mathbf{p}$ that has a uniformly dense constellation. So, for the same scheduling complexity, we can reduce the buffer requirements to $r_i + (1/M')$ for each sensor. When $M < N$, the optimal variable-sized channels are simply M equally-sized channels because there is no other choice if we are to stick with a partition $\mathbf{p}$ that yields a uniformly dense constellation. When $M \geq N$, the optimal variable-sized channels start to differ from the equally-sized channels. Because the allocations are able to approximate the data rates more closely than with equally-sized channels, each sensor will be sending at a total rate in the range $r_i \pm (1/M')$ (across all allocated channels) which is closer to its data rate r_i during each second than if we had used equally-sized channels. Thus, the variation in transmission rates for a particular sensor is decreased.

[2]This is a lower bound on the required buffer space, because if $\bar{\mathbf{r}}$ is not in the uniform rate set $\bar{R}$, the lag could be worse and force the sensor to store more data.

M	3	4	5	6	7	8	9	10	11	12
$N = 3$	$\frac{3}{4}$	$\frac{4}{7}$	$\frac{5}{11}$	$\frac{6}{17}$	$\frac{7}{26}$	$\frac{8}{40}$	$\frac{9}{61}$	$\frac{10}{92}$	$\frac{11}{139}$	$\frac{12}{209}$
$N = 4$	$\frac{3}{3}$	$\frac{4}{5}$	$\frac{5}{7}$	$\frac{6}{10}$	$\frac{7}{14}$	$\frac{8}{19}$	$\frac{9}{26}$	$\frac{10}{35}$	$\frac{11}{47}$	$\frac{12}{63}$
$N = 5$	$\frac{3}{3}$	$\frac{4}{4}$	$\frac{5}{6}$	$\frac{6}{8}$	$\frac{7}{11}$	$\frac{8}{14}$	$\frac{9}{18}$	$\frac{10}{23}$	$\frac{11}{29}$	$\frac{12}{37}$

Table 6.1: Values of fairness ratio γ_M for various combinations of N and M.

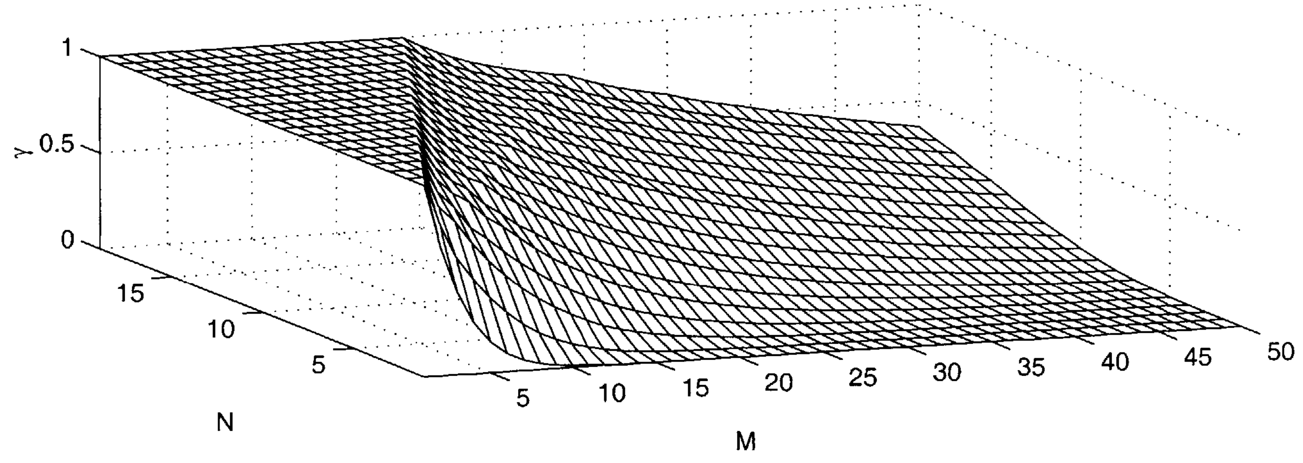


Figure 6-2: Values of fairness ratio γ_M.

The main drawback to using variable-sized channels in this situation is that each sensor must be able to send data at a wide range of possible rates on differently-sized channels. However, because we can choose channel sizes anywhere between equally-sized to $\mathbf{p}^+$, we can tune the system to the abilities of the sensors. By choosing $\mathbf{P}$ appropriately, we can vary the ratio between M and M' and the range of channel sizes. If we define the ratio M/M' as γ, we can express the total buffer space required for all of the sensors as $1 + \gamma(N/M)$. We will define γ_M as the maximum value of γ, for $M' = \bar{\mathbf{P}}^+(M)$ and will refer to it as the *fairness ratio.* We can sacrifice buffer space for sensor simplicity. For example, if we choose the sizes

$$\mathbf{p} = (1/30, 1/30, 2/30, 2/30, 4/30, 4/30, 8/30, 8/30) \ ,$$

instead of

$$\mathbf{p}^+ = (1/40, 1/40, 2/40, 3/40, 4/40, 6/40, 9/40, 14/40) \ ,$$

we get $\gamma = 4/15$ as opposed to $\gamma_M = 1/5$. However, each sensor only needs to be able to send at 4 distinct rates which are geometrically related and may be more practical to implement.

Table 6.1 shows values of γ_M for a number of values of N and M, and Figure 6-2 shows similar data graphically. Our previous bounds for a system with M equally-sized channels showed that the lag was bounded closely to the reservation by $\pm 1/M$. By varying the sizes of the channels, we can get an exponential decrease in the lag/reservation difference, and we require less buffer space for the sensors. Also, by reducing the variability of the channel capacity allocations to the sensors, we decrease the latency from $r_i + (1/M)$ to $r_i + (1/M')$ because less data needs to be buffered before being sent.

Although using variable-sized channels reduces the buffer space required, in the region where M is large compared with N, where variable-sized channels give the most benefit, $1/M$ is much smaller than r_i. The reduction in buffer space that results is not a very large part of the space needed for the reservation for each individual sensor. Even if we instead consider the total buffer space for all of the sensors (perhaps at the central server, where data may need to be sent to the sensors) this reduction becomes larger, but still remains the smaller portion of the buffer space required. The total buffer space required is

$$\begin{aligned} B &= \sum_{i=1}^{N} \frac{1}{M} \cdot \gamma + r_i = 1 + \sum_{i=1}^{N} \frac{1}{M} \cdot \gamma \\ &= 1 + \frac{N}{M} \cdot \gamma \end{aligned}$$

For this case, if M is much greater than N, the bulk of the buffer space comes from the first term, and as N increases, the maximum reduction due to γ decreases as well. In many cases, however, γ can provide a measurable benefit. For instance, if we consider the $N = 3$, $M = 8$ case again, the total buffer space with equal channels is 1.375 Gbits as compared with the space needed for variable channels with the maximum γ_M of 1.075 Gbits, a reduction of about 30%. Even for $N = 9$, $M = 24$, we get 1.11 Gbits compared with 1.375 Gbits, a 25% savings.

The real advantage to using variable-sized channels is that the variation in the lag is reduced by a factor of γ. This means that the data that is sent by the sensors is received more smoothly over time instead of in large bursts. At the end of time slot n, each sensor has sent its desired amount of bits, $r_i n$ plus or minus $(1/M')$ Gbits. For example, with $N = 9$, and $M = 24$, $\gamma_M = .29$. So, if we used M equal channels, the variation of the lag would be ± 41.7 Mbits; with M variable-sized channels, the variation would be $\pm 41.7 \cdot \gamma_M$ Mbits, or ± 12 Mbits. With $N = 10$, and $M = 100$, $\gamma_M = 4.7 \times 10^{-4}$. Now we reduce the lag variation from $\pm 10,000$ Kbits to $\pm 10,000 \cdot \gamma_M$ Kbits, or ± 4.7 Kbits.

6.3 Many, Many Resources: Virtual Channel Groups

Another region of interest is when M is very large. Consider a telephone cable with a total capacity of 1 Gbit/sec. Assume that the link consists of 1000 smaller, 1 Mbit/sec physical channels.[3] If we were to try and allocate these physical channels between a few users with large data rates, any scheduling scheme that tries to allocate the physical channels individually will quickly become very complex. However, if we consider instead the problem of partitioning the M channels into a smaller number of channel groups that can be allocated to the users as a whole, the results above tell us that we can achieve the allocation granularity of M channels by using a smaller number of larger channels that grows logarithmically in M. To do this, we run the channel capacity design algorithm m rounds until $\bar{\mathbf{P}}^+(m) \geq M$. If $\bar{\mathbf{P}}^+(m) = M$, we are finished. If $\bar{\mathbf{P}}^+(m) > M$, we can simply replace the m-th element of $\mathbf{p}^+$ with a smaller one so that $\bar{\mathbf{P}}^+(m) = M$. These capacities still produce a uniform constellation and can represent any allocation with a granularity of $1/M$.[4]

[3]The numbers are not realistic, but the right order of magnitude.

[4]The first part of this statement follows because if we have a sequence $\mathbf{p}^+$ of sizes, we can insert any other size d between $\mathbf{p}_i^+$ and $\mathbf{p}_j^+$ such that $\mathbf{p}_i^+ \leq d \leq \mathbf{p}_j^+$ and be able to have a uniformly dense constellation with the new set of sizes $\mathbf{p}'$. This is because $d \leq \mathbf{p}_j^+$, so it does not violate the property of Lemma 5.4, and the remaining $\mathbf{p}_k^+$, $k \geq j$ also do not violate the property.

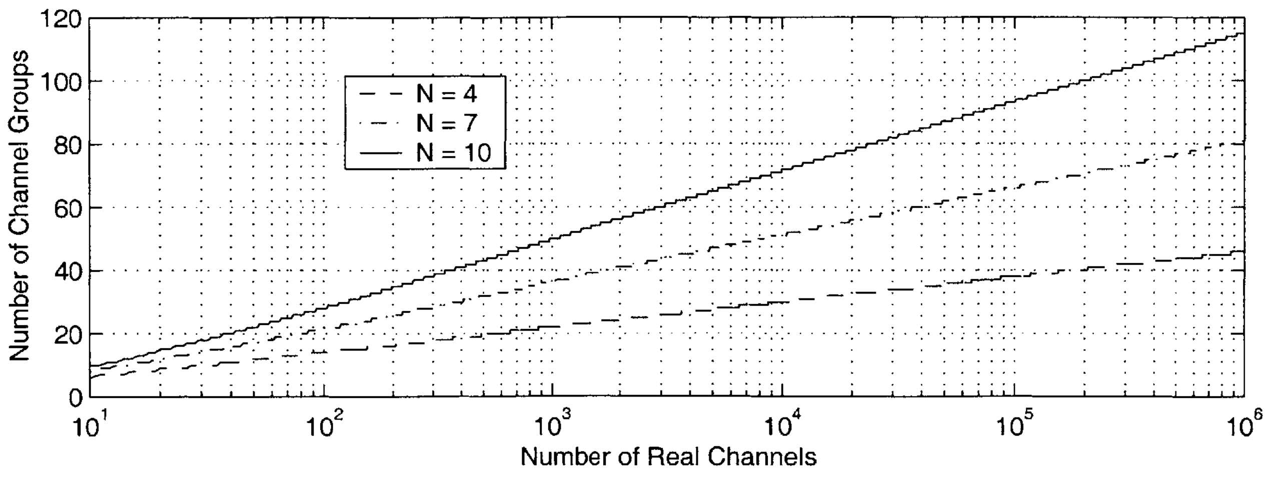


Figure 6-3: Number of channel groups needed for $N = 4, 7, 10$ and values of M between 10 and 10^6.

For example, if $N = 10$ and $M = 1000$, we only need $m = 50$ channel groups to simulate the 1000 physical channels.[5] This can be a benefit in a practical implementation, as the scheduling algorithm only needs to keep track of the channel groups, not the actual channels themselves. Figure 6-3 shows the number of channel groups needed for $N = 4, 7, 10$ and values of M between 10 and 10^6. The reduction in control complexity, also reduces the amount of storage needed to specify which users are allocated which channels. If we take the above example again, we will only require $50 \log_2 10 \approx 200$ bits to store a complete allocation of the 10 users to the 1000 channels without any loss of granularity. If we were to allocate the channels individually, we would require $1000 \log_2 10 \approx 4000$ bits. This method of grouping channels allows a large reduction in the amount of control data that needs to be manipulated.

6.4 Varying Numbers of Users

Up until now we have restricted the number of competing users to be a constant. This is required because the sizes of the resource pieces or channel groups that we choose depend on the number of users. We can obviously account for changing numbers of users if we can switch between sets of piece sizes depending on how many users are currently competing. In a somewhat static case such as this, where users do not come and go often, we can simply change the channel groups when a new user arrives or an existing user departs. This will only require us to also keep track of the number of users ($\log_2 N$ bits) for which we make a particular allocation.

If it is practically infeasible to have this many possible configurations, we can always use a few configurations for a few values of N, and simply choose the configuration for the smallest value of N that is at least as large as the number of users.

[5]For $m = 50$, we can actually simulate 1083 physical channels.

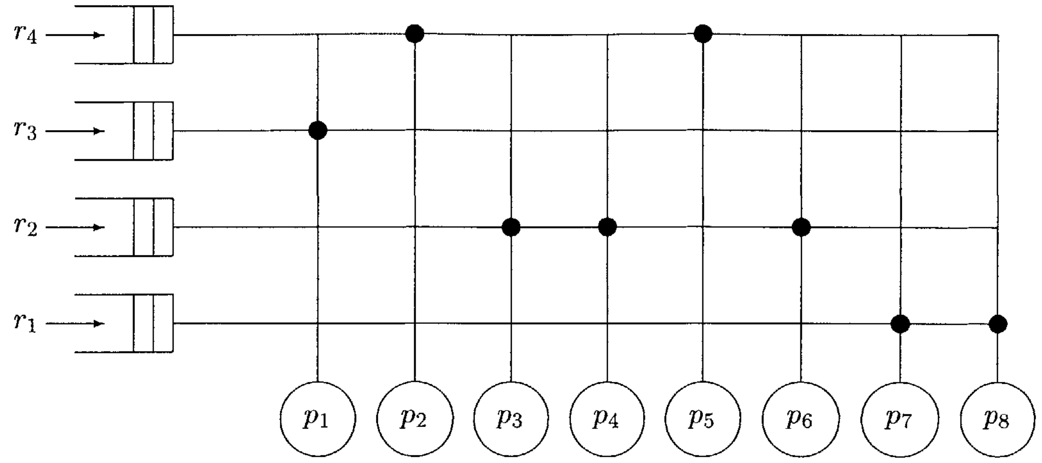


Figure 6-4: Batch queueing setup. Each of the four queues is connected up to some subset of the servers.

6.5 Batch Queueing System

Another possible application when M can be very large compared with N is in certain parallel processing applications. If we have a large number of identical processors that must be allocated among a few processes, we can come up with groups of processors that can be allocated together without reducing the ability to allocate the processors with a granularity of a single processor. This not only reduces the complexity of scheduling the processors, but allows the groups to be decided beforehand to minimize other costs such as communication between processors. This also requires that we are trying to allocate the processors according to some set of computational reservations assigned to the processes, and that the processes always have enough work for the number of processors assigned to them. The next application describes a situation which might be able to model a situation in which the work for each of the processes comes probabilistically, but with some average rate.

Although we have talked about variable-sized channels in the case where the data rates are constant, it is also important to consider a queueing model, where jobs arrive and must be serviced according to a probabilistic model. The previous examples assumed that each user could always make use of whatever allocation it was given; here, data may not always be available, or queued, for processing or transmission.

We consider a setup in which jobs of N different classes arrive at a total rate of less than R equally-sized jobs per round. They are kept in separate queues for each class. There are M servers available; server i can accommodate a maximum of P_i jobs of the same class every round, and incurs a cost of $C_i(s)$ dollars when it services s jobs during a particular round. This setup can be pictured as N input queues being routed to M servers as shown in Figure 6-4. At each time slot, we need to select which servers are connected with each queue. Scheduling decisions are made every round, wherein each server is allocated to some class. A server is never left idle if there are additional packets to be processed (i.e. the scheduling policy is non-idling). As before, $\mathbf{A}[n] \in \mathcal{A}$ is the allocation matrix for round n, and $\mathbf{s}[n] = \min(\ell[n], \mathbf{PA}[n])$ is the schedule that determines how many jobs are able to be processed from each class. The total cost during each round is

$$C[n] = \sum_{i=1}^{M} C_i(s_i[n])$$

with the constraint that a server can not be idle if there are still packets to be processed. Our objective is to select the parameters P_i with $\sum_{i=1}^{M} P_i \geq R$ such that the average cost per round is minimized,

$$\bar{C} = \lim_{n\to\infty} \frac{1}{n} \sum_{m=1}^{n} C[m] \ .$$

Similar to our earlier work, we will concentrate on choosing sizes P_i such that the vector $\mathbf{P} = m\mathbf{P}'$, where $\mathbf{P}'$ is an integer vector such that $\mathbf{p}' = \mathbf{P}'/\bar{\mathbf{P}}' \in \Gamma(M, N+1)$ and m is an integer. This guarantees that servers with sizes $\mathbf{P}$ can emulate $\bar{\mathbf{P}}'$ servers of size m. Just as before, if the choice of $\mathbf{P}$ is left completely unconstrained, the problem of selecting and using the optimal vector $\mathbf{P}$ is difficult.

Although we have stated this optimization problem precisely, we are not concerned here with exact solutions or examining specific scheduling policies; instead, we would just like to outline how our previous results prove useful in approaching a solution to the optimization problem.

In many contexts, a cost $C_i(s) = P_i + \alpha s$ for a busy server is appropriate. The first cost term, P_i, is a fixed start-up cost equal to the size of the server if the server is busy and zero otherwise, and the second job-specific cost, αs, is proportional to the number of jobs served. It is easy to see that the average job-specific cost over a large number of rounds under any non-idling policy is equal to the average number of packets that are processed per round which is independent of the scheduling policy and server sizes, making the job-specific cost not very interesting as far as our optimization objective is concerned.

So, we will neglect the job-specific cost, and concentrate on the start-up cost for a busy server, $C_i(s) = P_i$. This cost is equal to the size of the server if the server is busy, and independent of the actual load on the server. For example, when a number of processors are shared among a number of users, if one user locks up a processor for a certain amount of time, but leaves the processor idle for part of the time, the effective cost is the total amount of time, not the busy time, because no other users can use the processor. In this case, the choice of server sizes affects the average cost.

We begin by considering the case when $R \leq \bar{\mathbf{P}}^+(M, N+1)$. When we set the speeds P_i of the servers equal to P_i^+ for $N+1$ bins and M pieces, we will be able to serve at a total rate up to $\bar{\mathbf{P}}^+(M, N+1)$ jobs per round. This is easy to see by considering two cases. First, if there are more than $\bar{\mathbf{P}}^+(M, N+1)$ jobs in the queues, we know that we can allocate any set of $\bar{\mathbf{P}}^+(M, N+1)$ of them in N queues by the definition of $\mathbf{P}^+$, filling all of the servers during that slot. Second, if there are $K < \bar{\mathbf{P}}^+(M, N+1)$ jobs in the queues, imagine an $(N+1)$-th queue with $\bar{\mathbf{P}}^+(M, N+1) - K$ imaginary jobs in it. By definition of $\mathbf{P}^+$, we can schedule all of the (real plus imaginary) jobs from $N+1$ queues into M servers by allocating each server to a single queue; all of the servers that are allocated to queue $N+1$ are left idle for that slot. Thus, no matter how many jobs are in the queues, we will be able to allocate the M servers to the N queues in such a way that either a server is completely full, or is completely empty for the entire slot. This allows us to implement non-idling scheduling policies by only making decisions at fixed, regular time instants. Also, from the definition of $\mathbf{P}^+$, any other $\mathbf{P}$ that violates the condition of Lemma 5.4 will not be able to schedule at least one possible configuration of the queues in this way. It is also clear that the cost $C[n]$ is never less than the number of packets serviced during round n. In this case, the cost $C[n]$ is exactly equal to the number of packets serviced at each round, and so $C[n]$ is minimized at each

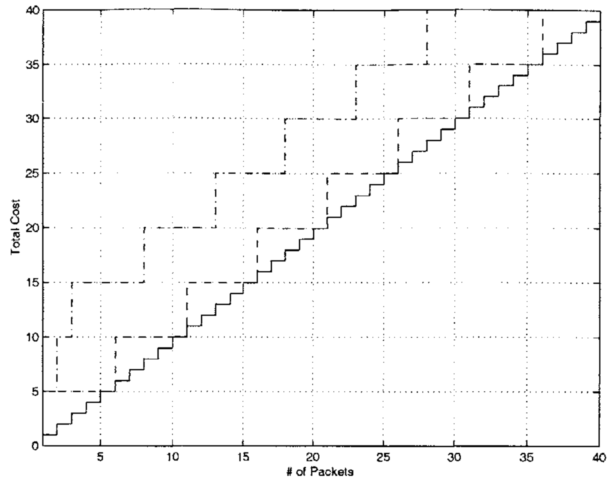


Figure 6-5: Cost profiles for variable- vs. equally-sized servers; $R = 40$, $M = 8$, $N + 1 = 3$.

round over all possible sets of server sizes that sum to at least R. This implies that the average cost $\bar{C}$ is also minimized when we use server sizes equal to the elements of $\mathbf{P}^+(M, N+1)$.

As a comparison, consider using M equally-sized servers instead, each of speed $P_i = \lceil R/M \rceil > 1$. By the construction of $\mathbf{P}^+$, the difference in cost between processing K packets and processing $K + 1$ packets is exactly one dollar. This follows because we can always find an allocation that serves exactly those $K + 1$ packets with no idle time. However, for equally-sized servers, the cost differential for an additional packet from user j is either 0 if there is a partially empty server already assigned to user j or $\lceil R/M \rceil$ if all servers assigned to i are currently full, and the total cost required for a given number of packets is never less than the cost required with the variable-sized servers because there may be partially empty servers. In other words, $C[n]$ may be larger than its minimum possible value. In the worst case, if each user has a single packet, N packets require $N \cdot \lceil R/M \rceil$ dollars because each packet requires a separate server. For the variable-sized servers, each packet will use a server of size 1, and so only N dollars will be spent. We can visualize the difference between the M equally-sized servers and the M variable-sized servers by plotting the cost required to process some number of packets during a round (see Figure 6-5). The figure actually shows three cost profiles. The solid line corresponds to the variable-sized servers, where processing p packets requires p dollars. The dashed line shows the best-case cost needed when we use equally-sized servers. This minimum cost occurs when only one of the user queues has a partially empty server assigned to it at a time. The dash-dot line shows the worst-case cost needed for equally-sized servers. The worst-case cost occurs when all of the queues have partially empty servers. Ideally we would like to plot cost versus the number of packets in each queue, but we can not visualize this easily for $N > 2$. As an example of the difference between the best-case and worst-case costs, assume we have a total of 12 packets waiting to be processed. No matter how we distribute them to the three users, we will always require at least 3 servers of size $R/M = 5$ to process them. This gives us the best-case cost of 15 dollars. It is also true that there is no way to distribute the 12 packets to the users that uses more than 4 servers. This gives us the worst-case cost of 20 dollars.

Now we have to consider what happens when $R > \bar{\mathbf{P}}^+(M, N+1)$. As we have mentioned, we will concentrate on choosing sizes P_i such that the vector $\mathbf{P} = m\mathbf{P}'$, where $\mathbf{P}'$ is an integer vector which corresponds with a uniformly dense constellation and m is an integer. By this definition $\mathbf{P}$

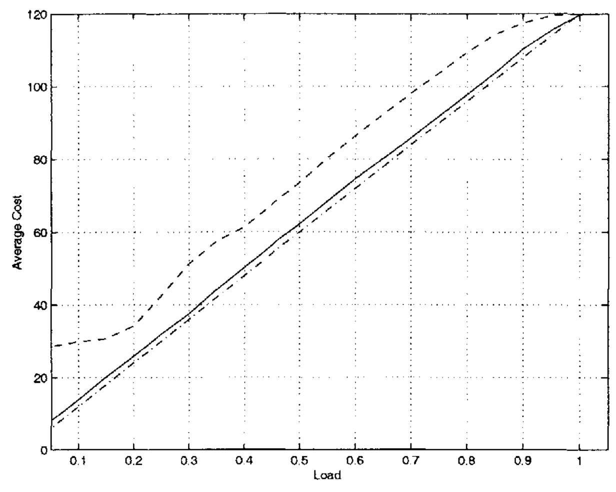


Figure 6-6: Average cost $\bar{C}$ over a range of loads for $M = 8$, $N = 2$. The solid line shows the average cost when using $\mathbf{P}' = \mathbf{P}^+(8,3)$ and $m = 3$, the dashed line shows the average cost when using equal servers of size 15, and the dash-dot line shows the minimum possible average cost (R equal servers of size 1).

has a larger, scaled constellation with a coarser resolution. In order to handle an average rate less than R, we need $m\bar{\mathbf{P}}' = \bar{\mathbf{P}} \geq R$. By the emulation property, servers with sizes $\mathbf{P}'$ can emulate $\bar{\mathbf{P}}'$ servers of size 1. Thus, servers with sizes $\mathbf{P}$ can emulate $\bar{\mathbf{P}}'$ servers of size m. If we are only considering variable-sized servers that emulate some number of equally-sized servers, then the size of the emulated servers, m, is the metric that we would like to minimize. The smaller m is, the closer we can approximate the current profile of packets in the users' queues. For a size vector $\mathbf{P}'$, we define the smallest m that meets the minimum service rate requirement as

$$m = \left\lceil \frac{R}{\bar{\mathbf{P}}'} \right\rceil .$$

From this, it is clear that the smallest value of m (we will call it m^+) that is possible with the uniformly dense restriction on the vector $\mathbf{P}'$ occurs when we maximize $\bar{\mathbf{P}}'$ by choosing $\mathbf{P}' = \mathbf{P}^+(M, N+1)$.

It is also clear that, in general, we will have capacity that exceeds the maximum average rate R and so there will actually be a number of possible vectors $\mathbf{P}'$ that can be used, as long as $m^+\bar{\mathbf{P}}' \geq R$. For example, if $M = 8$, $N = 2$, $\bar{\mathbf{P}}^+(M, N+1) = 40$. If $R = 60$, then $m^+ = 2$ and $m^+\bar{\mathbf{P}}^+(M, N+1) = 80$. We could also choose $\mathbf{P}' = (1,1,2,3,4,6,6,7)$ which corresponds with a uniformly dense constellation, and $m^+\bar{\mathbf{P}}' = 60 = R$. This choice of $\mathbf{P}'$ will result in smaller initial costs needed to buy the servers, and also has a smaller spread between the smallest and largest elements which would be more practical to implement. In general, we can pick any element $\mathbf{P}'$ that corresponds with some $\mathbf{p}' \in \Gamma(M, N+1)$ such that $m^+\bar{\mathbf{P}}' \geq R$.

Simulation results are shown in Figure 6-6. The plot shows the average cost when using the same set of servers for a number of different loads. The load is the ratio between the average arrival rate of jobs and the maximum processing rate of the servers. During each round, the number of jobs that arrive at each of the two input queues is a Poisson random variable. We see that using variable-sized servers, we have a roughly constant decrease in average cost for low loads. As the load increases, the queues are very full more often, and then the servers can almost always be completely filled no matter how we choose the sizes of the servers. By choosing the sizes as a multiple of the vector $\mathbf{P}^+(8,3)$, we are able to achieve a smaller average cost than for equally-sized servers over the entire range of loads.

6.5.1 Package Delivery Problem

Now consider another illustration of these results. A shipping company must deliver a large number of packages from its central office to a small number, N, of satellite offices every day. Assume that the packages are all the same size, and that packages arrive at an average rate P per day. The company has M possible drivers, and we will assume that it will buy exactly M trucks to deliver the packages. In the course of buying the trucks, the company has to decide what sizes (as integer multiples of the package size) to buy.

We assume that a component of the cost for a delivery is directly proportional to the size of the truck (as before we neglect the cost proportional to the load). If this is the case, then the company would like to choose the sizes of the trucks to minimize the empty space on the trucks when they make deliveries.

As we saw for the batch queueing system, we will assume that the company buys M trucks with a total capacity of at least $R > P$. The sizes of the trucks will be chosen to be the elements of the vector $m\mathbf{P}^+(M, N+1)$ where m is the smallest integer such that $m\bar{\mathbf{P}}^+(M, N+1) \geq R$. If $P < R \leq \bar{\mathbf{P}}^+(M, N+1)$ (and thus $m = 1$), these sizes will allow us to accommodate any possible set of deliveries up to the maximum capacity $\bar{\mathbf{P}}^+(M, N+1)$ of the trucks, with no empty space. It is clear that this distribution of sizes minimizes the delivery cost because it always uses trucks whose total size equals the number of packages that need to be delivered, which is the minimum size required to deliver the packages. If we were to choose to buy M trucks of the same size instead we would end up sending partially empty trucks to some of the destinations.

When $R > \bar{\mathbf{P}}^+(M, N+1)$ the best-case cost for the first of every m packages is equal to m (the size of the smallest truck), no matter how many packages ($\leq R$) we deliver, or where they are delivered to. If we were to use M identical trucks of size $\lceil R/M \rceil$ instead, we pay a cost of at least $\lceil R/M \rceil$ dollars for the first of every $\lceil R/M \rceil$ packages. This means that by choosing the sizes of the trucks appropriately, we can reduce the average cost per package.

The best- and worst-case cost profiles of these two choices for size vectors are shown in Figure 6-7 for particular choices of R, M, and N. Note that even though the best-case profile for equally-sized trucks dips below the profile for variable-sized trucks occasionally, on average the cost is higher. In order to quantify the higher cost, we would calculate the expected value of the cost as a function of the queue lengths given a steady state probability distribution over the possible queue length vectors. In order to discuss this, we would need to specify how the packages are allocated to the trucks; when there are too many packages waiting, we need to decide how to allocate the trucks to the destinations. As a practical rule, we will assume we choose the allocation that minimizes the maximum number of packages left in any queue after removing the allocated packages.[6]

As a numerical example, assume there are 10 satellite offices that receive fewer than 5000 packages per day on average. We only have 75 drivers, so we can only buy 75 trucks. The obvious choice is 75 trucks that can each hold 67 packages, for a maximum capacity of 5025 packages per day, and in the worst case $66 \cdot 10 = 660$ packages worth of wasted space (10 of the trucks carry a single package). If we choose the sizes of the trucks to vary according to $\mathbf{P}^+$, we can deliver

[6]Another practical allocation rule might be to only send full trucks on deliveries. In this case, the variable-sized trucks can accommodate a finer granularity of packages, and so would not leave the packages waiting as long as equally-sized trucks of the same total capacity.

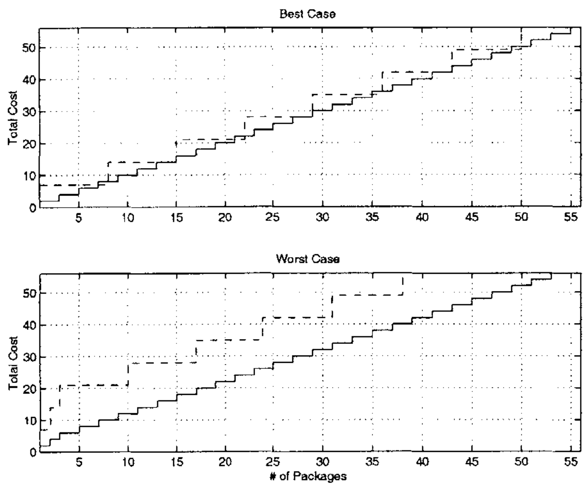


Figure 6-7: Best and worst case cost profiles for equally-sized trucks (dashed lines) vs. best and worst case cost profiles for variable-sized trucks (solid lines); $R = 56$, $M = 8$, $N + 1 = 3$, $m = 2$.

a total of 7871 packages per day while never wasting any space. The only issue with using these variable-sized trucks is that the largest truck is 716 times bigger than the smallest. If we limit the sizes of the trucks to be less than 210, we can still handle more than 5000 packages per day without wasting any space. If we allow the trucks to be sizes that are multiples of 2 ($m = 2$), we need a maximum truck size of 144 to guarantee 5000 packages per day with a worst case 10 packages worth of wasted space. If we increase m to 8, with a worst case 70 packages of wasted space we only need a maximum truck size of 96 packages, and the only sizes required are multiples of 8 from 8 packages to 96 packages. This tradeoff between m and the maximum truck size can be used to balance the feasibility of buying trucks with many varying specific sizes against the costs incurred by using partially empty trucks.

We might even extend this idea in another direction, and consider how choosing the truck sizes might reduce the number of trucks required. Like our previous channel grouping example, we can use fewer variable-sized trucks to emulate the performance of many equally-sized trucks. For instance, to deliver to two satellite offices with no empty capacity we can use three trucks of sizes 1, 1, and 2 instead of four trucks of size 1. This allows the shipping company to reduce the number of drivers and perhaps the amount of maintenance required for the trucks. From our previous example, if there are 10 satellite offices that need to receive fewer than 5000 packages per day on average, we can either use 5000 trucks of size 1, or 71 trucks of sizes $\mathbf{P}^+$ for $M = 71$, $N + 1 = 11$ to insure that we never have any empty space in any truck. We can also use a number of other possible truck sizes if the sizes correspond to some vector in Γ.

In addition, we may be able to use fewer trucks if we allow a small amount of empty space. As a simple example when this might take place, consider the following. If we needed to transport at most 55 packages to two destinations every day, we could choose 9 trucks with sizes $\mathbf{P}_1 = (1, 1, 2, 3, 4, 6, 9, 14, 15)$ and guarantee that there would never be any empty space, or we could use 8 trucks with sizes $\mathbf{P}_2 = (1, 2, 3, 4, 6, 9, 14, 16)$ if we are willing to live with empty space every once in a while. If we use the sizes in $\mathbf{P}_2$, there are only 9 combinations (out of approximately 1600) of at most 55 waiting packages that can not be exactly transported with these trucks. As long as these combinations do not occur very often, and if we have some extra capacity, we can reduce the number of trucks while not affecting our costs a great deal.

6.5.2 Variable Length Packets

We now briefly discuss an extension of the previous batch queueing model to one in which we allow variable length packets. If the packets require an integer number of rounds to process, and the processing can be preempted if necessary at every round, we can simply view the system as having a larger arrival rate. However, if the processing can not be preempted, the set of servers available at each round for scheduling varies from round to round. This means that we are not always guaranteed to have a set of servers with speeds that can insure a non-idling schedule. However, if we use the algorithm given in Section 6.6 to generate the resource partition we can still guarantee a non-idling schedule if we have an upper bound on the number of servers that will not be available during any round.

On the other hand, if we view the length of a packet as the number of processing cycles required, and the size of a server as its speed, we can process longer packets in a single round by assigning them a larger portion of a server. This allows the servers to be available at each round, but makes the problem of assigning servers to users more complicated. In this case, although the servers will be able to handle any possible workload as above without partially idling any server, we are now faced with the problem of deciding how to choose the packets that are to be processed. Because the packets vary in size, it is not always easy to choose the packets from the heads of the queues so that the total workload of the chosen packets equals the total capacity of the servers. To see this, consider the following example. We have two users and four servers with speeds of 1, 1, 2 and 3 packets/second. The first user has two packets of length 2 at the head of its queue, and the second user has a packet of length 2 followed by a packet of length 3 in its queue. If we can not split (or preempt) the packets, we can only assign two packets to be processed, even though the servers have enough capacity to handle one more packet. In addition, if we choose the packet from the first user to be processed in the server with speed 2, we will be forced to assign a packet of length 2 to the server with speed 3, causing it to idle. If we choose the packet from the second user first, the speed 3 server will be able to process a packet of length 3.

It is clear that an obvious goal would be to assign packets from the queues to the servers so that we have the fewest number of idle cycles (on servers that are not idle) as possible. This problem is very hard to solve optimally as a function of M and N. It is equivalent to a bin packing problem where the bin sizes correspond to the server speeds and the objects to be packed are the packets. Also, the packing order may be constrained such that we must always pack a packet at the front of one of the queues if the packets are to be served in the order they arrive. This bin packing problem is easily seen to be NP-hard. However, there are many heuristic algorithms for the bin packing problem that may prove useful in practice.

6.6 Robust Size Vectors

So far we have concentrated on discussing sets of resource pieces that are able to emulate a large number of equally-sized pieces. However, this property is not very robust to deletion of pieces from the set. If an element is removed from a size vector or, equivalently, set to 0, the resulting vector does not, in general, yield a uniformly dense constellation. In this section, we present a method for designing size vectors that retain the ability to yield uniformly dense constellations even when elements are removed. Of course, when we talk about a subset of a size vector yielding a uniformly dense constellation, we have to restrict ourselves to looking at a subset of $\mathbb{S}_N$, containing $\bar{\mathbf{r}}$ such that $\sum_i r_i = \alpha < 1$ where α is the sum of the sizes of the remaining pieces. We denote this set $\alpha\mathbb{S}_N$.

We will use the term *k-robust* to denote a size vector that is robust to the removal of k pieces. We use the notation $\mathcal{D}_k(M)$ to denote the collection of all the possible subsets of the indices from 1 to M that are missing at most k elements. The set $\mathcal{D}_k(M)$ is the set of all index sets $d \subset \{1, \dots, M\}$ such that $|d| \geq (M - k)$. If $M < k$ then $\mathcal{D}_k(M) = \emptyset$.

Algorithm 6.1 (Robust Uniformly Dense Schedule Constellations) *In order to generate the k-robust size vector* $\mathbf{p}$ *that has the maximally dense uniform constellation, given N and M, do the following to generate the proper piece sizes* $\mathbf{p}$ *(*$\mathbf{P}$ *is an integer vector of length M):*

1. *Set* $P_1 = 1$.

2. *For* $i = 1, \dots, M-1$*: if* $\mathcal{D}_k(i) = \emptyset$ *then set* $P_{i+1} = 1$*; otherwise set* $P_{i+1} = \min\limits_{d \in \mathcal{D}_k(i)} \left\lceil \frac{1 + \sum_{j \in d} P_j}{N-1} \right\rceil$.

3. *Set* $p_i = \frac{P_i}{\sum_{j=1}^{M} P_j}$ *for* $i = 1, \dots, M$.

We can see that the resulting size vector will always yield a uniformly dense constellation even when an arbitrary subset of no more than k pieces are removed from the size vector. This follows because if we are told that the pieces are removed, and the resulting size vector is $\mathbf{p}^k$, then we know that $p^k_{i+1} - p^k_i$ is smaller than the maximum allowed for any possible $\mathbf{p}$ of length $M - k$ if we are to maintain a uniformly dense constellation. We also know that this algorithm produces the vector $\mathbf{P}$ that is an upper bound on all possible k-robust size vectors because if some other size vector $\mathbf{P}'$ was such that $P'_{i+1} - P'_i > P_{i+1} - P_i$ for some i then there exists a subset d' of the first i pieces with no less than $i - k$ elements that achieves the minimum in the iterative step such that

$$P'_{i+1} > P_{i+1} = \min_{d \in \mathcal{D}_k(i)} \left\lceil \frac{1 + \sum_{j \in d} P_j}{N-1} \right\rceil = \left\lceil \frac{1 + \sum_{j \in d'} P_j}{N-1} \right\rceil$$

and so we will have violated Lemma 5.4 for the size vector of length $|d'| + M - i$ with elements equal to p'_j for $j \in d' \cup \{i+1, \dots, M\}$.

By the nondecreasing property of $\mathbf{P}$, we know that

$$\min_{d \in \mathcal{D}_k(i)} \left\lceil \frac{1 + \sum_{j \in d} P_j}{N-1} \right\rceil = \left\lceil \frac{1 + \sum_{j=1}^{i-k} P_j}{N-1} \right\rceil$$

and so it would seem that the minimization is not needed. However, we see that this algorithm will also compute the k-robust size vector with the densest uniform constellation when the possible

M	3	4	5	6	7	8	9	10	11	12
$N = 3$	$\frac{3}{3}$	$\frac{4}{5}$	$\frac{5}{7}$	$\frac{6}{10}$	$\frac{7}{14}$	$\frac{8}{20}$	$\frac{9}{28}$	$\frac{10}{39}$	$\frac{11}{54}$	$\frac{12}{74}$
$N = 4$	$\frac{3}{3}$	$\frac{4}{4}$	$\frac{5}{6}$	$\frac{6}{8}$	$\frac{7}{11}$	$\frac{8}{14}$	$\frac{9}{18}$	$\frac{10}{23}$	$\frac{11}{30}$	$\frac{12}{38}$
$N = 5$	$\frac{3}{3}$	$\frac{4}{4}$	$\frac{5}{5}$	$\frac{6}{7}$	$\frac{7}{9}$	$\frac{8}{11}$	$\frac{9}{14}$	$\frac{10}{17}$	$\frac{11}{21}$	$\frac{12}{26}$

Table 6.2: Values of fairness ratio γ for 1-robust size vectors and various combinations of N and M.

deletions are constrained; i.e. only certain combinations of pieces may be deleted together. In these cases, the minimization might not be simplified.

We also note that the vector $\mathbf{p}^+$ for $N + k$ bins is also a k-robust size vector for N, but is not the k-robust size vector with the maximally dense uniform constellation. This follows from the definition of $\mathbf{p}^+$. If we think of the $N + k$ bins as N real bins plus k bins whose sizes are equal to the sizes of the deleted pieces, and that each holds a single specific resource piece of the same size, then by the definition of $\mathbf{p}^+$, the remaining $M - k$ pieces must yield a uniformly dense constellation (in $\alpha\mathbb{S}_N$) when allocated to the N remaining bins. But, this property is the same as the k-robust property; if k pieces are unavailable, the remaining $M - k$ must yield a uniformly dense constellation in $\alpha\mathbb{S}_N$. However, $\mathbf{p}^+$ for $N + k$ does not yield the maximally dense uniform constellation for all k-robust size vectors for N bins because $\mathbf{p}^+$ is also constrained so that when the k extra bins hold on to more than k resource pieces, the remaining pieces must still yield a uniformly dense constellation.

6.6.1 Applications

We can also extend the applications discussed above to account for cases when the processors, channels, or queues are unreliable. For instance, if there is a probability that at most k queues are either broken or busy with other jobs when it comes time for scheduling, we can use a k-robust size vector to define the sizes of the queues so that we still maintain the beneficial properties discussed earlier.

Similarly for the channel group application, in Section 6.3, by using a k-robust size vector we increase the value of γ, but allow ourselves to handle k failures in the channels during a scheduling round, or equivalently k transmission errors during a round where the channel must retransmit the data during the next round and so is unavailable for scheduling.

We give a few representative values of γ for 1-robust size vectors in Table 6.2. If we compare this table to Table 6.1, we see that γ decreases much more slowly for robust size vectors. However, a k-robust size vector designed for a particular N grows more quickly, and has a smaller γ than γ_M for a non-robust size vector of the same length designed for $N + k$. We can see this for 1-robust size vectors when we compare Tables 6.1 and 6.2.

Chapter 7

Auction-Based Scheduling

In many situations where resources are under contention, it is important that we guarantee fair access to the resources in adverse circumstances. For example, fair access to a shared communication link by a number of data streams is a widespread problem in modern communication networks. When an allocation mechanism is designed, it must make sure that the link is not unfairly monopolized by a single, or a subgroup, of users. An example of an approach to doing this is the PGPS algorithm we discussed earlier. By limiting the relative rates at which competing streams can use the resource, the algorithm guarantees each stream a certain minimum rate of service. This makes it impossible for one stream to penalize others by sending too much data too quickly. By augmenting our basic allocation framework from Chapter 3 with the use of auctions as scheduling rules, we can capture the flavor of this approach in a well-defined setting. The results in this chapter establish that if we use an auction mechanism as a scheduling rule, with monetary bids as inputs, then if we control the relative amounts of income that the users receive over time, there is no way for the users to choose their bids in order to receive an unfairly high allocation of the resource.

Our interest in auctions stems from actual allocation mechanisms based on auctions that have been used in some cases. A paper by Waldspurger, et. al. [67] describes a mechanism called Spawn which requires processes competing for resources on a computer network to bid for processing time. Each process is given a stream of funding (measured in monetary units/time slice) with which to bid for time slices in an auction, against other processes. Each process can only bid on a single processor in the network during any round. The actual mechanism used to allocate the time slices is a second-price sealed bid auction, so each process gets one chance to submit a bid, and the process that submits the highest bid wins the time slice, paying only the second-highest bid. The goal of the auction process is to allocate the resource (processing time) fairly amongst the competing processes, where "fairly" means that processes with more money get more time (if they want it). However, it is not immediately obvious what bidding strategies will ensure that this goal is reached.

This model falls very close to our dynamic framework, where lag is identified with accumulated wealth, and the bids determine the schedule and payments that reduce the wealth. We investigate how the use of an auction in place of a quantizer for a simple repeated resource allocation problem (a single indivisible resource and many users) can change the properties of the allocation mechanism. The analysis in this section may be seen as a first step towards a more general investigation of more complex allocation mechanisms. The simple model presented in the previous chapters may be used in well-controlled and characterized environments; the auction mechanisms we look at here are more useful in somewhat uncontrolled environments. The results we present here show that auctions can be valuable in allocation mechanisms that are designed to guarantee fairness even when we do not have any control over the competing users except for the amount of money we pay

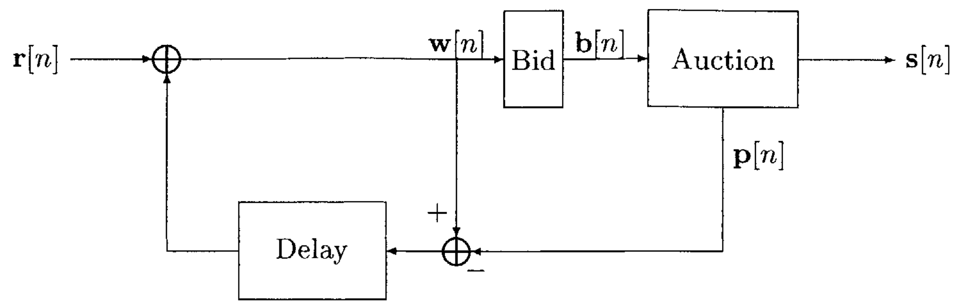


Figure 7-1: Block diagram of auction-based lag evolution equation.

them.

In this chapter, we look at a dynamic scheduling model in which we use an auction mechanism instead of a quantizer as the core static allocation mechanism. We start by describing the general auction-based method in Section 7.1. We continue in Section 7.2 with an investigation of an auction-based dynamic model by showing that a naive approach to using auctions to achieve a fair allocation may not work, and presenting a modified approach that does achieve fairness. In Section 7.5, we allow the users of a shared resource to strategize, and show that this allocates the resource fairly according to predetermined reservation parameters.

7.1 Auction Model

In order to investigate the fairness of this mechanism, we describe a mathematical model for the actual bidding process. The system in [67] was designed to work for large networks of computers on which many processes competed for resources. We would like to investigate a simpler problem, and so the example below works only with a pair of competing processes and a single resource. We will show that one of the simplest versions of this allocation problem does not guarantee fairness under the auction and bidding strategy combination as put forth in [67]. In Section 7.4 we discuss how we might apply the results we present here to a setting with many networked resources.

The auction-based model fits within our previous framework, with the following difference: the schedule during each round differs from the feedback signal, as depicted in Figure 7-1. We use the notation $\mathbf{w}[n]$ for the lag in this case, to be more evocative of the underlying representation of the lag as accumulated *wealth*. The output $\mathbf{b}[n]$ of the bid block in Figure 7-1 is a vector that is at most equal to the input wealth vector. It represents the bids that each user makes in order to try and win the auction. The schedule at each round is $\mathbf{s}[n]$, which simply indicates the winning user with a 1. The feedback signal, or the *price* $\mathbf{p}[n]$, is a vector whose value at each round is based on the schedule and bids for that round. Each user receives an additional amount of income equal to $\mathbf{r}[n]$ at each round n. We can write this relationship as follows:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mathbf{r}[n+1] - \mathbf{p}[n] \ .$$

The same questions that we asked of our dynamic model before can be asked here. In this case, there is more complexity. The price vector is not simply a quantized version of $\mathbf{w}[n]$, but depends on the values of $\mathbf{b}[n]$. However, we can think of the output $\mathbf{s}[n]$ as the output of a quantizer; an auction of a single object generally gives the resource to the highest bidder, equivalent to a ∞-norm based quantizing scheduler.

Given a particular bidding strategy and auction mechanism (mapping from bids to schedule) we would first like to know whether the accumulated wealth is bounded. If it is, that implies that the average price charged to each user per round is equal to the average rate of income for the user. Second, we would like to know if the wealth stays bounded, does the fraction of time a user is scheduled track the fraction of the total income the user receives? For example, if we give 20% of the income to the first user every round we would like to know whether the first user gains the use of the resource 20% of the time or not. If we think of the income as a proportional priority or reservation like $\mathbf{r}[n]$ before, we would like the two fractions to be equal. If this property is guaranteed for a particular bidding strategy and auction mechanism, we will call the mechanism (or strategy for a given mechanism) *fair.*

Definition 7.1 (Auction Fairness) *If user i gets a fraction p_i of the income, either probabilistically or deterministically, then a mechanism (or strategy) is* fair *if*

$$\lim_{M\to\infty} \frac{1}{M} \sum_{n=n_0}^{n_0+M-1} \mathbf{s}[n] = \bar{\mathbf{r}} = p_i \ .$$

This concept of fairness is useful, because a fair mechanism allows us to use the wealth distribution as a method of controlling the resource allocations of the users over periods of time; by varying the allocation of income, we can match the allocation of the resource to a set of reservations we assign to the users. This definition of fairness is essentially the same as the definition of deterministic fairness given in Chapter 3. For a deterministic model, the two definitions are equivalent. If we know that the sequence of wealth vectors is a ergodic process with finite mean and variance, then the above is also equivalent to the definition of stochastic stability given in Chapter 3 as well. If we know that $\mathbf{E}[r_i[n]] = p_i$ or the time average of $r_i[n]$ is equal to p_i, then fairness simply means that over a long period of time the actual fraction of time a user wins is equal to the fraction of income the user receives. However, fairness is not guaranteed if the wealth remains bounded, as was the case with lag. We may have a fair mechanism that charges different prices to different users so that even if the wealth stays bounded, the actual allocation is biased away from the desired value.

The system model in which [67] investigates the use of auctions is very similar to our dynamic mechanism from Section 3.2. In some ways, a single object auction is a quantizer-like operation. It takes a vector of bids, and allocates the object to the user with the largest bid. However, auctions differ from quantizers in a very important way; an auction outputs a price in addition to the actual scheduling decision. By decoupling the price and schedule, auctions allow for more flexibility while also making the mechanism more complex. The flexibility is one reason that auctions are attractive for certain situations. It allows the price to adapt to the bids being made by the users. In Section 3.2 we showed that if the average amount of income $\bar{\mathbf{r}}$ is outside of the feasible region $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$ for our scheduling model, then we can not guarantee boundedness of the state variable, even for very simple constellations. We do not have this problem when we use an auction in place of the quantizer, at least for the simple resource allocation cases considered here. Because the price charged in an auction rises with the bids, if users are accumulating large amounts of wealth (or lag in our earlier model) and bidding proportionately then the price will rise also to counter the increasing wealth. The price allows the auction to adapt to the levels at which the users are bidding. This adaptability is one of the key reasons that auctions have been looked at for complex decentralized computing resource allocation systems such as the Spawn system in [67].

However, along with the flexibility comes more complex behavior. Because the price is a scaled version of the schedule, the bidding strategies and specific auction mechanism determine whether or not the overall dynamic system will guarantee a fair allocation of the resource to the users.

We can relate this to the credit-based scheduling algorithms discussed in Section 3.5. The credit-based algorithms are meant to guarantee that the rate at which credit accumulates for a user is proportional to the fraction of the resource that the user acquires. With an auction mechanism, this is not necessarily the case, as we show in the next section.

Although auction theory concerns a wide variety of different auctions [50], we will concentrate on only two of the most common rules, first- and second-price auctions. A *first-price* auction chooses as the winner the user with the highest bid, and charges a price equal to the winning bid. In contrast, a *second-price* auction charges a price equal to the second-highest bid. Both auctions are commonly used as real-world allocation mechanisms; in this context they are not optimal in any way, but are a good starting point and allow us to investigate and compare with existing results. In Section 7.2, we will show that using a first-price auction mechanism with a naive bidding strategy and constant income does not guarantee fairness even for two users. When we make the income probabilistic, we can guarantee fairness using the same mechanism and bidding strategy. We prove this for a single auction with an arbitrary number of users in Section 7.3 and discuss how we might extend the single resource case to a network of resources in Section 7.4. We then look at both first- and second-price auctions in Section 7.5 and allow the users to use any bidding strategy they wish, while keeping the income constant at each round. We consider both the finite-round and infinite-round cases and show that if the users attempt to maximize the fraction of time they win the auction, the mechanism has an asymptotic guarantee of fairness.

7.2 Simple Auction Allocation Strategies

In [67], each process' bid is determined with an escalating-bid strategy. An escalating-bid strategy is simply one in which each competing process bids its entire amount of funding for each time slice, or $\mathbf{b}[n] = \mathbf{w}[n]$. In this case, the bidding function in Figure 7-1 is simply an identity function.

The question of whether or not the escalating-bid strategy achieves fairness when used with a second-price auction is left open in [67]; only empirical support is presented. We show here that this strategy does not guarantee fairness even over long periods of time even when there are only two users. We restrict our view here to a centralized system in which all users are competing for the same resource; in [67], the model allowed the users to travel around and use one of many networked resources. Although the particular combination of second-price auction and escalating-bid strategy does not guarantee fair allocation of resources for this setup, we show that through a modification of the funding process, we can guarantee fairness on a single processor asymptotically for any number of users.

Consider the following dynamic system, meant to model the evolution of the wealth of two users both competing for the same resource. Although this problem originally came up in the context of processing time on a processor, it might also be used to model behaviors of related systems consisting of independently funded agents competing for an exclusive resource (e.g. data streams accumulating credits for the use of a communication link). Each user, i, receives some amount, $r_i[n]$, of funding at the beginning of every time slice. An auction is held during each time slice, at the end of which the highest of the two bids wins the auction, and the winning bidder must pay the second-highest bid. Denote the bid from user i during time slice n as $b_i[n]$. If $b_1[n] > b_2[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + r_1[n+1] - b_2[n] \\ w_2[n+1] &= w_2[n] + r_2[n+1] \; . \end{aligned}$$

If $b_2[n] \geq b_1[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + r_1[n+1] \\ w_2[n+1] &= w_2[n] + r_2[n+1] - b_1[n] \ . \end{aligned}$$

If $b_2[n] = b_1[n]$, it is not clear which user should win; we will assume for now that user 2 wins the tie, and our simulations show that this affects the results quantitatively, but not fundamentally, even if the tie is broken randomly. The wealth of user i at the beginning of time slice n (before funding arrives) is given by $w_i[n]$.

7.2.1 Escalating Bid – Deterministic Income

The Spawn system in [67] uses as its basic allocation mechanism a second-price auction, and uses an escalating-bid bidding strategy for each user — in other words, each user bets its entire wealth every round. We would like to know whether or not this strategy is fair with respect to the second-price auction.

We can formalize the mechanism as follows: At every time slice, user i bets all of its current wealth, $w_i[n]$, and receives a constant income, $r_i[n] = \rho_i$. In this case, the wealth evolution equations above become: If $b_1[n] > b_2[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + \rho_1 - w_2[n] \\ w_2[n+1] &= w_2[n] + \rho_2 \ . \end{aligned}$$

If $b_2[n] \geq b_1[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + \rho_1 \\ w_2[n+1] &= w_2[n] + \rho_2 - w_1[n] \ . \end{aligned}$$

This is a simple nonlinear dynamic system with two state variables $w_1[n]$ and $w_2[n]$. We can easily plot the state trajectories as a function of n. We will assume for now that both users start out with no stored wealth, i.e. $w_1[0] = w_2[0] = 0$. In this simple case, if we assume ρ_1 and ρ_2 are integers, the system can fail to achieve a fair allocation of the resource.

Example 7.1 (Unfairness) *Set $\rho_1 = 1$ and $\rho_2 = 2$. After a few time slices, $\mathbf{w}[n]$ will be cycling between $(4,4)$ and $(5,2)$. This means that user 2 wins the auction $1/2$ of the time (remember that user 2 wins all ties), and user 1 wins the other $1/2$ of the time. But, user 2 receives $2/3$ of the wealth! So, this bidding strategy does not achieve fairness even for this simple case.*

This example illustrates a basic phenomenon that is present when all of the quantities in the wealth evolution equations are integers (or, by extension, rational numbers). After an initial transient period, the wealth falls into a cyclic pattern, repeating itself after some number of time slices. In the example above, the period of repetition was 2. Ideally, we would like user i to win the auction a fraction f_i of the time, where

$$f_i = \frac{\rho_i}{\sum_j \rho_j} \ .$$

But, this would require that the period of the repetition be a factor of $\sum_j \rho_j$.[1] Our simulations show that this is generally not the case. The example above has a repetition period of 2, but $\sum_j \rho_j = 3$. This behavior makes it impossible for the bidding strategy given above to achieve fairness for an arbitrary funding distribution. Table 7.1 shows a number of actual values of the fraction $\hat{f}_i$ of the

(ρ_1, ρ_2)	Period	$\hat{f}_1$	$\hat{f}_2$	f_1	f_2
(1, 1)	2	1/2	1/2	1/2	1/2
(1, 2)	2	1/2	1/2	1/3	2/3
(1, 3)	3	1/3	2/3	1/4	3/4
(1, 4)	9	2/9	7/9	1/5	4/5
(1, 5)	26	4/26	22/26	1/6	5/6
(1, 6)	19	3/19	16/19	1/7	6/7
(2, 3)	2	1/2	1/2	2/5	3/5
(2, 5)	9	3/9	6/9	2/7	5/7
(3, 4)	7	3/7	4/7	3/7	4/7
(37, 53)	53	22/53	31/53	37/90	53/90
(2, 1)	11	8/11	3/11	2/3	1/3
(3, 1)	4	3/4	1/4	3/4	1/4
(4, 1)	34	29/34	5/34	4/5	1/5
(5, 3)	18	12/18	6/18	5/8	3/8

Table 7.1: Fraction of wins for users 1 and 2.

time that user i wins the auction for various pairs of (ρ_1, ρ_2). The table shows that of all of the small pairs of incomes ($\rho_i \leq 5$) where $\rho_1 \leq \rho_2$, only the pair $(3, 4)$ achieves fairness exactly. All of the others, however, are as close to fair as is possible given the period of repetition (e.g. for $(1, 5)$, 4/26 and 22/26 are the closest fractions to 1/6 and 5/6, respectively, with a denominator of 26). This is not the case for the income pairs where $\rho_1 > \rho_2$. In particular, for $(4, 1)$ the best fractions would be 27/34 and 7/34. Figure 7-2 show the behavior of the system for income pairs $(1, 3)$ and $(3, 4)$.

This repetition behavior of the wealth trajectories can cause problems for an unsuspecting system implementor. Even for a simple wealth distribution of $(\rho_1, \rho_2) = (1, 2)$, the system would have the same performance as if the division were even. Although the desired effect is for one of the users to use the resource twice as often as the other, the end result is that they both use the resource equally often. In fact, because we are allowing the second user to win ties, we are even giving it an unfair advantage, but still the resource is split evenly. If a system using this auction model is implemented, and at some time there are two competing users, where one has twice the wealth as the other coming in at every round, the system will fail to provide priority to the wealthier user.

Earlier, we mentioned that the winner of ties did not make an important difference in the behavior of the system. If the first user were to win all ties, then by symmetry if $\rho_1 \leq \rho_2$ we can just switch $r_1[n]$ with $r_2[n]$ and we will get results similar to the bottom section of Table 7.1. If we were to break the tie probabilistically it turns out that the unfair example given earlier is fair on average, but only if we use probabilities equal to f_i. If we use other probabilities the mechanism is not fair. Also, for other choices of income, choosing probabilities equal to f_i does not always work.

7.2.2 Escalating Bid – Probabilistic Income

Now, consider the following modification of the previous mechanism: At every time slice, user i bets all of its current wealth, $w_i[n]$. However, now user i receives an income of 1 at each time step

[1]Or that it be a multiple $\alpha \sum_j \rho_j$, and the number of times that user i wins be a multiple of α.

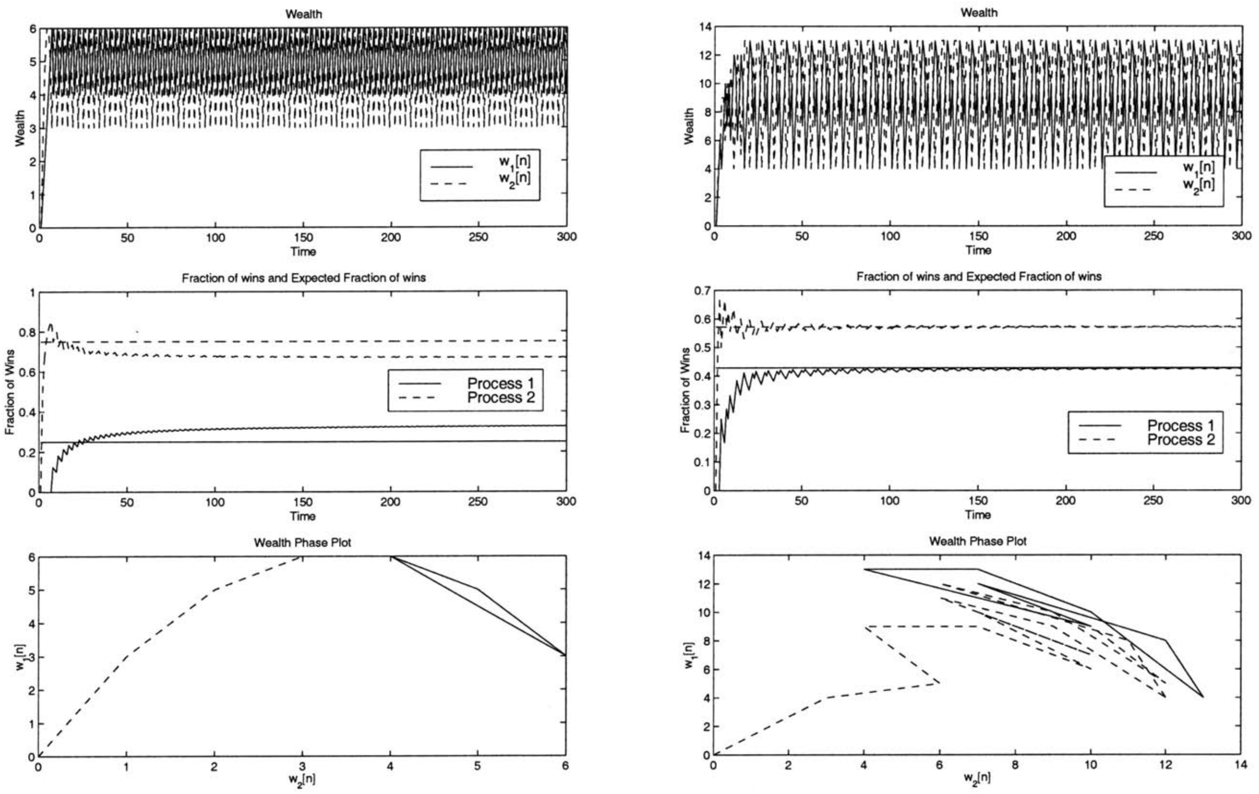


Figure 7-2: Plots showing the wealth and fraction of wins for both users with $\rho = (1, 3)$ and $\rho = (3, 4)$, respectively. In the phase plot, the solid line shows the final periodic repetition, while the dotted line shows the initial transient.

with probability p_i, $p_1 + p_2 = 1$. In this case, the wealth evolution equations are the same, but the income ρ_i is now a Bernoulli random variable; it takes the value 1 with probability p_i, and 0 otherwise. We also make one more change: a tie (both users bid the same amount) is broken probabilistically (to make the analysis simpler). If $b_1[n] > b_2[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + \rho_1 - w_2[n] \\ w_2[n+1] &= w_2[n] + \rho_2 \ . \end{aligned}$$

If $b_2[n] > b_1[n]$,

$$\begin{aligned} w_1[n+1] &= w_1[n] + \rho_1 \\ w_2[n+1] &= w_2[n] + \rho_2 - w_1[n] \ . \end{aligned}$$

With probability p_i, user i wins the tie. This system does not have the same unfairness problems as the first system, and is still implementable using only integers.

Theorem 7.1 (2-User Fairness) *The escalating-bid strategy is a fair bidding strategy for allocating a shared resource between two competing users when the income is distributed probabilistically among the users.*

Proof Idea: Consider a time slice m at which both users have first received two units of income (this must happen with probability 1). If we consider the Markov chain described by the above equations, where each state in the Markov chain is labeled by a possible wealth pair (w_1, w_2), it turns out to have one recurrent class of two states: $(2, 1)$ and $(1, 2)$ (see Figure 7-3). So, after an initial transient period, we are left with a two-state Markov chain. The limiting probabilities for this chain are p_1 for $(2, 1)$ and p_2 for $(1, 2)$. Thus, in the long run, the fraction of time we spend in state $(2, 1)$ corresponds to the fraction of time that user 1 wins the auction, or p_1. The same is true for user 2, and so the system is fair. ■

This theorem will follow as a special case of the theorem proved in the next section.

7.3 One Auction, Many Users

It is fairly easy to analyze the above case for two competing users completely and show that the limiting probabilities of the states in the recurrent class are equal to the distribution probabilities of the wealth, and that the recurrent class is the only possible recurrent class. However, when the number of competing users is larger than two, the mechanism is still fair but the proof of fairness becomes more complicated.

Example 7.2 (Three Users) *Consider a case where we have three users, 1, 2, and 3. We can write out similar equations to the wealth evolution equations for two users. If we define the wealth vector as (w_1, w_2, w_3), then there are two recurrent classes of states in the Markov chain defined by the evolution equations:*

$$\{(2,1,1), (1,2,1), (1,1,2)\}$$
$$\textit{and}$$
$$\{(2,1,0), (1,2,0), (2,0,1), (1,0,2), (0,2,1), (0,1,2), (1,1,1)\}$$

Either of these classes is reachable starting from $(0, 0, 0)$. Luckily, it turns out that both of the classes have limiting probabilities that correspond with fair allocation of the resource to the users according to the distribution probabilities, p_1, p_2, and p_3, as will be shown below.

In the work that follows, we show that Theorem 7.1 can be extended to apply to the case with more than two users competing for a single resource.

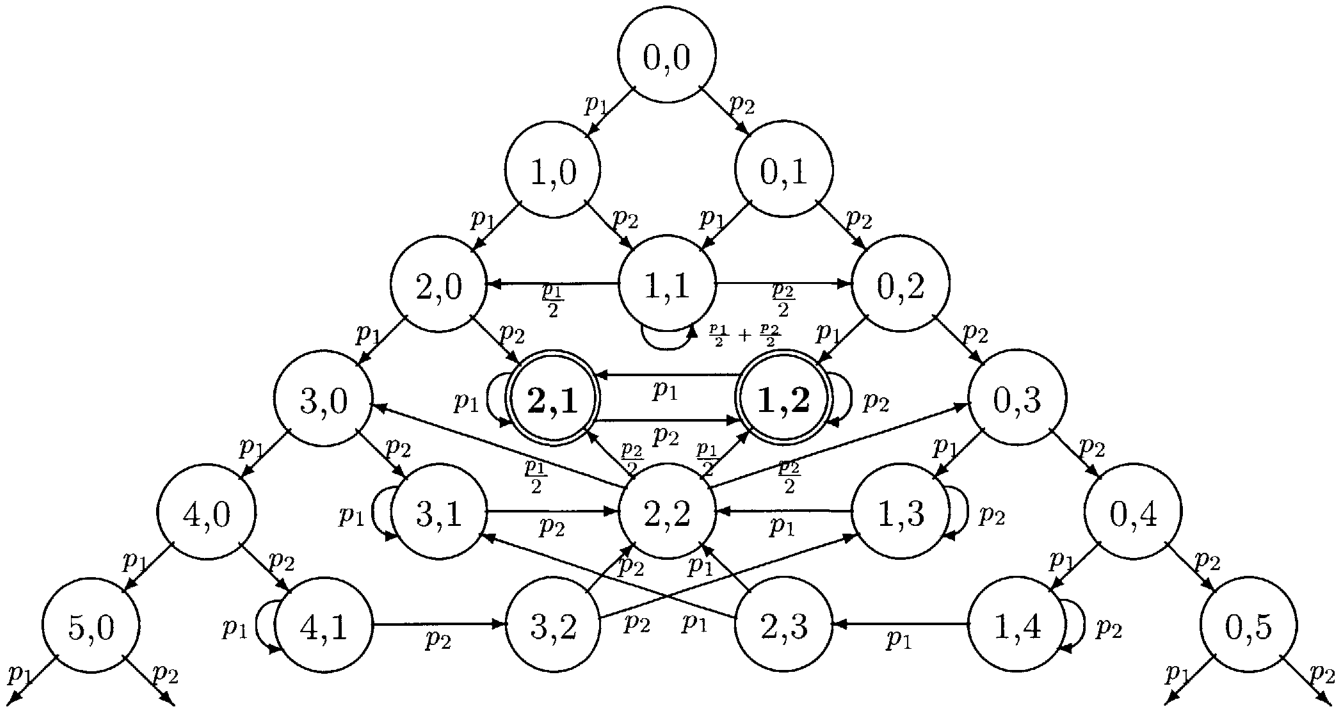


Figure 7-3: Markov chain defined by the two user auction. Each state is labeled with the wealth of the two users.

7.3.1 Model

We now present a wealth evolution model that allows many competing users. In this model, N independent users compete for the use of a single resource. The competition is in the form of a sequence of rounds, where each user may place a bid for the use of the resource during the subsequent round. The user that places the highest bid wins the use of the resource, and must pay an amount equal to the second-highest bid. After the winner is chosen, the resource allocates a single unit of income to one of the competing users according to probabilities assigned to each user as in Section 7.2.2. As before, the only bidding strategy we will consider is the escalating-bid strategy, where each user bids all of its accumulated wealth in each auction (i.e. $b_i[n] = w_i[n]$). The dynamic wealth evolution equations can be captured as follows, for the three possible cases for some user i. If $b_i[n]$ is the unique maximum bid at round n and $b_j[n] = w_j[n]$ is the second-highest bid,

$$w_i[n+1] = w_i[n] + \rho_i - w_j[n] \; .$$

If $b_i[n]$ is less than the maximum bid at round n,

$$w_i[n+1] = w_i[n] + \rho_i \; .$$

If $b_i[n]$ is equal to the maximum bid, but not unique, the winner of the auction is chosen randomly, uniformly among the highest bidders.[2] As before, ρ_i is a Bernoulli random variable, which equals

[2]This tie breaker is not sensitive to the actual probabilities assigned to each of the highest bidders, so the analysis below assumes they are uniform.

1 with probability p_i, and 0 otherwise. The fairness of this mechanism can be interpreted in the following sense. If $p_j/p_i = \alpha > 1$, then user j receives (on average) α times as much income as user i, so user j has a higher priority than user i. The initial values of the $w_i[n]$ are all 0.

7.3.2 Results

This section will establish the fact that if we implement this auction model, with N competing users, then in an expected finite amount of time we will enter a class of states within which the expected fraction of the auctions that a user wins is equal to the expected fraction of the wealth that the user receives from that point on. This implies that the mechanism is fair as we have defined fairness.

In order to present the fairness results, we first need to make the following definitions.

Definition 7.2 (Wealth Chain) $\mathbf{w}[n]$ *is the Markov chain where the state* $\mathbf{w}[n] = (w_1[n], w_2[n], \ldots, w_N[n])$, *and the transition probabilities from* $\mathbf{w}[n]$ *to* $\mathbf{w}[n+1]$ *are defined by the wealth evolution equations given above. The chain* $\mathbf{w}[n]$ *evolves over the space of nonnegative integer vectors of length* N, $\mathbb{Z}_+^N$.

An *essential class* $C \in \mathbb{Z}_+^N$ is a set of states in $\mathbf{w}[n]$ such that every state c in the class *communicates* with every other state in C and that no state in the class leads to any state outside of C with positive probability [9]. A state a communicates with a state b if b can be reached with finite probability from a in some number of steps, and the same for a from b. This implies that the probability of moving between a state in C and a state outside of C is 0. We can think of an essential class as a set of states that "captures" the chain; whenever the chain enters the class, it never leaves. So, if we can show that the chain will enter an essential class in finite number of rounds we only have to analyze the essential classes to determine the long-term fairness behavior of the chain.

The three key parts to the fairness proof are, (i), showing that the wealth chain consists of a finite collection of finite-sized essential classes with the remaining states being inessential and that the essential classes are ergodic, (ii), showing that the wealth chain enters one of these ergodic classes in an expected finite amount of time, and (iii), showing that within each ergodic class the stationary probabilities imply the allocation of the resource is fair.

In our wealth chain $\mathbf{w}[n]$ with N competing users, there are $N-1$ essential classes of states we will call $C(m)$, for $0 \leq m < N-1$. These classes are important because, as we will show, the chain $\mathbf{w}[n]$ must eventually end up in one of them and remain there.

Definition 7.3 (Fair Classes) *A* fair class $C(m) \in \mathbb{Z}_+^N$, *with* $0 \leq m < N-1$, *is a subset of the states of* $\mathbf{w}[n]$ *such that the one of the following two properties hold for each element* $\mathbf{w} = (w_1, w_2, \ldots, w_N) \in C(m)$ *(for* $m \geq 1$, $\mathbf{k}(m) = (k_1, k_2, \ldots, k_m)$, *where each* k_i *is a user index between* 1 *and* N*):*

(i) m *of the values* w_i *are equal to* 0, *one of the* w_i*'s is equal to* 2, *and the rest (at least one) are equal to* 1. *In this case, we use the notation* $\mathbf{w} = \omega_i^{\mathbf{k}(m)}$ *(*$i \notin \mathbf{k}(m)$*) where the wealth of user* i *is* 2, *and the wealth of each of the users* $k_j(m)$, $1 \leq j \leq m$ *is equal to* 0.

(ii) $m-1$ *(for* $m > 0$*) of the values* w_i *are equal to* 0, *and the rest (at least one) are equal to* 1. *We use the notation* $\mathbf{w} = \omega_0^{\mathbf{k}(m-1)}$ *(*$i \notin \mathbf{k}(m-1)$*) where the wealth of each of the users* k_j, $1 \leq j \leq m-1$ *is equal to* 0 *and the wealth of the rest of the users is equal to* 1. *(If* $m = 1$, *then there is only one of these states, denoted by* ω_0.*)*

We start by showing that every class $C(m)$ is an essential class and that there are no other essential classes in $\mathbf{w}[n]$.

Lemma 7.1 (Existence and Completeness of Essential Classes) *Every fair class $C(m)$, for all $N \geq 2$ and $0 \leq m < N-1$, is an essential class and there are no other essential classes.*

The essential classes we consider here are aperiodic, meaning that we can be in any state in the class at any time with positive probability. This property follows from the fact that any state in a set $C(m)$ has a nonzero probability of leading to itself. This means that these fair classes $C(m)$ are ergodic classes because they are essential, aperiodic and finite [18]. The ergodicity of one of these classes implies that once the chain enters the class, there is a unique limiting probability distribution over the states in the class and that over time the time-average occupancy of a particular state converges to the limiting probability of being in that state [18]. This allows us to determine these probabilities and show that they imply long-term fairness within each class.

Lemma 7.2 (Inevitability of Fair Classes) *If n_T is the first round during which $\mathbf{w}[n_T] \in C(m)$ for some m, then $\mathbf{E}[n_T] < \infty$ for any given $\mathbf{w}[0]$.*

With the above lemmas, we know that after some expected finite number of rounds, $\mathbf{w}[n]$ will end up in one of the fair classes $C(m)$. This says that the dynamics of $\mathbf{w}[n]$ for very large values of n are determined by the fair classes. So, if we want to know whether or not the system is asymptotically fair, we only need to analyze the fair classes.

Limiting Probabilities

In this section we will assume that we have already entered a fair class $C(m)$; we will concentrate on showing that the wealth chain is fair when the state space is restricted to $C(m)$.

We have defined fairness for this mechanism as being the property that the time-average resource allocation that user i gets is proportional to the probability p_i that the user gets paid at each round, or the time-average fraction of the income that the user is paid. Because each fair class $C(m)$ is ergodic, the time-average allocation for user i is equal to the steady-state, or limiting, probability that user i wins the auction. Label the limiting probabilities of the states in a fair class $C(m)$ as follows:

(i) $\pi_i^{\mathbf{k}(m)}$ ($i \notin \mathbf{k}(m)$) is the limiting probability of state $\omega_i^{\mathbf{k}(m)}$ given that we are in class $C(m)$.

(ii) $\pi_0^{\mathbf{k}(m-1)}$ ($i \notin \mathbf{k}(m-1)$) is the limiting probability of state $\omega_0^{\mathbf{k}(m)}$ given that we are in class $C(m)$.

The limiting probability of state $\omega \in C(m)$ corresponds to the probability that the wealth chain $\mathbf{w}[n] = \omega$ at some large value of n. Once we know these limiting probabilities, we can compute the steady-state probability that user i wins by adding the state probabilities over all states in which user i wins the auction (either because it has the largest wealth, or because it wins the probabilistic tie-breaker). It is obvious that when in state $\omega_i^{\mathbf{k}(m)}$, user i will always win the auction because by the definition of $\omega_i^{\mathbf{k}(m)}$, user i has 2 dollars and all the other users have less wealth. In state $\omega_0^{\mathbf{k}(m-1)}$, the winner of the auction is chosen randomly from the users j such that $j \notin \mathbf{k}(m-1)$. When computing the steady-state probability that a user i wins, we have to consider these two possibilities. We define a few other quantities which are convenient when dealing with these probabilities:

(iii) $r_m^{\mathbf{k}(m)} = \prod_{j \notin \mathbf{k}(m)} p_j$ is the product of all the probabilities p_j of users that are not represented in the set $\mathbf{k}(m)$.

(iv) $r_m = \sum_{\mathbf{k}(m)} r_m^{\mathbf{k}(m)}$ is the sum over all possible values of $\mathbf{k}(m)$ of the above products.

Given these definitions, we can now determine the limiting probabilities of the states in $C(m)$ and from them deduce the steady-state probability that user i wins the auction.

Lemma 7.3 (Limiting Probabilities) *Given* $\mathbf{w}[n]$ *is in a fair class* $C(m)$*, the limiting probabilities of the states in the class are given by*

$$\begin{aligned} \pi_i^{\mathbf{k}(m)} &= p_i \cdot \frac{r_m^{\mathbf{k}(m)}}{r_m} \\ \pi_0^{\mathbf{k}(m-1)} &= (N-(m-1)) \cdot \frac{r_{m-1}^{\mathbf{k}(m-1)}}{r_m} \end{aligned}$$

Lemma 7.4 (Winning Probabilities) *Given* $\mathbf{w}[n]$ *is in a fair class* $C(m)$*, the probability that user* i *wins an auction at some arbitrary time* n *is equal to* p_i.

This last lemma follows from Lemma 7.3 by simply adding up all of the probabilities for which user i wins the next auction given that we observe $\mathbf{w}[n]$ in some state in $C(m)$. Before we present the final theorem that establishes the fairness of the overall auction mechanism, consider the following simple example that illustrates the implications of the previous lemmas.

Example 7.3 (3-User Chain with $m = 1$) *Recall the second recurrent class mentioned in Example 7.2 above, with 7 states. This class is denoted* $C_3(1)$*. For this class, we can write down the some of the states using the notation just presented:*

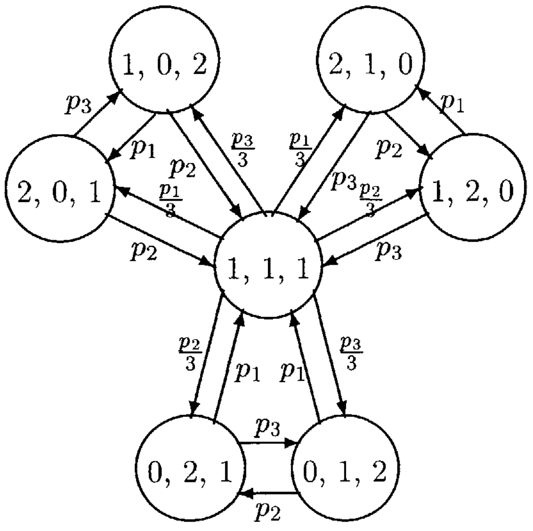
$$\begin{aligned} \omega_1^{\{2\}} &= (2,0,1) \\ \omega_1^{\{3\}} &= (2,1,0) \\ \omega_0 &= (1,1,1) \end{aligned}$$

We can also draw a picture of the class, including all of the transition probabilities, shown in Figure 7-4. The quantities $r_1^{\mathbf{k}(m)}$ *and* r_1 *are*

$$\begin{aligned} r_1^{\{1\}} &= p_2p_3 \\ r_1^{\{2\}} &= p_1p_3 \\ r_1^{\{3\}} &= p_1p_2 \\ r_1 &= p_2p_3 + p_1p_3 + p_1p_2 \\ r_0 &= p_1p_2p_3 \; . \end{aligned}$$

And finally, the limiting probabilities for the three states listed above are

$$\begin{aligned} \pi_1^{\{2\}} &= p_1 \cdot \frac{p_1p_3}{p_2p_3 + p_1p_3 + p_1p_2} \\ \pi_1^{\{3\}} &= p_1 \cdot \frac{p_1p_2}{p_2p_3 + p_1p_3 + p_1p_2} \\ \pi_0 &= 3 \cdot \frac{p_1p_2p_3}{p_2p_3 + p_1p_3 + p_1p_2} \; . \end{aligned}$$



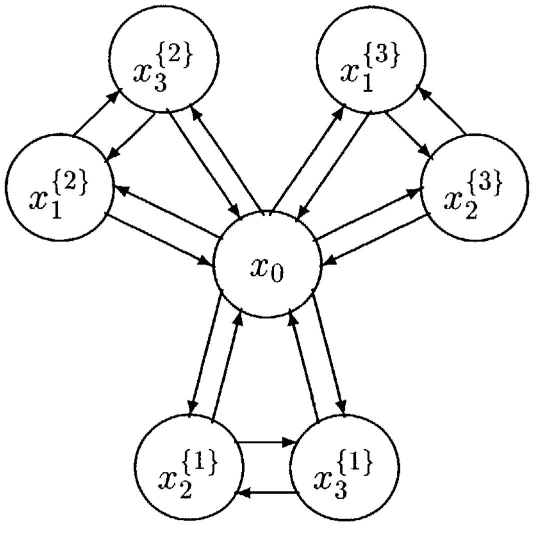


Figure 7-4: Recurrent class $C_3(1)$ for a 3-user auction. The left figure is labeled with the each user's wealth in each state; the right figure uses our notation to label each state.

We can also check for fairness by adding the probabilities that user 1 *wins an auction:*

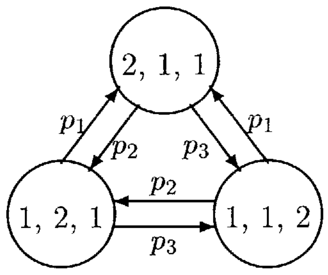
$$\begin{aligned}
\text{Prob[user 1 wins an auction]} &= \pi_1^{\{2\}} + \pi_1^{\{3\}} + \frac{1}{3}\pi_0 \\
&= \frac{p_1^2 p_3 + p_1^2 p_2 + p_1 p_2 p_3}{p_2 p_3 + p_1 p_3 + p_1 p_2} \\
&= \frac{p_1(p_1 p_3 + p_1 p_2 + p_2 p_3)}{p_2 p_3 + p_1 p_3 + p_1 p_2} \\
&= p_1 \; .
\end{aligned}$$

Theorem 7.2 (**N-User Fairness**) *Given the auction model described above, the escalating-bid strategy is a fair strategy for allocating a shared resource between N competing users when the income is distributed probabilistically among the users.*

Proof: With the previous lemmas, the proof follows easily for $m \geq 1$. We see that the expected time until the chain enters a fair class is finite and so the time-average allocation that a user will get is determined by the behavior in the fair classes over long time periods. This allocation is equal to the winning probabilities from Lemma 7.4 by the ergodic property of $C(m)$. We can write this because we know that $n_\tau < \infty$ with probability 1, so if $n_0 \leq n_\tau$ (if not, the fairness follows easily):

$$\lim_{M\to\infty} \frac{1}{M} \sum_{n=n_0}^{n_0+M-1} \mathrm{s}[n] = \lim_{M\to\infty} \frac{1}{M} \sum_{n=n_0}^{n_\tau} \mathrm{s}[n] + \frac{1}{M} \sum_{n=n_\tau}^{n_0+M-1} \mathrm{s}[n] = \lim_{M\to\infty} \frac{1}{M} \sum_{n=n_\tau}^{n_0+M-1} \mathrm{s}[n] = p_i$$

with probability 1. The last step follows because after n_τ, the chain is in one of the ergodic fair classes. The only case left is when $m = 0$; an example for $N = 3$ is shown in Figure 7-5. For this case, $C(m)$ has N states, which we can call ω_i. It is easy to see that the probability of going from state ω_j to state ω_i is just p_i, because all that has to happen is that user i receive the unit of wealth



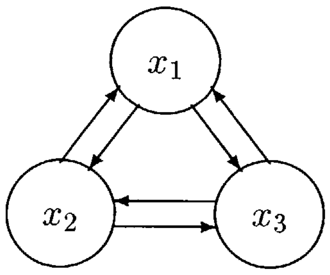


Figure 7-5: Recurrent class $C_3(0)$ for a 3-user auction. The left figure is labeled with the each user's wealth in each state; the right figure uses our notation to label each state.

after j wins the auction. The limiting probability for state ω_i, $\pi_i = p_i$. This can easily be verified:

$$\begin{aligned} \sum_j p_i \pi_j &= \pi_i \\ \sum_j p_i p_j &= p_i \\ p_i \sum_j p_j &= p_i \ . \end{aligned}$$

Fairness is also obvious; the only time user i wins an auction is in state ω_i, so the probability that i wins an auction is p_i.

Thus, because the Markov chain $\mathbf{w}[n]$ reaches some $C(m)$ in an (expected) finite time, and the probability that user i wins an auction at some arbitrary time once we are in $C(m)$ is equal to the fraction of the wealth that is allocated to user i (the fraction is p_i) for any $C(m)$, the overall auction model for N users competing for a single resource results in a fair allocation of the resource according to the probabilities p_i as n, the number of auctions, gets very large. ■

7.4 Many Resources

We have now shown that if we replace the constant income model used in [67] and instead use the probabilistic income model, we can guarantee fairness at each resource independently. We have concentrated only on the problem of allocating a single resource among competing users. We could also extend this to incorporate the more general allocation problems considered earlier in this thesis and in [67]. The original model in [67] allowed the users to move between a number of resources to try and find a resource that was not overloaded.

If we were to make the simple extension to multiple resources to be allocated among all of the users at each round, we would have to define a more general auction that determines the allocation of the objects and prices charged to the users. There are different possible ways of doing this. For example, the second-price auction can be extended to multiple objects by simply allocating the objects singly to the top bidders, and charging a price equal to the highest bid that did not win an object [16].

In [67], this is taken in a different direction, and the objects are distributed so that the users must pick one of many objects to bid on. The problem is to distribute the users among the resources to balance the resource contention and achieve overall fairness for the network of resources. This might be done in a centralized or decentralized way. In [67], the competing users were seen as independent entities, so this overall fairness needed to be achieved using a decentralized method. We present a possible way of approaching this problem as a way of indicating how the overall system might guarantee fairness.

Let us assume that each user i has a priority r_i associated with it. We have M identical resources available for N users. Each user can only use a single resource at a time, and would like to use as much of the resource as possible. If $N \leq M$, it is obvious that we can assign each user to a different resource and each user then has the use of an entire resource. So, contention only arises when $N > M$. Call the set of users competing at resource j during some round $\mu(j)$. In order to use our probabilistic income model, we define ρ_i for a user i at resource j as

$$\rho_i = \frac{r_i}{\sum_{k \in \mu(j)} r_k} = \frac{r_i}{R(j)} .$$

Thus, the portion of the resource that user i receives, ρ_i, is determined by the other users that are competing at the same resource. The obvious goal for each user is to find a resource with the smallest $R(j)$ in order to maximize ρ_i. The overall system goal is then to achieve a distribution of users that minimizes the spread of $R(j)$ over the resources and thus match the users' priorities. The resource that a user i is using is called $\sigma(i)$; the collection of these assignments, or the *allocation* is Σ. We can view the overall optimization objective as computing

$$\Sigma^* = \arg\min_{\Sigma}[\max_{j} R(j) - \min_{j} R(j)] = \arg\min_{\Sigma}[\max_{j} \bar{R}(\Sigma)] .$$

Just as in Chapter 5, computing the solution this problem is NP-hard; given 2 equally-sized resources, an algorithm to solve this problem could also solve the partition problem.

Although it is not easy to solve this problem exactly, we can still try to approximate the answer and use the decentralized nature of the problem to help make it more tractable. If we allow each of the users to make their own decisions independently, the parallelism will decrease the time complexity of determining a good solution. The mechanism for determining a good assignment in [67] used prices as signals that the users could use to help make decisions. If a neighboring resource was currently charging a lower price, a user would move to that resource. It is clear that in equilibrium, the price charged at a resource would be equal to $R(j)$ for resource j on average. It makes sense, then, to use $R(j)$ as a method of ranking resources. User i would compare $R(\sigma(i)) - r_i$ with $R(j)$ for all resources j that can be observed when deciding whether or not to move. The smallest would offer the largest fraction of the resource if the allocation mechanism is fair. This simple distributed algorithm does not necessarily converge to the optimal solution, and does not necessarily converge at all; it may end up cycling. However, if $\bar{R}(\Sigma)$ is small in equilibrium, a probabilistic income model will guarantee local fairness and thus approximate overall fairness.

7.5 Strategic Auction Mechanisms

In this section, we again investigate a scenario in which a number of users compete to gain access to a single shared resource over a number of rounds. This differs from the previous discussion because we will allow the users to choose their bids according to arbitrary rules. In other words, the bidding function as shown in Figure 7-1 is not the identity function. Our goal is to investigate whether the auction mechanisms we considered earlier are fair when we use non-naive bidding policies.

At the beginning of each round the resource is allocated to one of the users using a sealed-bid first- or second-price auction. Each competing user's sole goal is to maximize the fraction of time it can gain the resource. The users receive income and must use this money to bid on the resource at each round. This setup differs from the traditional value-based auctions, where the competing users are assumed to have some value for the resource, and bid on the object in order to maximize a profit function. Although our setup does assume utility-maximizing users, the utility to the users is not profit, but amount of resource.[3]

We begin by investigating a repeated auction that has a finite number of rounds, and a fixed amount of starting wealth with no income; we will show that the first- and second-price auctions achieve approximately fair allocations of the rounds. Then we extend the fairness result to an infinite round model with constant income at each round. We would like to know whether or not these simple and commonly-used mechanisms are able to ensure fair allocation of the resource. We define fair in this case to mean that the ratio between the number of rounds a pair of users wins is approximately equal to the ratio between the amount of starting wealth, or income, for the two users.

In [31], a similar problem is considered, where a number of users try to guarantee winning some fraction of a number of resources, but the auctions for each of the resources are performed simultaneously. A randomized bidding strategy is developed that guarantees a lower bound on the expected number of resources that an user using the strategy will win. Their analysis also leads to them noting that for the case when all users begin with the same amount of money, if the auctions are performed sequentially (as in our model), and if the total number of resources available is divisible by the number of users, every user can guarantee an equal share of the resources. This is a special case of Theorem 7.7.

Our main results establish that the first- and second-price auctions, when used for repeated resource allocation, exhibit a form of approximate fairness when a competing user uses a strategy that maximizes the number of resources it can guarantee. The results depend upon the fact that each user is aware of the other users' stored wealth at each round, as well as the winner of each round and the corresponding price charged. For the case when we have only two competing users, we give a complete characterization of the fairness (i.e. for N rounds, we show how many each user wins for all possible initial wealths), and provide bidding strategies that guarantee these numbers of wins for each user. When there are many users, we give a similar lower bound on the number of wins that is within 1 of the desired number, but the upper bound is much looser except for a special case. Finally, we provide similar results for the infinite horizon case, when the users are trying to maximize the fraction of times they can guarantee wins over an unbounded number of rounds. For this case, the income is provided to the users at a constant rate during each round.

7.5.1 Finite Rounds Model

First, let us assume that there are N users and a single shared resource. Name the users $1, 2, \dots, N$. User i receives a lump sum W_i before the beginning of the first round. The allocation of the resource is performed in R rounds. Each round consists of an auction that determines which user is able to use the resource, followed by the time slice that is allocated for the actual usage of the resource. The users engage in a sequence of R of these rounds. Each auction is run independently of the others; the only difference is the amount of money each user has available to bid. We will generally denote the bid by an user for round n as $b_i[n]$, or simply b_i if the round number is understood. Each user can bid between zero and its current wealth at each round. After each auction, the highest

[3]Implicitly this means that any wealth left at the end of the rounds is useless to the users. This makes sense in cases when wealth is not tied to real wealth, but to virtual credit that can be taken away at the end of the allocation rounds.

bidder is declared the winner and must pay some price to be able to use the resource during that round. The price is determined by the type of auction used. Here we will consider two possibilities: the first-price auction, in which the price is equal to the highest bid; and the second-price auction, in which the price is equal to the second-highest bid.

As we stated before, the basic goal of each of the users is to maximize the number of wins that it can guarantee, R_i^*, no matter what the other users bidding strategies are. This choice of an optimization criterion implies that the sum of these guarantees over all of the users is equal to the total number of rounds R when $N = 2$ because the mechanism is equivalent to a zero-sum game [56]. This model can be characterized as an extensive game with perfect information [56]. In game-theoretic terminology, we are searching for a strategy profile B that forms a sub-game perfect equilibrium (SPE). Finding a SPE involves a dynamic programming iteration described below. In many extensive games, although a SPE exists for every extensive game, it is difficult to find a SPE especially in this case when the possible strategies are numerous. Here we show below that optimal strategies can be characterized and verified relatively easily.

We denote the bidding strategy for user i by B_i and the collection of strategies for the other users as B_{-i}. The current state, or wealth is $\mathbf{w}[n] = (w_1[n], w_2[n], \ldots, w_N[n])$. A bidding strategy is a collection of bidding functions $\beta_i(n, \mathbf{w})$, one for each round, that produce bids as a function of the current wealth of all of the users. The bidding strategy for all users is $B = (B_1, B_2, \ldots, B_N)$, and consists of bidding functions $\beta(n, \mathbf{w}) = (\beta_1(n, \mathbf{w}), \beta_2(n, \mathbf{w}), \ldots, \beta_N(n, \mathbf{w}))$ (we also use the notation $\mathbf{b}[n] = \beta(n, \mathbf{w})$ when the amount of wealth $\mathbf{w}$ is understood). The price that is charged at time n is $p[n]$, and depends on the type of auction used. In the first-price auction $p[n] = \max_j \beta_j(n, \mathbf{w}[n])$, or the largest bid, and in the second-price auction, $p[n]$ is equal to the second largest bid. Given a bidding strategy B_i, as the rounds progress (we label the first round by R, or by the number of rounds remaining) the wealth of the users evolves as

$$\begin{aligned}\mathbf{w}[n-1] &= \mathbf{w}[n] - p[n] \cdot \sigma(\beta(n, \mathbf{w}[n])) \\ &= \mathbf{w}[n] - \mathbf{p}[n] \;.\end{aligned}$$

For all users j, $\sigma_j(\cdot) = 0$ except for the winner of the auction, k, for which $\sigma_k(\cdot) = 1$. So, the equation above tells us that after each round the winner of the auction at that round must pay $p[n]$, and the rest of the users maintain their current wealth. Given this definition of the wealth evolution, user i can guarantee R_{B_i} wins in R rounds when user j begins with wealth W_j ($\mathbf{W} = (W_1, W_2, \ldots, W_N)$):

$$R_{B_i}(R, \mathbf{W}) = \min_{B_{-i}} \sum_{n=1}^{R} \sigma_i(\beta(n, \mathbf{w}[n])) \;.$$

For user i to maximize the number of guaranteed wins in R rounds, he needs to determine a strategy B_i^* such that

$$R_{B_i^*}(R, \mathbf{W}) = \max_{B_i} R_{B_i}(R, \mathbf{W}) = \max_{B_i} \min_{B_{-i}} \sum_{n=1}^{R} \sigma_i(\beta(n, \mathbf{w}[n])) \;.$$

For simplicity, we will define $R_i^* = R_{B_i^*}$. Because of the sequential nature of the problem, we can rewrite this as

$$R_i^*(R, \mathbf{W}) = R_i^*(R, \mathbf{w}(R)) = \max_{b_i(R)} \min_{b_{-i}(R)} [\sigma_i(\mathbf{b}(R)) + R_i^*(R-1, \mathbf{w}[R-1]] \;.$$

And more generally, when there are n rounds remaining

$$R_i^*(n, \mathbf{w}[n]) = \max_{b_i[n]} \min_{b_{-i}[n]} [\sigma_i(\mathbf{b}[n]) + R_i^*(n-1, \mathbf{w}[n-1]] \;.$$

Finally, we know that $R_i^*(0, \mathbf{w}[0]) = 0$ for all i. Using these definitions, we can iteratively solve for $R_i^*(R, \mathbf{W})$ starting from $R_i^*(0, \mathbf{w}[0])$. The only other detail left to be specified is how to break ties. For the finite rounds model, we will assume that the lowest-numbered user with the highest bid wins the auction.

Two users

We first focus on the case when there are only two users ($N = 2$). At each round user i tries to maximize the minimum number of wins that it can guarantee from now until the final round by choosing its bid $b_i(\cdot)$ appropriately (the bid can be anything from 0 to the user's current wealth). From our model above, we are trying to compute the $R_1^*(R, \mathbf{W})$ and $R_2^*(R, \mathbf{W})$ using

$$\begin{aligned} R_1^*(n, \mathbf{w}[n]) &= \max_{b_1[n]} \min_{b_2[n]} [\sigma_1(n) + R_1^*(n-1, \mathbf{w}[n-1])] \\ R_2^*(n, \mathbf{w}[n]) &= \max_{b_2[n]} \min_{b_1[n]} [\sigma_2(n) + R_2^*(n-1, \mathbf{w}[n-1])] \\ \mathbf{w}[n-1] &= \mathbf{w}[n] - p[n] \cdot \sigma(\beta(n, \mathbf{w}[n])) \ . \end{aligned}$$

By these definitions,

$$R_1^*(R, \mathbf{W}) + R_2^*(R, \mathbf{W}) \leq R \ . \tag{7.1}$$

However, because this model (when looked at as a strategic game) is a zero-sum game we expect that

$$R_1^*(R, \mathbf{W}) + R_2^*(R, \mathbf{W}) = R \ . \tag{7.2}$$

Before the theoretical results are shown, we present a few observations based on solving the iteration above to find $R_1^*(R, \mathbf{W})$ and $R_2^*(R, \mathbf{W})$. If we solve this problem for small W_1, W_2 and R, the values have the following properties:

- $R_1^*(R, \mathbf{W})$ is shown in Figure 7-6, for $0 \leq W_1, W_2 \leq 20$ and $R = 8$ and with a second-price auction. The dotted lines indicate lines of slope $m/(R-m)$, $m = 1, \ldots, R-1$. Each of these lines falls roughly within the region in which $R_1^* = m$; this indicates a sort of proportional fairness of the mechanism, in which ratios of starting wealth that are close to $m/(R-m)$ achieve a win ratio of $m/(R-m)$.

- For each possible set of initial wealth parameters, there are a number of optimal strategies that each can play (usually multiple choices at each round). Interestingly, among the optimal strategies that have been examined, there is always a "conceding" strategy in which neither user ends up losing any money; at least one user bids zero in every round, and so the other user wins by default and does not have to pay. The optimal strategy we use in the proofs below is not of this form.

So far we have not specified the actual auction mechanism used for the allocation. In the next two sections, we present fairness results for two common mechanisms: the first- and second-price auctions.

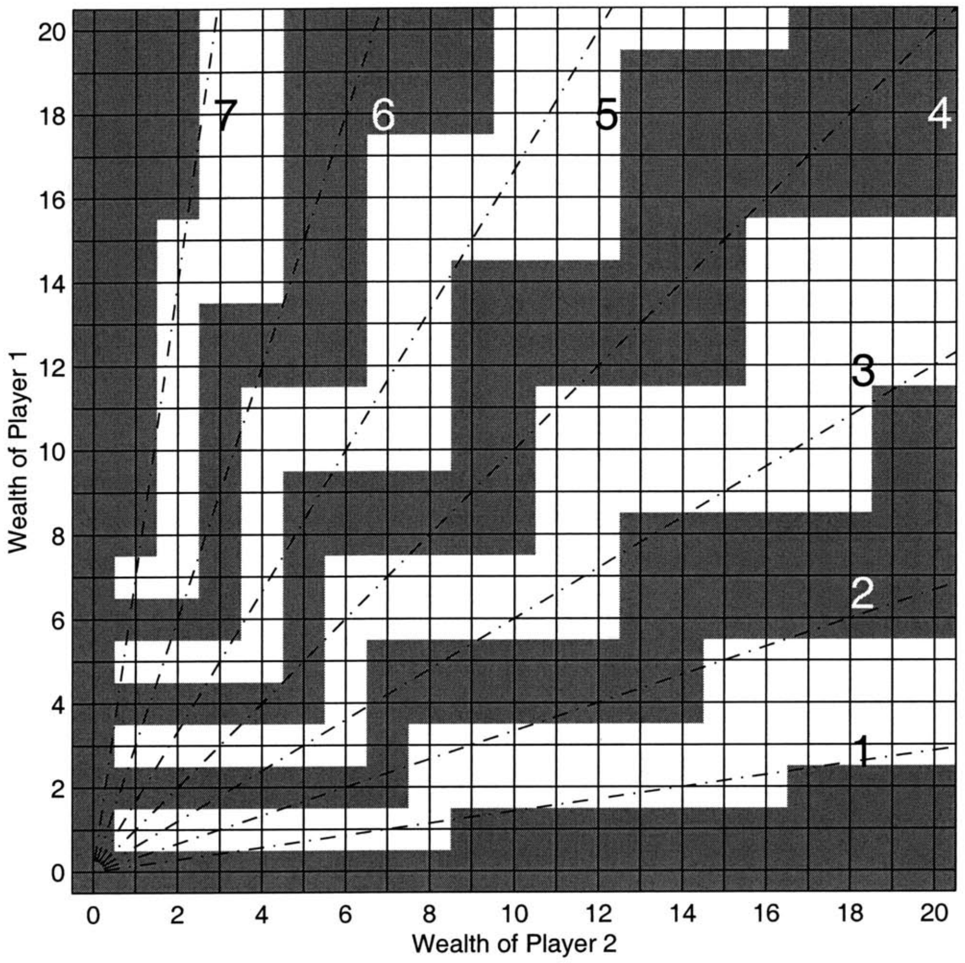


Figure 7-6: Optimal number of wins for user 1 for initial wealth less than 20 and $R = 8$. The numbers on the plot indicate the number of wins by user 1 when the starting initial wealth falls within the surrounding connected region.

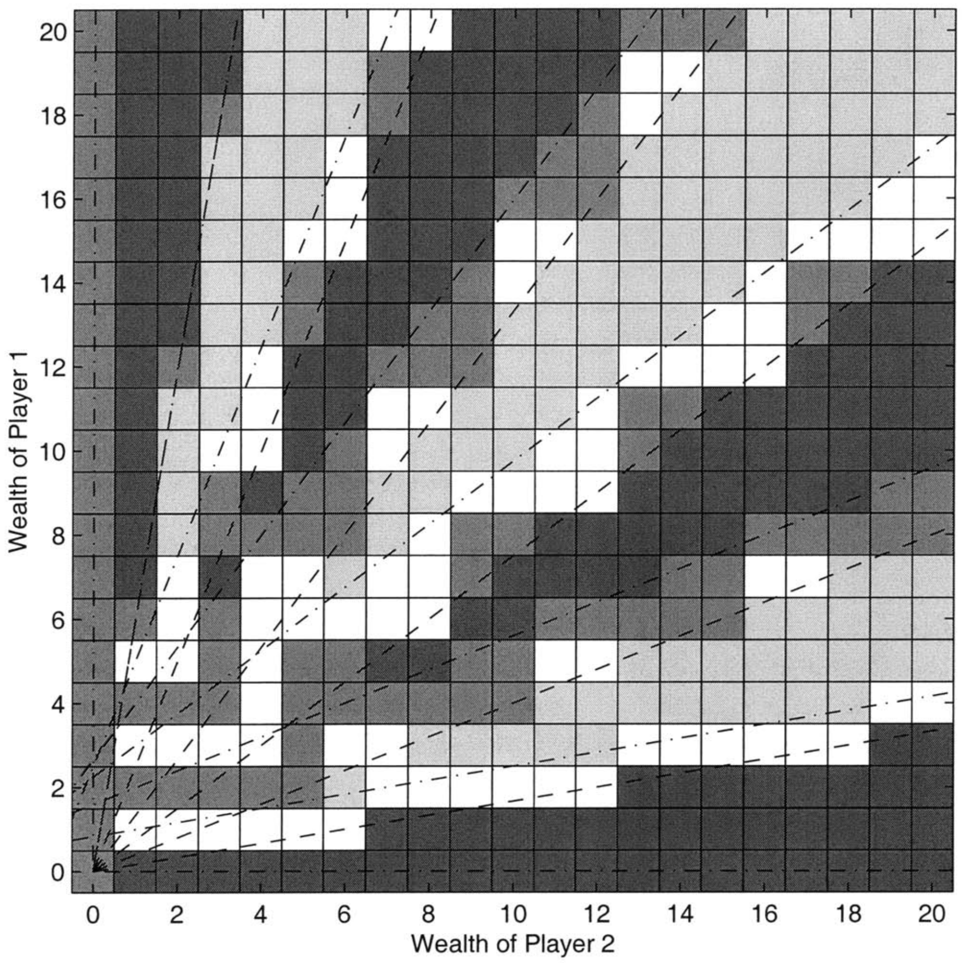


Figure 7-7: This graph indicates the regions in which the users can guarantee a certain number of wins using a second-price auction. Going counterclockwise from the bottom axis, the regions indicate that user 1 can guarantee 0 wins, 1 win, 2 wins, etc. up to 6 wins. The dashed lines indicate linear bounds within which the users can guarantee wins, and the shaded squares indicate which initial wealth distributions fall within the linear bounds.

Fairness for the Second-Price Auction Our main goal in this section is to establish the fairness properties of this repeated game when using a second-price auction mechanism. The ideal measure of fairness that we use as a comparison is that we would like the ratio between the number of wins for each user, R_1/R_2, to be as close to W_1/W_2 as possible. This is similar to the notion of deterministic fairness we presented in Chapter 3. However, with only a finite number of rounds, we are constrained so that $R_1+R_2 = R$, and can not represent any possible W_1/W_2. In this section, we show in Theorem 7.4 that when the first user uses the strategy specified in Theorem 7.3, that user guarantees a number of wins at least as big as $\lfloor RW_1/(W_1+W_2)\rfloor$ when the wealth of each user is large enough. If both users use the strategy, the first user guarantees this or $\lceil RW_1/(W_1+W_2)\rceil$ and the second user guarantees the rest. Subsequently we show the same result for first-price auctions (Theorem 7.6).

It turns out that there is a very simple rule for the optimal bidding strategy for each user that will guarantee the maximum number of wins for each. First, define the following quantities:

$$\phi_i(k) = \left\lfloor \frac{W_i}{k} \right\rfloor \qquad \phi_i^\epsilon(k) = \left\lfloor \frac{W_i-\epsilon}{k} \right\rfloor = \begin{cases} \phi_i(k), & W_i \neq mk \\ \phi_i(k)-1, & W_i = mk \end{cases} \tag{7.3}$$

for positive integer m and $\epsilon \in (0,1)$. Then, the following theorem holds.

Theorem 7.3 *If $\phi_2^\epsilon(k) \geq \phi_1(\ell+1)$ and $\phi_2^\epsilon(k+1) < \phi_1(\ell)$ for some $k, \ell > 0$ and $k+\ell = N$, then user 1 has a strategy by which he can guarantee at least ℓ wins and user 2 has a strategy by which she can guarantee at least k wins.*
If $\phi_2^\epsilon(1) < \phi_1(R)$ ($k=0$, $\ell=R$) then user 1 can win every round.
If $\phi_1(1) \leq \phi_2^\epsilon(R)$ ($k=R$, $\ell=0$) then user 2 can win every round.
Proof: First, we show that user 1 can guarantee ℓ wins. Assume that W_1, W_2, ℓ, and k all satisfy the two conditions in the theorem, and user 1 bids $b_1 = b_1(i) = \phi_1(\ell)$ in each auction as long as he is able. Also, assume (for later contradiction) that user 2 can win $k+1$ times. Then we know that the minimum amount of money that user 2 must spend to win $k+1$ times is

$$B_2 = (k+1)b_1 \ . \tag{7.4}$$

It is clear that as long as user 1 follows this strategy, he will have enough money to bid $\phi_1(\ell)$ until after he has won at least ℓ times, because $\phi_1(\ell)\ell \leq W_1$. We also know that if $\phi_1(\ell) > \phi_2^\epsilon(k+1)$, then $\phi_1(\ell) > \frac{W_2-\epsilon}{k+1}$. So, the following reasoning shows that in order to win $k+1$ times, user 2 must bid more than her wealth to win the $(k+1)$-st time, which is not allowed, and we have a contradiction. We have

$$B_2 = (k+1)b_1 = (k+1)\phi_1(\ell) > (k+1)\frac{W_2-\epsilon}{k+1} = W_2 - \epsilon$$

which implies that $B_2 \geq W_2$. But, this means that after k wins, user 2 must have already paid $B_2 - \phi_1(\ell)$, and so would only have $W_2 - (B_2 - \phi(\ell)) \leq \phi(\ell)$ wealth remaining. In order to win for the $(k+1)$-st time, user 2 must bid more than $\phi(\ell)$, which is not allowed given her remaining wealth. Thus, user 1 can guarantee ℓ wins.

Next, we show that user 2 can guarantee k wins. Again, assume that W_1, W_2, ℓ, and k all satisfy the two conditions in the theorem. Now, user 2 bids $b_2 = b_2(i) = \phi_2^\epsilon(k)+1$ in the ith auction as long as she is able. User 2 can bid this high because she will only pay at most $b_2(i)-1$ for any win (because user 1 wins all ties) and $k(b_2-1) = k\phi_2^\epsilon(k) \leq W_2$. We assume again (for later contradiction) that user 1 can win $\ell+1$ times. Then, the minimum amount of money user 1 must spend is

$$B_1 = (\ell+1)b_2 = (\ell+1)(\phi_2^\epsilon(k)+1) > (\ell+1)\frac{W_1}{\ell+1} = W_1 \ .$$

So, $B_1 > W_1$ and we have a contradiction.[4]

For the last two cases, note that if $\phi_2^\epsilon(1) < \phi_1(N)$ then user 1 can bid $\phi_1(R)$ every time and win every round. Similarly if $\phi_1(1) \leq \phi_2^\epsilon(R)$ and user 2 bids $\phi_2^\epsilon(R) + 1$ every round. ■

It is important to note that the conditions on $\phi_1(\cdot)$ and $\phi_2^\epsilon(\cdot)$ cover all possible pairs (W_1, W_2), so this theorem tells us how many wins an user will have as a function of the starting wealth of the two users. One important point to remember when thinking about fairness in the finite rounds model is that there are only a finite number of possible allocations that can be achieved. As was mentioned earlier with the simulation observations, the only possible ratios between R_1^* and R_2^* are $m/(R-m)$, where $m = 0, 1, \ldots, N$. This means that for certain values of $\mathbf{W}$, the ratio W_1/W_2 can not be achieved. However, the next corollary and theorem establish an approximate notion of fairness.

Corollary 7.1 *If W_1 is large enough,*

$$\frac{k+1}{\ell} - \frac{k}{W_1} > \frac{W_2}{W_1} > \frac{k}{\ell+1} \tag{7.5}$$

then user 1 can guarantee ℓ wins and user 2 can guarantee k wins.

Proof: From the definitions above,

$$\phi_1(\ell) > \frac{W_1}{\ell} - 1 \qquad \text{and} \qquad \frac{W_2 - \epsilon}{k+1} \geq \phi_2^\epsilon(k+1)\ .$$

So, if W_1 is large enough so that $W_1/\ell - 1 > (W_2 - \epsilon)/(k+1)$ then we know that

$$\begin{aligned} \frac{W_1}{\ell} - 1 &> \frac{W_2 - \epsilon}{k+1} \\ \frac{k+1}{\ell} - \frac{k+1}{W_1} &> \frac{W_2 - \epsilon}{W_1} \\ \frac{k+1}{\ell} - \frac{k}{W_1} &> \frac{W_2}{W_1}\ . \end{aligned}$$

The last step follows because $\epsilon < 1$. This final inequality implies that $\phi_1(\ell) > \phi_2^\epsilon(k+1)$.

We can show the second inequality by seeing that

$$\frac{W_2}{k} > \frac{W_1}{\ell+1}$$

implies that $\phi_2^\epsilon(k) \geq \phi_1(\ell+1)$ This follows by noting that if W_2/k is not an integer, then ϵ and the implication is obvious. If W_2/k is an integer, then $W_1/(\ell+1)$ must be less than an integer, and so will drop to the next lowest integer when floored, and $\phi_1(\ell+1)$ will still be less than or equal to $\phi_2^\epsilon(k)$. ■

It tells us that for ratios of W_2 to W_1 near a ratio k/ℓ where $k + \ell = R$, user 2 wins k auctions and user 1 wins ℓ auctions. It is important to note that this corollary only defines a nonempty set of W_2/W_1 if $W_1 + W_2$ is greater than a value that grows (asymptotically) as $R^2/4$.[5]

[4] As a clarification of a not-so-obvious detail, the ϵ in the definition of $\phi_i^\epsilon(k)$ is needed because otherwise if $W_i = mk$ for some integer m, user 2 could not bid $\phi_2(k) + 1$ after it has won $k-1$ auctions. It would only have $\phi_2(k)$ left to bid.

[5] Is there a tighter bound that gives a (sub-)linear bound? Not in the case of a first-price auction. In that case, these bounds are within 1 of the best linear bound (simply add or subtract $1/W_1$ on either side to widen the bounds), as can be verified by the plots. Asymptotically, as N increases, the minimum total wealth needed for these linear bounds to prove useful grows as $k\ell$, whose maximum value is $R^2/4$.

Theorem 7.4 (Second-Price Fairness) *In a second-price repeated auction with two users there exists some integer L_R such that for all W_1, W_2 with $W_1, W_2 \geq L_R$, if*

$$\frac{W_1}{W_2} \in \left[\frac{n}{R-n}, \frac{n+1}{R-(n+1)}\right) ,$$

then user 1 *can guarantee either $\ell = n$ or $\ell = n+1$ wins, and user* 2 *can guarantee $k = R - \ell$ wins. Also, $n = \lfloor W_1 R/(W_1 + W_2) \rfloor$.*

Proof: We will find a value L_R such that if $W_1 > L_R$, then

$$\frac{n+2}{R-(n+1)} - \frac{n+2}{W_1} > \frac{n+1}{R-(n+1)} > \frac{W_1}{W_2} \geq \frac{n}{R-n} > \frac{n}{R-n+1} .$$

The first inequality tells us (with Corollary 7.1 and Theorem 7.3) that user 1 can guarantee no more than $n+1$ wins, and the fourth inequality tells us that user 1 can guarantee at least n wins. This follows because the number of wins for either user increases with its wealth, and thus also increases (for user 1) with W_1/W_2.

The second and third inequalities establish the assumption in the statement of this theorem. We also know that if user 1 can guarantee n wins, then user 2 can guarantee $R - n$ wins (similarly for $n + 1$).

The last thing is to find a value for L_R. To do this, we simply solve the first inequality above for W_1:

$$\begin{aligned} \frac{n+2}{R-(n+1)} - \frac{n+2}{W_1} &> \frac{n+1}{R-(n+1)} \\ W_1(n+2) - (n+2)(R-(n+1)) &> W_1(n+1) \\ W_1 &> (n+2)(R-(n+1)) . \end{aligned}$$

And, from the second inequality, we know that

$$\begin{aligned} \frac{n+1}{R-(n+1)} &> \frac{W_1}{W_2} \\ W_2 &> \frac{W_1(R-(n+1))}{n+1} \\ W_2 &> \left(\frac{n+2}{n+1}\right) \cdot (R-(n+1))^2 . \end{aligned}$$

So, if we set L_R equal to the maximum of these two bounds over all n, the theorem follows. For this case, L_R turns out to be $2(R-1)^2$. ∎

Fairness for the First-Price Auction If we follow similar reasoning, but use first-price auctions instead of second-price auctions at each step, we can prove the following theorems.

Theorem 7.5 *If $\phi_2(k) > \phi_1(\ell + 1)$ and $\phi_2(k+1) \leq \phi_1(\ell)$ for some $k, \ell > 0$ and $k + \ell = N$, then user 1 has a strategy by which he can guarantee at least ℓ wins and user 2 has a strategy by which she can guarantee at least k wins.*
If $\phi_2(1) \leq \phi_1(R)$ ($k = 0$, $\ell = R$) then user 1 can win every round.
If $\phi_1(1) < \phi_2(R)$ ($k = R$, $\ell = 0$) then user 2 can win every round.

Proof: In order to prove this, use the bidding strategies where user 1 bids $\phi_1(\ell)$ all the time and user 2 bids $\phi_2(k)$ every round. Then we have

$$\begin{array}{l} B_1 = (\ell+1)b_2 = (\ell+1)\phi_2(k) > (\ell+1)\frac{W_1}{\ell+1} = W_1 \\ B_2 = (k+1)(b_1+1) = (k+1)(\phi_1(\ell)+1) > (k+1)\frac{W_2}{k+1} = W_2 \ , \end{array} \tag{7.6}$$

and again we have a contradiction. User 1 and user 2 must win at least ℓ and k rounds, respectively.

For the final two cases, note that if $\phi_2(1) \leq \phi_1(R)$ then user 1 can bid $\phi_1(R)$ every time and win every round. Similarly if $\phi_1(1) < \phi_2(R)$ and user 2 bids $\phi_2(R)$ every time. ∎

Again, the conditions on $\phi_1(\cdot)$ and $\phi_2(\cdot)$ cover all possible starting wealths. A corollary similar to the second-price corollary follows.

Corollary 7.2 *If W_1 is large enough,*

$$\frac{k+1}{\ell} \geq \frac{W_2}{W_1} \geq \frac{k}{\ell+1} + \frac{k}{W_1} \tag{7.7}$$

then user 1 can guarantee ℓ wins and user 2 can guarantee k wins.
Proof: The proof follows similar reasoning as the proof of Corollary 7.1. ∎

Finally, we can prove a similar fairness theorem for the first-price auction.

Theorem 7.6 (First-Price Fairness) *In a first-price repeated auction with two users there exists some integer L_R such that for all W_1, W_2 with $W_1 \geq L_R$, if*

$$\frac{W_1}{W_2} \in \left[\frac{n}{R-n}, \frac{n+1}{R-(n+1)}\right) ,$$

then user 1 can guarantee either $\ell = n$ or $\ell = n+1$ wins, and user 2 can guarantee $k = R - \ell$ wins. Also, $n = \lfloor W_1 R/(W_1 + W_2) \rfloor$.
Proof: We will find a value L_R such that if $W_1 > L_R$, then

$$\frac{n+2}{R-(n+1)} \geq \frac{n+1}{R-(n+1)} > \frac{W_1}{W_2} \geq \frac{n}{R-n} \geq \frac{n}{R-n+1} + \frac{n}{W_1} .$$

The first inequality tells us (with Corollary 7.2 and Theorem 7.5) that user 1 can guarantee no more than $n+1$ wins, and the fourth inequality tells us that user 1 can guarantee at least n wins. This follows because the number of wins for either user increases with its wealth, and thus also increases (for user 1) with W_1/W_2.

The second and third inequalities establish the assumption in the statement of this theorem. We also know that if user 1 can guarantee n wins, then user 2 can guarantee $N - n$ wins (similarly for $n+1$).

The last thing is to find a value for L_R. To do this, we simply solve the last inequality above for W_1:

$$\begin{array}{rcl} \frac{n+1}{R-(n+1)} & > & \frac{n}{R-(n+1)} - \frac{n}{W_1} \\ W_1(n+1) & > & W_1 n - n(R-(n+1)) \\ W_1 & > & n(R-(n+1)) \ . \end{array}$$

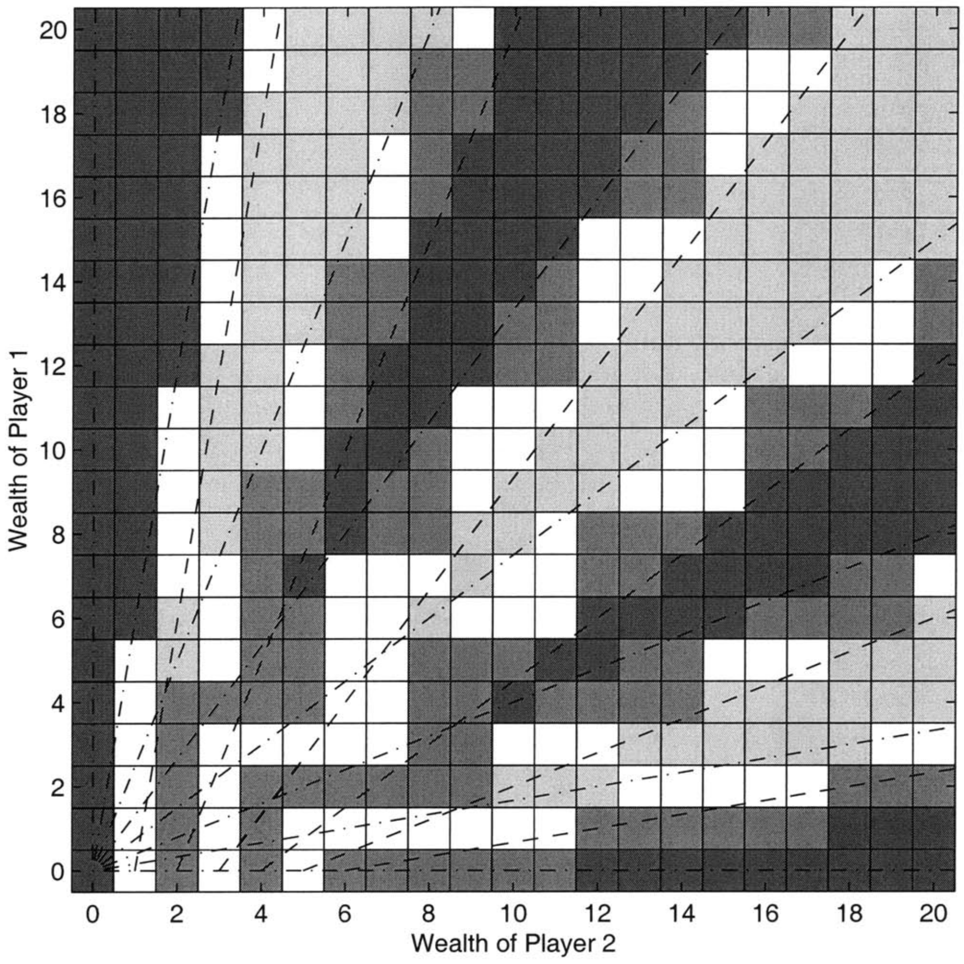


Figure 7-8: This graph indicates the regions in which the users can guarantee a certain number of wins using a first-price auction. Going counterclockwise from the bottom axis, the regions indicate that user 1 can guarantee 0 wins, 1 win, 2 wins, etc. up to 6 wins. The dashed lines indicate linear bounds within which the users can guarantee wins, and the shaded squares indicate which initial wealth distributions fall within the linear bounds.

And, from the second inequality, we know that

$$\begin{aligned}\frac{n+1}{R-(n+1)} &> \frac{W_1}{W_2}\\ W_2 &> \frac{W_1(R-(n+1))}{n+1}\\ W_2 &> \left(\frac{n}{n+1}\right)\cdot(R-(n+1))^2\,.\end{aligned}$$

So, if we set L_R equal to the maximum of these two bounds over all n, the theorem follows. For this case, L_R turns out to be less than $(R-1)^2/2$. ∎

Many Users

Now, we take the results from the two user case, and extend them to apply to the case in which we have many users ($N > 2$), each of which is trying to win as many rounds as possible.

As before, we would like to find an optimal strategy that guarantees a minimum fraction of wins for an user that is close to the ratio between that user's wealth and the wealth of the other users. In this section as we previously showed for two users, we show that we can guarantee the closest achievable ratio that is less than the wealth ratio when the wealth of the users is large enough. When all users play this strategy, this leaves a number of wins (up to one per user) that are not accounted for in the guarantees.

As before, each of the M users begins with some initial wealth and bids during each round in order to win as many rounds as possible. For now, we will concentrate on the second-price auction.[6] Using the results from the two user case, we can show that the many user case is also approximately fair. To do this, we start with the following definition:

Definition 7.4 (Coalition) *A* coalition *is a subset of the users who are allowed to share the resources of the entire subset. Specifically, the bids from the users in a coalition can be as large as the entire wealth held by the coalition.*

In order to show the fairness of the many user case, we consider a modification of the many user case, where the users are broken up into two coalitions. The first coalition consists of a single user (called N, without loss of generality) and the second coalition consists of the rest of the users. The only case that this does not cover is when the single user has the highest priority in tie-breaking. The previous results for two players do not change qualitatively if we reverse the tie-break order, so we do not have to explicitly worry about it here. The first observation we can make about this modified game is that the performance of the second coalition is bounded below by the collective performance of the member users in the original many user game — the coalition can restrict itself to play simply as a collection of independent users if that gives the best results. Next, we will show that the first coalition (single user) with wealth W_N can achieve the same performance competing against the second coalition as in the two user case, with users whose wealths are W_N and $\sum_{i=1}^{N-1} W_i$. Before we present the lemma, we would like to make the observation that the number of wins that a user can guarantee is a monotonic function of the wealth that the user has. The simple way to see this is that a user with more money can always pretend he has less if it would help him, thereby having the ability to win as many rounds as a user with less money.

[6]We can use similar reasoning to show the same results for the first-price auction.

Lemma 7.5 (Coalition Bidding Strategies) *Assume that a sequence of bids for the second coalition of size $N-1$ is $b[i]$, $i = 1, \dots, R$, where each $b[i] = (b_1[i], \dots, b_{N-1}[i])$. Call the number of wins achieved by the coalition k using this sequence of bids. Then, the sequence of bids $b'[i] = (\max_j b_j[i], 0, \dots, 0)$ achieves k' wins such that $k' \geq k$. Thus the optimal sequence of bids $b^*[i]$ is of the same form as $b'[i]$, and guarantees $k^* \geq k' \geq k$ wins.*

Also, the first coalition (single user) has a bidding strategy that guarantees $R - k^*$ wins against any opposing coalition strategy.

Proof: First, define $b_j[i]$ as the largest element of $b[i]$, and $b_k[i]$ as the second largest element of $b[i]$. There are three cases to consider:

- $b_j[i] \geq b_k[i] \geq b_N[i]$,
- $b_j[i] \geq b_N[i] \geq b_k[i]$,
- $b_N[i] \geq b_j[i] \geq b_k[i]$.

In the first case, the second coalition wins, but pays a higher price ($b_k[i]$) than it needed to. If we set $b_k[i] = 0$, as in b', then the second coalition will pay less for the win at round i, making it possible for the coalition to win more rounds under b' than b. In the second case, the second coalition wins and pays the same price under b' and b, so there is no difference between the two strategies. In the third case, the second coalition loses, and the winner pays the same price under b' and b, so there is no difference between the two strategies. Thus, b' guarantees at least as many wins as b.

To prove that the singleton can guarantee $R - k^*$ wins against any bidding strategy, we start by noting that it can guarantee that many wins against any strategy in which only a single member of the opposing coalition bids at each round by analogy with the two user case. Now we have to show that it can only do better if the opposing coalition uses any other strategy. We do this by recalling that the only time that any other strategy differs from a strategy in which a single member of the coalition bids is when $b_j[i] \geq b_k[i] \geq b_N[i]$, or when the second coalition has the highest two bids. The only effect this will have is to reduce the second coalition's wealth in the next round. But, we know that the number of wins by the singleton can only increase as the opposing coalition's wealth decreases (or as the ratio $W_N / \sum_{i \neq N} W_i$ increases), so the singleton will win at least as many times as it would have if the second coalition behaved as a single user. ■

So, we can assume that the optimal bidding strategy for a coalition is simply for a single member of the coalition to bid. This means that the modified many user game can be reduced to a two user game, between the first user with wealth W_N and a super-user with wealth equal to $W_S = \sum_{i=1}^{N-1} W_i$.

We now approach the problem slightly differently than before, and provide lower bounds for the number of wins each singleton can guarantee using the linear bounds given in the previous corollaries. These are looser than the previous guarantees for the two user case. In order to show these fairness bounds for the many user game, we use Theorems 7.4 and 7.6. We now define $W_{-i} = \sum_{k \neq i} W_k$ as the sum of the starting wealths for all users other than user i.

Theorem 7.7 (Many User Fairness) *In a many user auction game (first- or second-price) with arbitrary tie-break ordering, if the starting wealth of the users is greater than the maximum value of L_R, and*

$$\frac{W_i}{W_{-i}} \in \left[\frac{n_i}{R - n_i}, \frac{n_i + 1}{R - (n_i + 1)}\right) ,$$

then user i can guarantee

$$n_i = \left\lfloor \frac{W_i R}{\sum_{j=1}^{N} W_j} \right\rfloor$$

wins.

Proof: From Theorems 7.4 and 7.6, we know that user i can guarantee at least n_i wins, where

$$\frac{n_i}{R-n_i} \le \frac{W_i}{W_{-i}} < \frac{n_i+1}{R-(n_i+1)}$$

for some value of L_R. By simply rearranging this, we get

$$\begin{aligned} n_i &\le \frac{W_i}{W_{-i}}(R-n_i) && (7.8)\\ n_i\left(1+\frac{W_i}{W_{-i}}\right) &\le \frac{W_i R}{W_{-i}} && (7.9)\\ n_i &\le \frac{W_i R}{\sum_{j=1}^{N} W_j}\ . && (7.10) \end{aligned}$$

and

$$\begin{aligned} n_i+1 &> \frac{W_i}{W_{-i}}(R-(n_i+1)) && (7.11)\\ (n_i+1)\left(1+\frac{W_i}{W_{-i}}\right) &> \frac{W_i R}{W_{-i}} && (7.12)\\ n_i &> \frac{W_i R}{\sum_{j=1}^{N} W_j} - 1\ . && (7.13) \end{aligned}$$

From this, we know that

$$n_i = \left\lfloor \frac{W_i R}{\sum_{j=1}^{N} W_j} \right\rfloor\ .$$

■

This theorem establishes that, in a many user auction game, each user can guarantee a number of wins that is the closest integer less than or equal to the fraction of wins corresponding to that user's fraction of the overall wealth. The sum of these guarantees is lower bounded by

$$\sum_{i=1}^{N} n_i \ge R - N\ . \qquad (7.14)$$

These guarantees can be met if each user uses the optimal bidding strategies for the two user auction. In fact, if we know that $W_i = Lw_i$, where $\sum_{j=1}^{N} w_j = R$ for some integer L then each user can guarantee exactly w_i wins and no more.[7]

Example 7.4 (Many User Auction) *Consider two examples of the many user auction:*

(i) There are four competing users, with starting wealths of 6, 12, 12, 18, *and they compete for* 8 *rounds. The users can guarantee* 1, 2, 2, 3 *wins, respectively.*

(ii) There are four competing users, with starting wealths of 6, 12, 13, 17, *and they compete for* 8 *rounds. The users can guarantee* 1, 2, 2, 2 *wins, respectively. The final unguaranteed round might be won by any of the users.*

[7]This fact was also noted in [31] for the case when $w_i = 1$ for all i.

7.5.2 Fairness

For both the two player and the many player scenarios above, each player could guarantee a number of wins that is within 1 of the desired number of wins for a given maximum number of rounds R. Although we have not shown an upper bound on the number of wins beyond the loose bound in Equation (7.14), it is safe to say that these auction mechanisms guarantee some level of fairness.

It is tempting to say that the fairness guarantee gets tighter as the number of rounds increases. If the number of wins is always within 1 of the desired value then as the number increases the fraction of wins converges to the desired fraction. However, we have to take into account the value L_R in the statement of Theorems 7.4, 7.6 and 7.7. In all of the theorems, L_R grows as R^2. This means that the amount of money, or at least the divisibility of the money, has to increase more quickly than the number of rounds to guarantee the simple bounds given in these theorems.

In spite of this, the mechanisms still provide a degree of approximate fairness, as can be seen in the theorems and figures given so far. We can not make the claim that these mechanisms are deterministically fair as we could for the deterministic models considered earlier, but they do guarantee that the number of wins, or total allocation is always very close to the desired number or allocation.

7.5.3 Infinite Rounds

Now we change the game so that we have an infinite number of rounds, and each user starts with no wealth, but receives a constant amount ρ_i of income at the beginning of each round. The wealth evolution equation is now

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \rho - \mathbf{p}[n] .$$

We would like to show that if both users follow the optimal bidding strategy, then the average number of wins per round (for user 1) converges to the ratio $\rho_1/(\rho_1 + \rho_2)$. As before, each user would like to maximize the guaranteed wins that he gets. Additionally, to make the problem cleaner, we break ties probabilistically. With probability 1/2, the first user wins a tie, otherwise the second user wins. We define $\underline{\mathbf{R}}(n, w_1, w_2)$ as the number of wins that user 1 can guarantee in the first m rounds with starting wealth configuration (w_1, w_2).

$$\underline{\mathbf{R}}(m+1, w_1, w_2) = \max_{b_1} \min_{b_2} \mathrm{E}\left\{\sigma_m + \underline{\mathbf{R}}(m, w_1 - b_2\sigma_m, w_2 - b_1(1-\sigma_m))\right\} . \tag{7.15}$$

Again, we define σ_m as the number of wins (0 or 1) by the first user at round m, but now σ_m is a random variable if $b_1(m) = b_2(m)$. By iterating this, starting at $m = 1$, we would hope that

$$\lim_{m\to\infty} \frac{\underline{\mathbf{R}}(m, w_1, w_2)}{m} = \nu^* ,$$

where ν^* is the optimal fraction of wins that user 1 can guarantee. [When is this guaranteed to converge?] Figure 7-9 shows $\underline{\mathbf{R}}(\cdot)$ after different rounds m with $(\rho_1, \rho_2) = (5, 3)$ with a limit on the maximum wealth either user can have of 20. It is apparent for this case that the limit is converging to a constant ν^* such that

$$\nu^* = \frac{\rho_1}{\rho_1 + \rho_2} .$$

We show this convergence in Figure 7-10. At round 50, the maximum value of $\underline{\mathbf{R}}(\cdot)$ corresponds with user 1 winning 33 times, and the minimum value corresponds with user 1 winning 29 times. The average value corresponds with user 1 winning just over 31 times, the expected number given the income rates for both users.

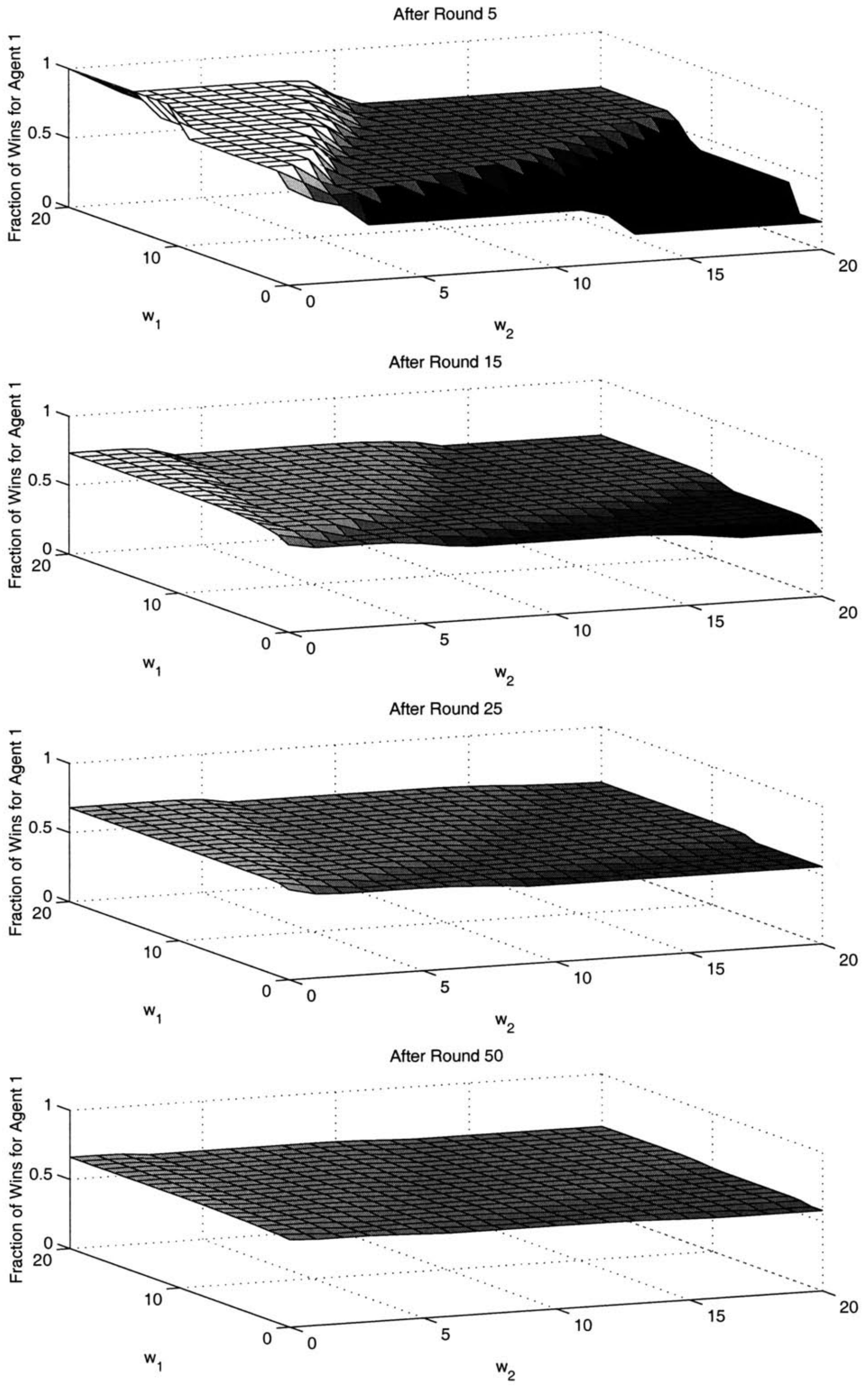


Figure 7-9: Plots of $\underline{\mathbf{R}}(\cdot)$ after 5, 15, 25, 50 rounds. Values $(\rho_1, \rho_2) = (5, 3)$ were used.

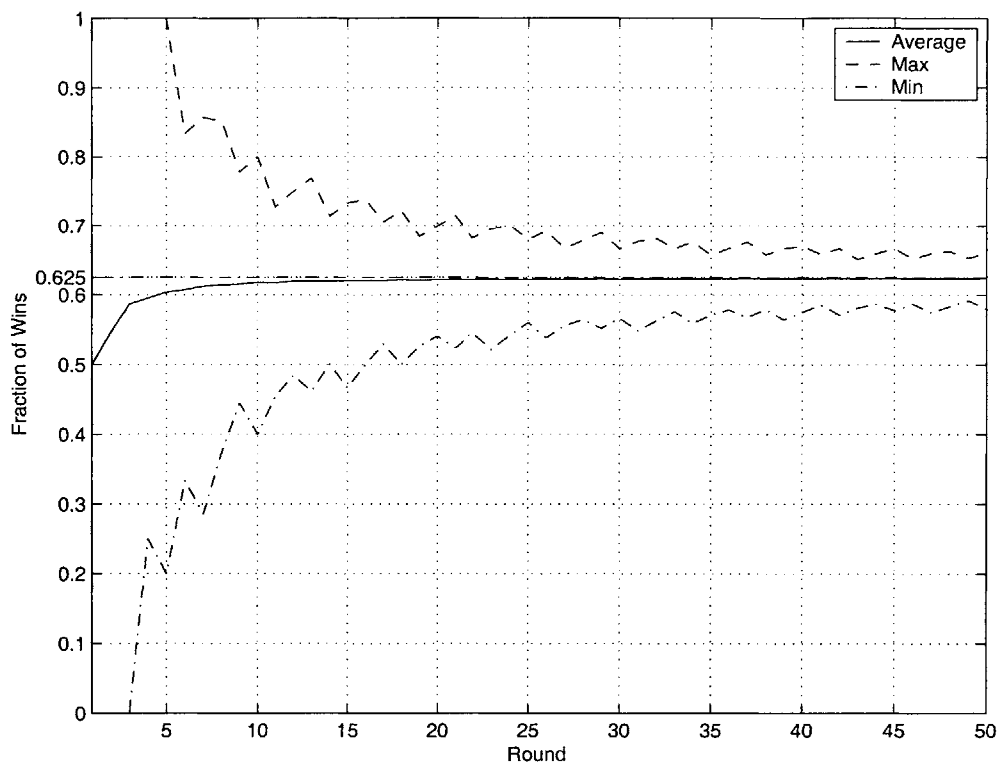


Figure 7-10: Convergence of $\underline{\mathbf{R}}(m, w_1, w_2)$ to a constant $\nu^* = \rho_1/(\rho_1 + \rho_2)$. The average of $\underline{\mathbf{R}}(m, w_1, w_2)$ over all values of w_1 and w_2 is shown along with the maximum and minimum for a number of values of m.

Optimal Strategies for First Price Auction

We now show that the same bidding strategy that guaranteed the maximum number of wins for the finite round model also guarantees the maximum fraction of wins for the infinite round model. First we will show this for the first price auction. Recall that in the first price auction, the optimal bidding strategy was for the first user to bid $\phi_1(\ell)$ at every round, if $\phi_2(k) > \phi_1(\ell+1)$ and $\phi_2(k+1) \leq \phi_1(\ell)$ for $k + \ell = R$. If we (just for motivation) extend this to large values of R while increasing the initial wealth at the same time, we get the following two relations:

$$\left\lfloor \frac{\rho_2 R}{k} \right\rfloor > \left\lfloor \frac{\rho_1 R}{\ell+1} \right\rfloor , \qquad \left\lfloor \frac{\rho_2 R}{k+1} \right\rfloor \leq \left\lfloor \frac{\rho_1 R}{\ell} \right\rfloor$$

which can be rewritten as

$$\left\lfloor \frac{\rho_2}{f_2} \right\rfloor > \left\lfloor \frac{\rho_1}{f_1 + 1/R} \right\rfloor , \qquad \left\lfloor \frac{\rho_2}{f_2 + 1/R} \right\rfloor \leq \left\lfloor \frac{\rho_1}{f_1} \right\rfloor .$$

We define $f_2 = k/R$ and $f_1 = \ell/R$. As R increases, the gap in the inequalities becomes smaller and smaller. In the limit as R approaches ∞, the only possible values for f_1 and f_2 are ones such that

$$\left\lfloor \frac{\rho_2}{f_2} \right\rfloor = \left\lfloor \frac{\rho_1}{f_1} \right\rfloor .$$

In order for the inequalities to hold as we increase R, the argument on each side of the equation must be equal to an integer, and so we have

$$\frac{\rho_2}{f_2} = \frac{\rho_1}{f_1} .$$

If we take this along with the restriction that $f_1 + f_2 = 1$, then we have the solution

$$f_i = \frac{\rho_i}{\rho_1 + \rho_2}$$

for $i = 1, 2$. So, the optimal bidding strategy, as R increases, tends towards bidding $\min(w_i, \rho_1 + \rho_2)$ at every round.

However, this argument does not establish that this bidding strategy will work for the infinite round model; it only provides motivation for considering the strategy. In the infinite round model, we do not start with a lump sum, we receive an income equal to ρ_i after every round. So, the amount of money available over R rounds is still $\rho_i R$ as in the finite rounds case, but it is not available at the beginning. The following theorem establishes that this difference does not have a large effect on the behavior of the model.

To prove that the strategy of bidding $\rho_1 + \rho_2$ at every round can guarantee winning a fraction f_1 of the time for user 1, we need to show that if user 1 follows this strategy he is guaranteed winning at least a fraction f_1 of the time against any opposing strategy. Then, we know that user 2 can guarantee f_2, which implies that user 1 can not guarantee any more than f_1.

Theorem 7.8 (First-Price Infinite Rounds Bidding) *If user 1 bids $\min(w_1, \rho_1 + \rho_2)$ at every round in an infinite rounds repeated auction then he will win at least a fraction $f_1 = \rho_1/(\rho_1 + \rho_2)$ of the rounds.*

Proof: First, establish that user 1 can actually bid this high while winning a fraction f_1 of the rounds. If he pays $\rho_1 + \rho_2$ at every round he wins, then he will have enough income (over a long period of time) to win a fraction f_1 of the rounds:

$$(f_1 R)(\rho_1 + \rho_2) = \rho_1 R \ .$$

Next, assume user 2 is to win a fraction $\epsilon > 0$ of the rounds that user 1 should win (leaving user 1 with $f_1 - \epsilon$). Every round that user 2 takes from user 1 allows user 1 to bid more than $\rho_1 + \rho_2$ on some other round that he would not have been able to bid that high on otherwise (because he does not need to pay for the loss). This means that user 1 can cause user 2 to pay more than $\rho_1 + \rho_2$ for some fraction ϵ of the rounds she would have paid less for before. But, if user 2 still wins them, user 1 can force user 2 to pay more for some other fraction of the rounds. Essentially, user 1 can force user 2 to pay more than $\rho_1 + \rho_2$ for all rounds that she wins if she tries to win more than a fraction f_2, but user 2 does not have enough income to sustain that rate of payment, so we have a contradiction. User 1 must win at least a fraction f_1 of the rounds.

The only final observation that needs to be made is that even though the winning user may not pay $\rho_1 + \rho_2$ for every win if they do not bid that high, if user 2 were to win a larger fraction of the rounds than f_2 the wealth of user 1 would grow without bound, and cause the winning bid to always equal $\rho_1 + \rho_2$ as long as user 2 wins more than a fraction f_2 of the rounds. ■

The same reasoning applies to user 2, so we can finally say that if both users use the same bidding strategy, each will win a fraction f_i of the time.

Corollary 7.3 (Fairness of Infinite Rounds Model) *If user i bids* $\min(w_i, \rho_1 + \rho_2)$ *at each round, then user i will win a fraction $f_i = \rho_i/(\rho_1 + \rho_2)$ of the rounds.*

Optimal Strategies for Second Price Auction

The second price auction is more complex. The proof for Theorem 7.8 also holds if we use a second-price auction with a few minor modifications. This shows that the second price auction also guarantees fairness when using the same strategies as above, but the behavior of the wealth accumulation is different. For the first-price auction, the wealth of the two users remains bounded (empirically) as the rounds progress. With a second-price auction, experiments show that a user may be able to accumulate an unbounded amount of wealth even though the mechanism is allocating the resource fairly. However, the wealth can only accumulate at a sub-linear rate. In all experiments we have tried, there is no way for a user to modify her bidding strategy in order to utilize this accumulated wealth to affect the fraction of wins that she gets. The wealth accumulates at a rate that is not large enough to change the fraction of wins. This seems to indicate that while the second price auction guarantees deterministic fairness in the infinite horizon case, it is not *bounded fair.*

7.6 Discussion

As we mentioned at the beginning of this chapter, and as we have shown throughout, auctions are useful tools because we can use them to make fairness guarantees even when the users are allowed to act as selfish agents, each trying to maximize his own allocation. This type of allocation mechanism is becoming more important as computing and communication systems become more complex, decentralized and uncontrolled.

We have shown (for a few possible cases) that when the actions of the users are assumed to be unknown and possibly adversarial, an individual user can still guarantee himself a minimum fraction of the available resource very close to the user's desired fraction. The original model in Chapter 3 did not make this same guarantee. The only constraints on the requests made in the original model were ones that were agreed upon in the statement of the problem. In a real system, however, there is nothing stopping a user from requesting a very large amount of the resource. Even if the user only gets a portion of what he asks for, he is still better off than if he were to ask for less. In fact, if the user asks for more than he desires, the user will receive a larger fraction of the resource. In situations when a user may not necessarily receive the fraction he desires because the resource is oversubscribed, it is beneficial to the user to ask for more than he desires in order to compensate. By using an auction, we decouple the user's desires from the requests (bids) he is allowed to make. By doing this, we allow the user to request whatever he would like, but we force him to do so using a bounded amount of bidding resources. In fact, this is a similar method to that which is used, for example, in credit-based algorithms in [30] and in the PGPS algorithm. To bound the behavior of the users, a limited amount of credit is allotted to each user which then determines how the allocation decisions are made.

The use of auction mechanisms connects our original mathematical model with a more realistic appreciation of the users' possible behavior in an actual system. Our original model assumed that in some way the actions of the users are unknown ahead of time but somehow well-behaved; any practical implementation of a fair allocation mechanism like the ones we have looked at here needs to consider the complete range of user actions, including worst-case adversarial scenarios.

Auctions have been heavily studied over the years, and so have been chosen as convenient tools for resource allocation in many areas. However, it is important that in addition to using empirical data to guarantee allocation fairness as was done in [67], we also determine what theoretical guarantees can be made. The results presented here are a simple beginning, an illustration that we can analyze these and possibly more complex mechanisms mathematically. The auctions we have looked at are special cases of more general scheduling rules that might be used in our allocation framework.

Chapter 8

Conclusion

We have presented an investigation of a number of aspects of a simple dynamic resource allocation problem. Our approach to this problem treats the entire allocation mechanism as a dynamic feedback system for which the behavior can be understood by analyzing the trajectories of the state variable, or the lag of the system. There are a few important observations to make after our presentation of the analysis. First, the model as we have formulated it is simple and even naïve in some respects. It operates only in discrete-time, the scheduling rules are constrained to correspond with minimum distance quantizing rules, the feedback dynamics are about as simple as possible, and very little application specific information is incorporated into the model (although we do give a few examples that illustrate how this can be done). Despite this, mechanisms of this simplicity have been used as the basis for proposed practical scheduling algorithms in the literature. There are a number of possible extensions to our basic model that would make the model both more complex and more powerful. We discuss a number of these below.

The second observation is that the view of a dynamic resource allocation problem as a geometric problem seems to be underutilized. It is true that many resource allocation problems do not fit within a nice geometric framework because of dependencies, discontinuous control and system descriptions, and other reasons. However, as we have shown here, there are classes of problems that can be usefully modeled this way. A large class of discrete-time queueing network models can be embedded in a geometric framework. By viewing a number of different allocation problems in a common framework it is easier to illustrate the similarities and differences between them.

These two observations make clear the pedagogical aspects of this thesis that have not been discussed. The model we present, although it is not rich enough to capture all of the behavior of interest to us, is complex enough to aid in the understanding of similar models. Our geometric interpretation of resource allocation problems can help to understand the relationship between different models, such as different classes of queueing systems.

As a concrete example, consider the different classes of queueing systems we considered earlier. All of them can be modeled in our framework, and the differences come out in the choice of constellation. Typical unconstrained queueing networks with N queues, each with a dedicated server, have a constellation that looks like the corners of a N-dimensional cube and a matrix $B[n]$ that corresponds to the interconnections of the servers. Reentrant multi-class networks place restrictions on the points in the constellation; they now form a subset of the corners of a cube. The matrix $B[n]$ also has a particular structure that simply routes jobs deterministically between buffers. The input-queued switch can be seen as a simple queueing network with a collection of N^2 queues and servers with constraints on which servers can be activated simultaneously, and no complex structure in the routing matrix $B[n]$. The input-queued switch is also a special case of a constrained queueing network, which can be modeled as a network of queues with constraints on which servers can be activated simultaneously.

With this view of different classes of queueing networks, we can begin to see how they relate, and why particular scheduling rules may work for some and not for others. For instance, in a queueing network of independent simple queues a work-conserving FIFO scheduling rule will always maintain stability while in a reentrant network that is not necessarily the case [13]. For the non-reentrant network, the FIFO rule is equivalent to using a quantizing scheduler based on the 2-norm, while for the reentrant network it does not because of the constraints placed on the servers. It is possible to define schedule rules for reentrant networks that are consistent with the 2-norm, and thus stabilize (at least) networks that satisfy the modeling assumptions in this thesis. The dichotomy between the maximum weighted matching, which guarantees stability for all feasible input rates, and other maximal matchings, which do not necessarily make the same guarantee, is also echoed in our presentation. It appears here as the difference between a smooth norm and a non-smooth norm.

Before we conclude, it would probably be appropriate to answer the question posed for the traffic light scheduling problem given in the introduction. As stated, the traffic light scheduling problem is identical to the 2×2 input-queued switching problem described in detail in Section 3.5.1. Thus, any scheduling rule that stabilizes an input-queued switch will also stabilize the traffic light model. For example, if we simply schedule the pair of directions with the largest total number of waiting cars (maximum weighted matching), the number of cars in line will never grow unbounded as long as the arrival rate remains feasible.

We end with a discussion of a number of related topics and extensions that would be interesting for further research.

Model Structure and Stability

Variable-Length Packets

Until now, we have only considered discrete-time queueing models with fixed-length jobs. In many queueing applications, it is much more natural to model the processing times (or allocations) as varying in length. As an illustration of how this might be done for our model and still allow us to model it meaningfully and perhaps extend our stability results, we consider the case when the processing times (or job/packet lengths) and inter-arrival times are determined according to corresponding probability distributions. We will look at a single server with N competing queues. With a single server, the scheduling times occur when a job departs from the server, or when a job arrives when all queues are empty. Call these times $\{\tau_k\}$ for $k = 0, \ldots, P$, where τ_0 is the time at which the first job arrives (the beginning of the busy period) and τ_P is the time at which the second-to-last packet departs within the busy period. If x_k is the processing time for the k-th packet processed by the server then $\tau_k - \tau_{k-1} = x_k$ and we can write the dynamics that govern the amount of work in the queue $\ell(t)$ as

$$\dot{\ell}(t) = \mathbf{r}(t) - \mathbf{s}(t) \ .$$

The arrival function $\mathbf{r}(t)$ is a point process; we define it as the derivative of the function $\mathbf{R}(t)$, where $\mathbf{R}(t)$ is equal to the amount of work (sum of the job lengths) that has arrived up until time t. The scheduling signal $\mathbf{s}(t)$ is a vector that indicates which of the queues is being serviced. We will define $\mathbf{S}(t)$ as the running integral of $\mathbf{s}(t)$. The scheduling decision made at time τ_k is $\mathbf{s}[k] = \mathbf{s}(\tau_k)$, and the amount of work that arrives after τ_{k-1} up until τ_k is $\mathbf{r}[k]$. Because we only care about the queue size at scheduling times, we call $\ell(\tau_k) = \ell[k]$ and we can rewrite the dynamics as

$$\begin{aligned}\ell(\tau_{k+1}) &= \ell(\tau_k) + \int_{\tau_k}^{\tau_{k+1}} (\mathbf{r}(t) - \mathbf{s}(t))dt \\ \ell[k+1] &= \ell[k] + \mathbf{r}[k+1] - x_{k+1}\mathbf{s}[k]\ .\end{aligned}$$

This looks much like our dynamic model. It differs because x_{k+1} may be correlated with $\ell[k]$ (it is the size of the job at the front of the queue that gets scheduled). If we do not allow the scheduling rule to make use of the value x_{k+1}, only the statistics of x_{k+1} and the current amount of work in the queues, this model does fit within our earlier framework with x_n taking the place of $B[n]$. However, we might also want to look at scheduling rules that make use of the value of x_n, or at least use the fact that $\mathbf{r}[n+1]$ is correlated with x_n.

Non-idling Scheduling Rules

Much of the existing work on stability of queueing networks under various scheduling disciplines, including [52, 37, 5], assumes the property that the scheduling disciplines are non-idling — whenever a server has jobs waiting in a buffer, it must be busy. This property is not necessary to prove stability, but generally makes it simpler. In most queueing network models, it is possible to define a non-idling policy because all of the servers operate independently. In [65], however, a constrained queueing model is developed that allows the status of a server to depend on the status of others. In this way, two servers may both have waiting jobs, but may be constrained such that both servers can not be busy simultaneously. It would be impossible to define a non-idling scheduling policy for this example.

In our general dynamic allocation model also, a non-idling policy can not always be defined. Moreover, the concept of a non-idling policy does not always make sense when the constellation is a collection of points in a continuous space. The stability results here identify classes of scheduling rules that always stabilize a model when it is stabilizable. These rules turn out to be non-idling policies when the constellation allows them, and so can be thought of (at least for the limited class of systems we consider here) as generalizations of certain non-idling policies.

As we showed in Chapter 4, if we make our scheduling decision by minimizing the squared distance between the schedule and the current vector of queue lengths, we can guarantee full throughput. It is also easy to see that when the queues are not constrained, this minimum squared distance rule corresponds to a non-idling policy; each queue operates independently, so the scheduling choice is $\mathbf{s} = 1$ (busy) or $\mathbf{s} = 0$ (idle). The squared distance is the squared difference between the queue length and the schedule. When the length is nonzero, choosing $\mathbf{s} = 1$ will result in the smallest squared difference. When the length is zero, $\mathbf{s} = 0$ is the best choice.

The concept of a generalized class of non-idling policies might prove to be useful not only for systems like constrained queueing networks, but perhaps other non-queueing systems. This concept has not been explored in much depth, either here or in the work on constrained queueing networks.

Schedulers

Another restriction we make in this thesis that allows us to reach our conclusions more easily is that the schedulers be based on quantizing scheduling rules. Other rules such as the p-fair algorithm [3] can be seen to have better performance than a quantizing scheduling rule with respect to minimizing the maximum lag in simple cases.

One area that deserves some investigation is the relative performance and possible optimality of different scheduling rules. Other work such as [52] has looked at this for related models, and would have much to connect with. The scheduling rule we chose for this thesis is simple, and does not make systematic use of all of the information we may have about the current and past values of the lag.

More general scheduling rules may be very difficult to deal with, and would not fit easily into the geometric framework we have here. However, we might allow for time-varying scheduling rules and constellations. We have not captured any time dependence in the constellations we discuss. An extension in this direction would give the framework much greater modeling power. This would include models which use two alternating constellations that may each be very simple to implement. Neither constellation may guarantee stability for all desired inputs, but by alternating we may be able to stabilize the system. As a simple example, consider allocating a single resource between two users. If we can only choose to give the resource to the first user in odd rounds and to the second user in even rounds, our stability region gets smaller, but it might be simpler to implement than a combined allocation mechanism.

It is also pretty clear that the proofs we give here for stability can withstand some variation from a norm-based scheduling rule. We discussed some extensions such as skewed-norm scheduling rules earlier in Chapter 4. For deterministic stability it looks as if we only need that the surface of the sub-level sets of the distance function become increasingly smooth as they get larger. The actual shape of the sets may change with increasing function value. In fact, even non-convexity is not always a problem as long as the surface becomes arbitrarily flat locally.

Stability of Stochastic Mechanisms

We have only given stochastic stability results for a subset of our stochastic model with bounded parameters and a single scheduling rule consistent with the squared 2-norm. It seems that these results can be extended as in the deterministic case to mechanisms for which the error metric is an arbitrary smooth norm, and also extended to include a subset of mechanisms with unbounded parameters $\mathbf{r}[n]$ and $B[n]$ whose distributions satisfy certain properties.

Although this class of models is perhaps practical, because all real systems have bounded parameters, there is no reason to believe that we can not extend the class to include a much wider range of parameter characterizations. Support for this comes from results such as those given in [35] and [53]. By combining arguments from these two sources, we can tentatively come to the conclusion that a stochastic model for which $\mathbf{r}[n]$ and $B[n]$ are white processes and the region of support of the distribution of $\mathbf{r}[n+1]$ overlaps that of $B[n+1]\mathbf{s}[n]$ has a lag vector that converges to a fixed random variable with finite mean. This argument might even go further as we have seen for the case considered in this thesis.

It would also be interesting to know whether or not quantizing rules consistent with smooth norms stabilize certain classes of stochastic models. This would allow us to say with certainty that we can guarantee stability while at the same time allowing a choice of norm in order to control the behavior of the lag beyond simple fairness.

Stability of Corner Norms

We did not discuss corner norm stability in much detail in Chapter 4, but it appears that there is interesting structure to problems using a corner norm in a quantizing scheduler. There are only certain corridors of the state space in which the lag might extend to infinity. These corridors are parts of unbounded decision regions for internal constellation points. If we can understand the behavior of the lag within these corridors and the surrounding regions, we can characterize the stability of the mechanism when the increment distribution is bounded.

For instance, in two dimensions we can break a constellation into collections of two neighboring surface points and an internal point. If the internal point has an unbounded decision region that extends between the regions for the two neighboring points, we might have an unstable trajectory if the average rate of arrival can generate trajectories that travel within the corridor of the internal point. For a stochastic model, the trajectories travel at an average speed out, then when they exit the corridor to the side, return at some other average speed, flip-flopping between the two neighboring regions until they enter the corridor again. If we can show that these trajectories always return within an expected finite length of time or even with probability 1, we will have some sense of the stability of a particular corner norm.

This problem is easy to conceive in any number of dimensions and looks perhaps tractable in 2-dimensions, but may quickly get very complex in higher dimensions as the corridors develop geometric structure of their own. It seems that an investigation of this problem could make connections with a number of areas in random walk theory.

Applications of Model

Our treatment of the model has been somewhat abstract; the search for more practical situations that are captured well by this model would be valuable. For example, a generalized vector Σ-Δ encoder has not been specifically proposed in the literature to our knowledge. Perhaps it is the large number of more complex application-specific coding methods that has obscured the simple Σ-Δ encoder. One possible application of the vector Σ-Δ encoder might be as a method of encoding multiple signals simultaneously where a scalar Σ-Δ encoder might have been used before for each signal. For instance, mechanisms similar to Σ-Δ encoders are used to dither images to fewer colors; a vector Σ-Δ encoder might be useful in some way for these cases when multiple images are involved.

Investigation of this encoder along with other unrelated systems such as the storage model mentioned earlier might bring out some possible applications of the results we develop in this thesis.

Variable-Sized Channel Allocation Problem

For the channel allocation problem in Chapters 5 and 6, there are a few interesting related topics that might prove interesting.

We have not investigated nonuniform constellations in much detail. The problem of choosing schedules is very closely related to well-known NP-hard problems even for small values of M; approximation algorithms that are practical in those situations may be applicable here. There is also the question of what is the best performance possible over all nonuniform constellations. Even a fairly tight upper bound on the average or maximum error would be a useful tool to help us understand how well we can do, and how well we are doing with a particular choice of constellation.

One specific related question that would be both practical and interesting is whether or not it is possible to use a polynomial-time approximation algorithm with a nonuniform constellation and thereby outperforms the best uniform constellation. It is not even obvious that this is possible. For instance, does the VSS algorithm given here ever achieve a smaller maximum or average error when used with a nonuniform constellation than with the uniform constellation for which it is optimal?

There is also the possibility that a similar approach might be used for constellations with other geometries. Our specific results only apply when the constellation lies within a specified region. In 3-dimensions, the constellation lies within a 2-dimensional triangle. Another interpretation of our results is as a decomposition of the triangle into a number of smaller ones whose sizes correspond to the piece sizes. It is clear that a planar constellation with a hexagonal boundary can also be decomposed in a similar way into a set of smaller hexagons. it is not obvious whether there is a simple algorithm that can be used to compute the sequence of smaller sizes as for the triangular case, and it is also not clear what gains this would give us, or how generalizable the approach is to other geometries.

Auction Mechanism

The auctions we have looked at are attractive because they are able to achieve fair allocations of a resource with a very simply-described scheduling rule. In many other cases we may need more complex mechanisms to deal with other possibilities. Allocating many items in parallel as in [31] is often required; just as that work might be incorporated with what we have presented here, there are other extensions such as to time-varying resource availability, and other domain-specific constraints on how the resource is to be allocated over time that would be required in other circumstances. Just as with our original framework, it is important to be able to apply these ideas in more complex situations. For example, consider a collection of computing resources that are available to be used by external users if there is idle time. The allocation mechanism needs to balance the fairness concerns of the external users while also dealing with the availability and amount of resources. There might be constraints on which users may use which resources at what times as well.

Another interesting twist on the basic idea is not to allow the users to know what other users are bidding. For privacy, security, or other concerns, it might be necessary to blind each user to the actions of the others. In this case, only partial information about the state of the resource will be available to the users in order to make their decisions. The users may know only whether they have won or lost each particular round, or they might know the value of the winning bid. Depending on the available information the users will use different bidding strategies.

This also brings up the issue of implementation. We have not suggested a specific implementation of any our mechanisms, but the ultimate usefulness of these allocation techniques is only proven when they are incorporated into a real system. Systems such as Spawn and others described in [68] and [10] are the final test of the practicality of particular economically-based approaches to resource allocation. Our results establish that the specific auction implementation can affect the fairness properties of an allocation mechanism. It would be interesting to see if a modification of the Spawn system to use our mechanism with probabilistic income would exhibit significantly better fairness properties.

Auctions are only one example of extended dynamics. Although the study of arbitrary feedback dynamics may be too complex, it might be interesting to investigate the connections between some of the more complex Σ-Δ encoders and our resource allocation problem. Is there a benefit to using a higher-order feedback loop as is done to shape noise in a scalar Σ-Δ encoder? How do the attendant stability problems for the scalar encoder relate to stability in the vector input model?

Bibliography

[1] S. Anily and A. Federgruen. Structured partitioning problems. *Operations Research*, 39(1):130–149, 1991.

[2] S. Anily and A. Federgruen. 2-echelon distribution-systems with vehicle-routing costs and central inventories. *Operations Research*, 41(1):37–47, 1993.

[3] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, pages 600–625, 1999.

[4] D. Bertsimas, D. Gamarnik, and J. N. Tsitsiklis. Stability conditions for multiclass fluid queueing networks. *IEEE Transactions on Automatic Control*, 41(11):1618–1631, 1996.

[5] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[6] L. D. Burns, R. W. Hall, D. E. Blumenfeld, and C. F. Daganzo. Distribution strategies the minimize transportation and inventory costs. *Operations Research*, 33(3):469–490, 1985.

[7] J. Candy and G. Temes, editors. *Oversampling Delta-Sigma Data Converters.* IEEE Press, New York, New York, 1991.

[8] Chang and Gill. Algorithmic solution of the change making problem. *Journal of the Association for Computing Machinery*, 17(1):113–122, 1970.

[9] K. L. Chung. *Markov Chains With Stationary Transition Probabilities.* Springer-Verlag, New York, New York, 1967.

[10] S. H. Clearwater, editor. *Market-Based Control: A Paradigm For Distributed Resource Allocation.* World Scientific, 1996.

[11] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices and Groups.* Springer-Verlag, New York, New York, 1999.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[13] J. Dai. Stability of fluid and stochastic processing networks. Course Notes at the Centre for Mathematical Physics and Stochastics (`http://www.maphysto.dk/`), 1998.

[14] J. G. Dai. On positive harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *Annals of Applied Probability*, 5:49–77, 1995.

[15] J. G. Dai and B. Prabhakar. The throughput of data switches with and without speedup. *Preprint*, 1999.

[16] G. Demange, D. Gale, and M. Sotomayor. Multi-item auctions. *The Journal of Political Economy*, 94(4):863–872, August 1986.

[17] M. V. Eyuboglu and J. G. D. Forney. Lattice and trellis quantization with lattice and trellis bounded codebooks – high rate theory for memoryless sources. *IEEE Transactions on Information Theory*, 39(1):46–59, 1993.

[18] R. G. Gallager. *Discrete Stochastic Processes.* Kluwer Academic Publishers, Boston, MA, 1996.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, CA, 1979.

[20] D. D. Gemmill. Approximate solutions for the cutting stock 'portfolio problem'. *European Journal of Operational Research*, 44:167–174, 1990.

[21] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[22] R. M. Gray. Quantization noise spectra. *IEEE Transactions on Information Theory*, 36(6):1220–1244, 1990.

[23] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998.

[24] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes.* Oxford University Press, Oxford, 1992.

[25] B. Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14:502–525, 1982.

[26] S. Hein and A. Zakhor. On the stability of sigma delta modulators. *IEEE Transactions on Signal Processing*, 41(7):2322–2348, 1993.

[27] S. Hein and A. Zakhor. *Sigma Delta Modulators: Nonlinear Decoding Algorithms and Stability Analysis.* Kluwer Academic Publishers, Boston, 1993.

[28] J. Holtz. Pulsewidth modulation for electronic power conversion. *Proceedings of the IEEE*, 82(8):1194–1214, 1994.

[29] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, Cambridge, MA, 1988.

[30] A. C. Kam and K.-Y. Siu. Linear-complexity algorithms for QoS support in input-queued switches with no speedup. *IEEE Journal on Selected Areas in Communications*, 17(6):1040–1056, 1999.

[31] M.-Y. Kao, J. Qi, and L. Tan. Optimal bidding algorithms against cheating in multiple-object auctions. *SIAM Journal of Computing*, 28(3):955–969, 1999.

[32] M. J. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space division switch. *IEEE Transactions on Communications*, 35(12):1347–1356, 1987.

[33] P. J. Kelly and M. L. Weiss. *Geometry and Convexity*. John Wiley and Sons, 1979.

[34] J. C. Kieffer. Stochastic stability for feedback quantization schemes. *IEEE Transactions on Informatino Theory*, 28:248–254, 1982.

[35] T. Koski. Statistics of the binary quantizer error in single-loop sigma-delta modulation with white gaussian input. *IEEE Transactions on Information Theory*, 41(4):931–943, 1995.

[36] P. Krishna, N. S. Patel, A. Charny, and R. J. Simcoe. On the speedup required for work-conserving crossbar switches. *IEEE Journal on Selected Areas in Communications*, 1999.

[37] P. R. Kumar. A tutorial on some new methods for performance evaluation of queueing networks. *IEEE Journal on Selected Areas in Communications*, 13(6):970–980, 1995.

[38] P. R. Kumar and S. P. Meyn. Stability of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 40(2):251–260, 1995.

[39] P. R. Kumar and S. P. Meyn. Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 41(1):4–17, 1996.

[40] H. Kushner. *Introduction to Stochastic Control*. Holt Rinehart Winston, New York, New York, 1971.

[41] H. J. Kushner. *Stochastic Stability and Control*. Academic Press, New York, New York, 1967.

[42] S. H. Lu and P. R. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, 36(12):1406–1416, 1991.

[43] G. Luckjiff and I. Dobson. Power spectrum of a sigma-delta modulator with hexagonal vector quantization and constant input. In *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, volume 5, pages 270–273, 1993.

[44] G. Luckjiff, I. Dobson, and D. Divan. Interpolative sigma delta modulators for high frequency power electronic applications. In *26th Annual IEEE Power Electronics Specialists Conference*, pages 444–449, 1995.

[45] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1997.

[46] V. A. Malyshev and M. V. Menshikov. Ergodicity, continuity and analyticity of countable markov chains. *Transactions Moscow Mathematical Society*, 39:1–48, 1981.

[47] S. Martello and P. Toth. Optimal and canonical solutions of the change making problem. *European Journal of Operational Research*, 4:322–329, 1980.

[48] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley & Sons, New York, 1990.

[49] L. Massoulié and J. Roberts. Bandwidth sharing: Objectives and algorithms. On the web at Kelly's site, 1999.

[50] R. P. McAfee and J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[51] N. McKeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267, 1999.

[52] S. P. Meyn. Feedback regulation for sequencing and routing in multiclass queueing networks. Submitted to ISIT 2000 and SIAM Journal of Control and Optimization, June 2000.

[53] S. P. Meyn and R. L. Tweedie, editors. *Markov Chains and Stochastic Stability.* Springer-Verlag, London, 1993.

[54] P. R. Milgrom and R. J. Weber. A theory of auctions and competitive bidding. *Econometrica*, 50(5):1089–1122, September 1982.

[55] J. Nieznański. Performance characterization of vector sigma-delta modulation. In *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 531–536, 1998.

[56] M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* The MIT Press, Cambridge, MA, 1994.

[57] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[58] A. W. Roberts and D. E. Varberg. *Convex Functions.* Academic Press, 1973.

[59] H. Sariowan, R. L. Cruz, and G. C. Polyzos. SCED: A generalized scheduling policy for guaranteeing quality-of-service. *IEEE/ACM Transactions on Networking*, 7(5):669–684, 1999.

[60] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann, San Francisco, 1996.

[61] B. D. Schutter and B. D. Moor. The extended linear complementarity problem and the modeling and analysis of hybrid systems. In P. A. et al., editor, *Hybrid Systems V*, pages 70–85. Springer-Verlag, 1999.

[62] D. Stiliadis and A. Varma. Providing bandwidth guarantees in an input-buffered crossbar switch. In *Proceedings of IEEE INFOCOM 1995*, pages 960–968, 1995.

[63] D. Stiliadis and A. Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, 1998.

[64] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proceedings of IEEE INFOCOM 1998*, pages 533–539, 1998.

[65] L. Tassiulas and P. P. Bhattacharya. Allocation of interdependent resources for maximal throughput. *Stochastic Models*, 1999.

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

[67] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, February 1992.

[68] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proceedings of the Eighteenth International Conference on Distributed Computing Systems (ICDCS-98)*, pages 612–621, 1998.

[69] J. W. Wright. The change-making problem. *Journal of the Association for Computing Machinery*, 22(1):125–128, 1975.

[70] L. Zhang. VirtualClock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems*, 9(2):101–124, 1991.

Appendix A

Properties of Convex Sets

In this appendix, we present a number of results related to convex sets that we use in the subsequent proofs and earlier analysis in this thesis. We state the results without proof; proofs and more detail is available from other sources such as [58, 33, 5]. The definitions of a convex set and the convex hull of a set were given in Section 3.2.

The main results we need relate to tangent planes (also called supporting hyperplanes) and normals of convex sets. The *surface* C^s of a convex set C is the set of points for which no neighborhood is contained within C. A (hyper-)plane Z is *tangent* to a convex set C at a point $c \in C$ if $c \in C^s$ and $C \cap Z \neq \emptyset$ and C is contained in one of the two closed half-spaces defined by Z. We denote a tangent plane at c by $Z(c)$. The normal to a tangent plane $Z(c)$ is $z(c)$.

This definition means that a convex set is always completely on one side of a tangent plane. Because of this, we can always define $z(c)$ to be a unit length vector such that it points inside the set C. Given this definition of the normal $z(c)$, we know that c always minimizes $d^{\mathrm{T}}z(d)$ over all $d \in C$ [58, 5].

Theorem A.1 *Given a convex set C and a tangent plane $Z(c)$, $d^{\mathrm{T}}z(c)$ is minimized when $d = c$.*

We also would like to know whether or not tangent planes exist. The following theorem states that every point on the surface of a convex set has a tangent plane [33].

Theorem A.2 *Every point $c \in C^s$ for a convex set C has at least one tangent plane $Z(c)$.*

From these two theorems, we know that every point c on the surface of a convex set maximizes $d^{\mathrm{T}}z$ for some z.

Corollary A.1 *For every point $c \in C^s$ for a convex set C there exists a z such that $d^{\mathrm{T}}z$ is minimized when $c = d$. Moreover, we can choose $z = z(c)$ for any tangent plane $Z(c)$.*

We can connect the definition of the normal $z(c)$ to norm sub-level sets with the following.

Theorem A.3 *If the sub-level set $\{c \mid f(c) \leq D\}$ for a differentiable function $f(\cdot)$ is a convex set C for all D, then if ∇f exists at $c \in C^s$,*

$$\frac{\nabla f(c)}{\| \nabla f(c) \|_2} = -z(c) \ .$$

Proof: The gradient is always perpendicular to the tangent plane of a differentiable (smooth) surface at some point $c \in C$. From [33], this tangent plane must be the unique supporting hyperplane of the set C at c. Thus, the gradient and $z(c)$ are parallel, and we have chosen $z(c)$ to point inwards, so the equation above holds. ∎

For $f(\cdot) = \|\cdot\|$, we know that the normal to a tangent plane at a point c on the surface of a norm sub-level set is equal to the gradient of the sub-level set at that point if the gradient exists.

Another fact we use, related to tangency, is the definition of a tangent line. A line L is tangent to a convex set C at a point c if $c \in C^s$, $L \cap C \subseteq C^s$ and if $c \in L$. A line tangent at point c is denoted by $L(c)$.

Theorem A.4 *If a line L is not a tangent line for some $c \in C$ for a convex set C, then it either intersects the surface of C at 2 points or at 0 points.*

Finally, we have the well-known result that a tangent plane to a convex set C always includes an extreme point of the set [5]. In particular, if C is a polytope there are finitely many extreme points and so we know that at least one of these points is always on every tangent plane $Z(c)$.

Theorem A.5 *The intersection of a tangent plane $Z(c)$ for some point c with the surface of a convex polytope always includes an extreme point of the polytope.*

Appendix B

Proofs

B.1 Proof of Theorem 5.1

Theorem B.1 (Maximum Error with Equally-Sized Pieces) *If* $\mathbf{p} = \mathbf{p}_0$, $M \geq N$ *and* $\mathbf{r} \in \mathbb{S}_N$, *then*

$$E(\mathbf{p}_0) = \frac{N-1}{N} \cdot \frac{1}{M} \ .$$

Proof: To show this, we determine a subset of $\mathcal{S}$ that is guaranteed to contain the closest $\mathbf{s} \in \mathcal{S}$ to $\mathbf{r}$. In order to do this, it is easier to deal with the vector $\mathbf{R} = M\mathbf{r}$, and the integer vector $\mathbf{S} = M\mathbf{s}$. The set of possible scaled schedules $\mathbf{S}$ is $\mathcal{S}_M$. If we are able to find the closest $\mathbf{S}$ to $\mathbf{R}$, then the corresponding $\mathbf{s}$ is closest to $\mathbf{r}$. It is easy to see that the only elements of $\mathcal{S}_M$ that we need to consider are the ones with all elements within 1 of the corresponding element of $\mathbf{R}$. We can show this by observing that there exists an $\mathbf{S} \in \mathcal{S}_M$ such that S_i is either equal to $\lfloor R_i \rfloor$ or $\lceil R_i \rceil$. Define $K = M - \sum_i \lfloor R_i \rfloor$, and pick any subset I of $(1, \ldots, N)$ of size K such that $R_i \neq \lfloor R_i \rfloor$ for all $i \in I$. We construct an $\mathbf{S}$ by setting

$$S_i = \begin{cases} \lfloor R_i \rfloor & \text{if } i \notin I \\ \lceil R_i \rceil & \text{if } i \in I \end{cases}$$

By this construction, $\sum_i S_i = M$, and $\| \mathbf{S} - \mathbf{R} \|_\infty \leq 1$. There are $\binom{N}{K}$ possible elements of $\mathcal{S}_M$ that can be constructed as above.

Define $\mathcal{E}(K)$ as the set of N-dimensional 0-1 vectors with K ones and $N - K$ zeros. This argument implies that we only need to worry about finding a vector $\mathbf{D} = \mathbf{R} - \lfloor \mathbf{R} \rfloor$ such that $\sum_i D_i = K$ and that maximizes the distance to the closest $\mathbf{E} \in \mathcal{E}(K)$. We can think of the elements of $\mathcal{E}(K)$ as the set of vectors $\mathbf{S} - \lfloor \mathbf{R} \rfloor$ for each $\mathbf{S}$ constructed above. Because we know $\mathbf{S}^*$ is one of these constructed vectors, we know that $\mathbf{E}^* = \mathbf{S}^* - \lfloor \mathbf{R} \rfloor \in \mathcal{E}(K)$.

Now, we find the vector $\mathbf{D}^*$ that maximizes the error

$$\mathcal{M}(\mathbf{D}) = \min_{\mathbf{E}} \| \mathbf{E} - \mathbf{D} \|_\infty \ .$$

Call $D^{(k)}$ the k-th largest component of $\mathbf{D}$. It is obvious that for any choice of $\mathbf{D}$, the choice of $\mathbf{E}$ that minimizes $\| \mathbf{E} - \mathbf{D} \|_\infty$ is the vector with component $E_i = 1$ if D_i is one of the K largest components of $\mathbf{D}$. So, $D^{(K)}$ is the smallest element such that the corresponding element of $\mathbf{E}$ is 1, and $D^{(K+1)}$ is the largest element such that the corresponding element of $\mathbf{E}$ is 0. It follows from this that the maximum error is either $1 - D^{(K)}$ or $D^{(K+1)}$. If $1 - D^{(K)}$ is the maximum error, we would like to make $D^{(K)}$ as small as possible in $\mathbf{D}^*$. This can be done by making $D^{(k)} = 1 - \delta/(K-1)$ for $k < K$ and small δ, and setting the rest of the elements of $\mathbf{D}$ equal to $(1+\delta)/(N-K+1)$. The error would then be $1 - (1+\delta)/(N-K+1) = (N-K-\delta)/(N-K+1)$. If the maximum error is equal to $D^{(K+1)}$, we would like to make $D^{(K+1)}$ as large as possible. We can do this by setting $D^{(k)} = 0$ for $k > K+1$, and setting the rest of the elements of $\mathbf{D}$ equal to $K/(K+1)$. So, the worst case error is equal to

$$\mathcal{M}(\mathbf{D}^*) = \max\left(\frac{N-K-\delta}{N-K+1}, \frac{K}{K+1}\right) .$$

This is maximized for $K = N-1$.

We know that we can write any $\mathbf{R} = \mathbf{D} + v$, where v is a nonnegative integer vector that sums to $M-K$, and the closest $\mathbf{S}$ can be written as $\mathbf{S} = \mathbf{E} + v$ for some $\mathbf{E}$ because $v = \lfloor \mathbf{R} \rfloor$. Then,

$$\mathcal{M}(\mathbf{R}) = \min_{\mathbf{S}} \| \mathbf{S} - \mathbf{R} \|_\infty = \min_{\mathbf{E}} \| \mathbf{E} - \mathbf{D} \|_\infty .$$

So, there is a $\mathbf{R}^* = \mathbf{D}^* + v$ such that

$$\mathcal{M}(\mathbf{R}^*) = \mathcal{M}(\mathbf{D}^*) .$$

And it follows then, that $\mathbf{r}^* = \mathbf{R}^*/M$ maximizes $e(\mathbf{p}_0, \mathbf{r})$ and so $E(\mathbf{p}_0) = e(\mathbf{p}_0, \mathbf{r}^*)$. If the worst case error happens when $K = N-1$ we have a maximum error of $(N-1)/N$. This corresponds to an error of $(N-1)/NM$ when we scale back to $\mathbf{r}^*$. ∎

B.2 Proofs of Lemmas 5.4, 5.5 and Theorem 5.4

Lemma B.1 *The sequence* $\mathbf{P}^+$ *of length* M *that is produced by the previous algorithm is an upper bound (element by element) on any other integral nondecreasing sequence* $\mathbf{P}$ *that corresponds to a uniformly dense constellation.*

Proof: Proof by contradiction. We show that

$$P_{k+1} \leq \left\lceil \frac{1+\bar{\mathbf{P}}(k)}{N-1} \right\rceil = f(\bar{\mathbf{P}}(k)) \qquad \text{(B.1)}$$

is true for any choice of $k < M-1$ for any sequence $\mathbf{P}$ that corresponds to a uniform constellation. To prove this, assume that $P_{k+1} = f(\bar{\mathbf{P}}(k)) + c$, where c is a positive integer and the sequence $\mathbf{P}$ corresponds to a uniformly dense constellation. Then, we can show that there is an nonnegative integer vector v_{k+1} whose sum is equal to the sum of the elements of $\mathbf{P}$, but that can not be represented by allocating those elements among the N components of v_{k+1}. That vector is $v_{k+1} = (\alpha, x_1, x_2, \ldots, x_{N-1})$, where

$$\alpha = f(\bar{\mathbf{P}}(k)) + \sum_{j=k+2}^{M} P_j + c - 1$$

and the integers x_i are chosen such that

$$\sum_{i=1}^{N-1} x_i = X = \bar{\mathbf{P}}(k) + 1$$

and $x_i \leq f(\bar{\mathbf{P}}(k))$ for all i. First, note that

$$\alpha + X = \sum_{j=1}^{M} P_j = \bar{\mathbf{P}}(M)$$

so we know that v_{k+1} has the correct sum. We now must show that such x_i exist, and then show that the vector v_{k+1} can not be constructed using elements of $\mathbf{P}$ if $P_{k+1} = f(\bar{\mathbf{P}}(k)) + c$. If $X \leq f(\bar{\mathbf{P}}(k))(N-1)$, then we can simply set $x_i = f(\bar{\mathbf{P}}(k))$ for all i until we reach $X = \bar{\mathbf{P}}(k) + 1$. To show this is true, we know that

$$\begin{aligned}
f(\bar{\mathbf{P}}(k))(N-1) &= \left\lceil \frac{1+\bar{\mathbf{P}}(k)}{N-1} \right\rceil (N-1) \\
&= \left(\frac{1+\bar{\mathbf{P}}(k)}{N-1} + \gamma \right) (N-1) \qquad 0 \leq \gamma < 1 \\
&= \bar{\mathbf{P}}(k) + (1 + (N-1)\gamma) \\
&\geq \bar{\mathbf{P}}(k) + 1 = X \ .
\end{aligned}$$

Now we need to show that there is no way of allocating the elements of $\mathbf{P}$ to an N-dimensional vector that equals v_{k+1}. Because we know that the elements with sizes P_j, $j > k$, are all bigger than $f(\bar{\mathbf{P}}(k))$ and thus bigger than all of the x_i, all of them must be part of α. But, α is less than the sum of these sizes, so all of these large elements can not be placed in the first bin, meaning that no allocation of the elements of $\mathbf{P}$ can be equal to v_{k+1}. Thus, there is no allocation of the elements of $\mathbf{p}$ that results in the schedule $\mathbf{s} = v_{k+1}\beta$, contradicting the uniformly dense property of its constellation.

We have shown, finally, that any sequence that corresponds to a uniformly dense constellation can not have an increase between P_k and P_{k+1} for any k that is larger than the value $f(\bar{\mathbf{P}}(k)) - P_k$ determined by Equation (B.1) above. So, because $\mathbf{P}_1^+ = P_1 = 1$, and the increments $P_{k+1} - P_k$ are bounded by $\mathbf{P}_{k+1}^+ - C_k^+$ and are increasing with $\bar{\mathbf{P}}(k)$, we know that $\mathbf{P}_k^+ \geq P_k$ for all k. ∎

Lemma B.2 *The set* $\mathbf{P}$ *corresponds to a uniformly dense constellation if and only if* $P_k \leq P_{k+1} \leq f(\bar{\mathbf{P}}(k))$ *for all* $k = 1, \ldots, M$.

Proof: Proof by induction. We will assume that $\mathbf{P}^{(k)} = (P_1, P_2, \ldots, P_k)$ corresponds to a uniformly dense constellation using k pieces. Then, we will show that if $P_{k+1} = R_{k+1} \leq f(\bar{\mathbf{P}}(k))$, $\mathbf{P}^{(k+1)}$ corresponds to a uniformly dense constellation. Remember that $P_{k+1} \geq P_k$ by definition.

In order to do this, we need to show that every nonnegative integer vector v that has the property that

$$\sum_{i=1}^{N} v_i = \sum_{j=1}^{k+1} P_j = \bar{\mathbf{P}}(k+1)$$

can be constructed by allocating the elements of $\mathbf{P}^{(k+1)}$ to the elements of v (i.e. there is some $(K+1) \times N$ matrix $A \in \mathcal{A}$ such that $v = \mathbf{P}^{(k+1)}A$). First, we show that at least one element of v is at least as large as R_{k+1} for any v we choose. To do this, we assume that $v_i < R_{k+1}$ for all i and show a contradiction. If $v_i < R_{k+1}$ for all i, then

$$\begin{aligned}
\sum_{i=1}^{N} v_i \leq (R_{k+1} - 1)N &= R_{k+1} + (R_{k+1}(N-1) - N) \\
&\leq R_{k+1} + (f(\bar{\mathbf{P}}(k)) \cdot (N-1) - N) \\
&= R_{k+1} + (\bar{\mathbf{P}}(k) + (1 + (N-1)\gamma) - N) \qquad 0 \leq \gamma < 1 \\
&< R_{k+1} + \bar{\mathbf{P}}(k) \\
&= \bar{\mathbf{P}}(k+1)
\end{aligned}$$

which contradicts the definition of v. Thus, at least one element of v is at least R_{k+1}. Call this element v_j.

Now, define $\hat{v}$ such that $\hat{v}_i = v_i$ for all $i \neq j$, and $\hat{v}_j = v_j - R_{k+1} \geq 0$. By the inductive hypothesis, there exists some allocation of $\mathbf{P}^{(k)}$ among the elements of $\hat{v}$ that give us the schedule $\hat{v}$. By simply adding R_{k+1} to $\hat{v}_j$, we get v from $\hat{v}$, so v can be generated by $\mathbf{P}^{(k+1)}$.

Finally, we observe that $\mathbf{P}^{(1)} = P_1 = 1$ obviously generates all integral vectors with a single nonzero element equal to 1, and the proof is complete. If we can generate all possible nonnegative integral vectors v that sum to $\bar{\mathbf{P}}(M)$, then if we scale $\mathbf{P}$ by β to get $\mathbf{p}$ we can generate all schedules $\mathbf{s} = v\beta$, and thus $\mathbf{p}$ has a uniformly dense constellation.

The reverse direction follows from the proof of Lemma 5.5, where it is shown that if the increment is too large between elements of $\mathbf{P}$, the resulting $\mathbf{p}$ can not have a uniformly dense constellation.
■

Theorem B.2 (Optimality of VSSA) *The VSS algorithm computes a schedule* $\mathbf{s}$ *such that* $\| \mathbf{s} - \mathbf{r} \|_\infty$ *is minimized for any* $\mathbf{r}$ *such that* $\sum_i r_i = 1$.

Proof: First, we note that for any vector $\mathbf{v}$ (not necessarily nonnegative) that has the property $\sum_i v_i = K > 0$, any nonnegative vector $\mathbf{s}$ with a uniformly dense constellation that minimizes $\| s - \mathbf{v} \|_\infty$ over all possible nonnegative vectors s with uniformly dense constellations that sum to K also satisfies the property that if $v_i < v_j$ then $s_i < s_j$. This can be shown by assuming that $v_i < v_j$ and $s_i \geq s_j$. The vector $\mathbf{s}'$ created by switching s_i and s_j in $\mathbf{s}$ would be closer to $\mathbf{v}$ than $\mathbf{s}$ is, contradicting the optimality of $\mathbf{s}$.

Second, if $v_i = v_j$, then either $s_i = s_j$ or $|s_i - s_j| = 1$. For the second case, the error is the same whichever of the two is larger. These two facts mean that the largest element in $\mathbf{s}$ corresponds to the largest element in $\mathbf{v}$.

We also know, from the proof of Lemma 5.4, that any length k prefix of $\mathbf{p}$ can be allocated to create any length N schedule in a uniformly dense constellation that sums to $\bar{\mathbf{p}}(k) = K$. In particular, any length N schedule in a uniformly dense constellation that sums to K must have at least one element at least as large as p_k.

Call $\mathbf{v}(m)$ the value of the quantities $\{v_k\}$ at the end of round m in the algorithm. Call $\mathbf{s}(m)$ the value of the computed schedule at the end of round m. Call $\delta(m) = \mathbf{s}^* - \mathbf{s}(m)$, where $\mathbf{s}^*$ is the schedule that minimizes $\| \mathbf{s}^* - \mathbf{r} \|_\infty$. We show by induction that $\delta(M) = 0$. First, we start with

$$\mathbf{s}^* - \mathbf{r} = \delta(1) - \mathbf{v}(1) \ .$$

This is true, because the only difference between $\delta(1)$ and $\mathbf{s}^*$ is p_M at location k^*, which is the same as the difference between $\mathbf{r}$ and $\mathbf{v}(1)$. So, the problem of minimizing the error on the left side of the equation is the same as the problem of minimizing the error on the right side. We know from the previous arguments that $\delta_j(1) \geq 0$ for all j, and $\sum_i v_i(1) = \bar{\mathbf{p}}(M-1) = \sum_i \delta_i(1)$. Now, we show the inductive step. We need to show that $\delta_j(m) \geq 0$ for all j and m, and that to minimize the error at step m we simply choose the largest element of $\mathbf{p}$ remaining and put it in s_k where $v_k(m)$ is maximum (see beginning of proof) and then solve the minimization problem at the next step. But this is always true because we know that the largest element of $\delta(m)$ is larger than p_{M-m}, and the largest element of $\delta(m)$ is in the same location as the largest element of $\mathbf{v}(m)$. Thus, we know that by choosing $k^* = \arg\max_k v_k(m)$ at each step, we are adding a value p_{M-m} to $s_{k^*}(m)$ that is less than $\delta_{k^*}(m)$. So, we have

$$\delta(m) - \mathbf{v}(m) = \delta(m+1) - \mathbf{v}(m+1) \ ,$$

and $v_j(m) \geq 0$ for all j and m. By induction, this means that

$$\mathbf{s}^* - \mathbf{r} = \delta(M) - \mathbf{v}(M) \ ,$$

and $\delta(M) = 0$ because we have subtracted a total of 1 from $\mathbf{s}^*$, but at each stage left a nonnegative $\delta(m)$. But, $\delta(M) = \mathbf{s}^* - \mathbf{s}(M) = \mathbf{s}^* - \mathbf{s}$. This means that $\mathbf{s}^* = \mathbf{s}$. ∎

B.3 Proof of Theorems 4.3 and 4.8

Before we prove Theorem 4.3, we give a few properties of smooth norms that we will use in the proof.

The sub-level set $\mathcal{D}(0, C)$ of a smooth norm $\| \cdot \|$ has a well-defined continuous gradient $\nabla(\cdot)$ defined as

$$\nabla(\mathbf{x}) = \begin{bmatrix} \frac{d\| \mathbf{x} \|}{dx_1} & \frac{d\| \mathbf{x} \|}{dx_2} & \cdots & \frac{d\| \mathbf{x} \|}{dx_N} \end{bmatrix}^{\mathrm{T}} .$$

For an arbitrary vector u such that $\| u \| = 1$, we define $z(\beta u)$ as the unit-length inward-pointing surface normal of the sub-level set $\mathcal{D}(\beta u, C)$ at the point $x(\beta u, C)$, where $x(\beta u, C)$ is the point at which the line connecting the origin with u intersects the surface of $\mathcal{D}(\beta u, C)$. By the definition of the normal, we know that

$$z(\beta u) = -\frac{\nabla(x(\beta u, C) - \beta u)}{\| \nabla(x(\beta u, C) - \beta u) \|_2} .$$

By the definition of $x(\beta u, C)$, we know that $x(\beta u, C) = \alpha u$ for some $\alpha < \beta$ for any u. Thus,

$$\begin{aligned} z(\beta u) &= -\frac{\nabla((\alpha - \beta)u)}{\| \nabla((\alpha - \beta)u) \|_2} \\ &= -\frac{\nabla(u)}{\| \nabla(u) \|_2} \\ &= z(u) . \end{aligned}$$

This last step follows from the linearity of the gradient operator and the scaling property of the norm. The plane tangent to $\mathcal{D}(\beta u, C)$ at $x(\beta u, C)$ is defined as

$$Z(\beta u, C) = \{v \mid z^{\mathrm{T}}(u)v = z^{\mathrm{T}}(u)x(\beta u, C)\} .$$

We define $\chi(\beta u, C) = z^{\mathrm{T}}(u)x(\beta u, C)$.

We also make use of the fact that a convex object such as $\mathcal{D}(\beta u, C)$ is always contained completely in one of the two half-spaces defined by a tangent plane on the surface of the object: either

$$\mathcal{D}(\beta u, C) \subseteq \{v \mid z^{\mathrm{T}}(u)v \geq \chi(\beta u, C)\} \quad \text{or} \quad \mathcal{D}(\beta u, C) \subseteq \{v \mid z^{\mathrm{T}}(u)v \leq \chi(\beta u, C)\} . \tag{B.2}$$

Theorem B.3 (Deterministic Stability of Smooth Norms) *A deterministic dynamic scheduler model* $(\bar{B}, \bar{\mathbf{r}}, \mathcal{S})$ *is deterministically stable under scheduling rule* $Q(\cdot)$ *for all* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$ *if* $Q(\cdot)$ *is consistent with a smooth norm* $\| \cdot \|$.

Proof: Our goal is to define a region $\mathcal{C}$ outside of which the deterministic drift condition (4.1) or equivalently (4.3) holds. We define the Lyapunov function $V(\cdot)$ as

$$V(\ell[n]) = \| \ell[n] \| .$$

We will define $\mathcal{C}$ by showing that for each possible direction u (a direction is a vector u such that $\| u \| = 1$), there exists a scaling factor $\beta_m(u)$ such that $\ell^+ = \bar{\mathbf{r}} + \beta_m(u)u$ satisfies Equation (4.3) for some $\epsilon > 0$. If we define $\beta_{max} = \max_u \beta_m(u)$, then we know that when ℓ^+ is outside of $\mathcal{D}(\bar{\mathbf{r}}, \beta_{max})$, Equation (4.3) holds. Thus, the drift condition holds for $\ell \notin \mathcal{D}(0, \beta_{max})$.

Everything will now be stated in terms of an arbitrary u. To make the notation simpler, we make the assumption that $\bar{\mathbf{r}} = 0$. If it does not, we can redefine $\mathcal{S}$ such that $\bar{B} \cdot \mathcal{S}$ is shifted by $-\bar{\mathbf{r}}$, and set $\bar{\mathbf{r}}$ equal to 0. We can think of this as moving our coordinate system to be centered around $\bar{\mathbf{r}}$. This implies that the origin is in the interior of $\mathrm{ch}(\bar{B} \cdot \mathcal{S})$.

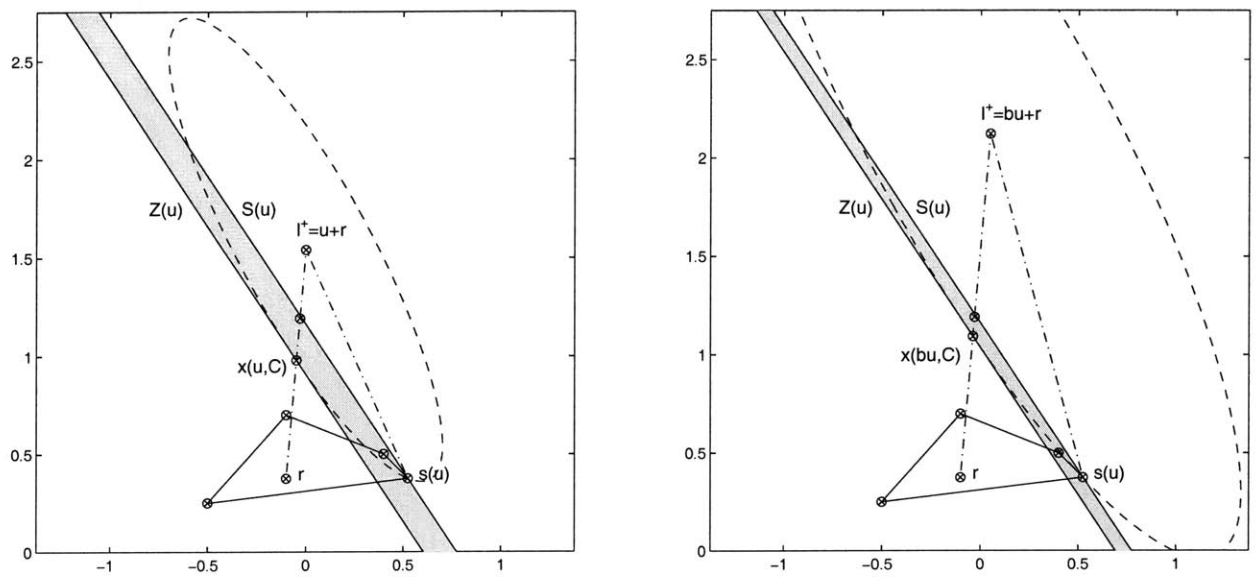


Figure B-1: Deterministic stability proof construction for u and βu.

We first need to make a few definitions related to the geometry of our norm. Given a direction u, a smooth norm $\| \cdot \|$ and a positive number β we define C to be the value of $\| \beta u - s(u) \|$, where $s(u)$ is defined as follows. Given the normal $z(u)$, we will define $s(u) \in \bar{B} \cdot \mathcal{S}$ as a schedule which maximizes $z^{\mathrm{T}}(u)s(u)$, or

$$s(u) \in \arg \max_{s \in \mathrm{CH}(\bar{B} \cdot \mathcal{S})} z^{\mathrm{T}}(u)s \ .$$

We know that an extreme point always maximizes the value of a linear function over a polytope, so we know that we can always pick $s(u)$ to be a schedule in $\mathcal{S}$. We define $S(u)$ as the plane parallel to $Z(u)$ that includes the point $s(u)$, or

$$S(u) = \{v \mid z^{\mathrm{T}}(u)v = z^{\mathrm{T}}(u)s(u)\} \ .$$

All of these quantities are depicted in Figure B-1 for an example two-dimensional constellation and norm. The region between $Z(u)$ and $S(u)$ is shaded in the figures.

We now observe that as β is increased, the constraint that ties $s(u)$ and $x(\beta u, C)$ together is defined by

$$\| \beta u - s(u) \| = \| \beta u - x(\beta u, C) \| \ .$$

But because $x(\beta u, C)$ is defined as the intersection of the surface of $\mathcal{D}(\beta u, C)$ with the line connecting the origin and βu, we can write $x(\beta u, C) = \alpha u$, where $\alpha < \beta$. Thus, we have

$$\| \beta u - s(u) \| = \| \beta u - \alpha u \| = (\beta - \alpha) \ .$$

We also know that if $\alpha \geq \epsilon_u > 0$, then the origin is on one side of $Z(u)$ and $\mathcal{D}(\beta u, C)$ is on the other. But this means that the origin (our translated $\bar{\mathbf{r}}$) is not inside $\mathcal{D}(\beta u, C)$ and so we have satisfied the drift condition for this particular value of β. If we can show that there exists a finite β_m such that $\alpha \geq \epsilon_u$ for all $\beta > \beta_m$ then we are done.

Another way of showing this is to establish that

$$\| \beta u - s(u) \| \leq \beta - \epsilon_u$$

for $\beta > \beta_m$. Consider the set $\mathcal{V} = \mathcal{D}(0, \beta)$; the above inequality is equivalent to saying that the point $\beta u - s(u)$ lies in the interior of this set. The point βu lies on the surface of this set, and the outward-facing normal is equal to $z(u)$ and so the tangent plane $\hat{Z}(\beta u)$ to the surface of $\mathcal{V}$ at the point βu is parallel to $Z(u)$. We also know that $z^{\mathrm{T}}(u)s(u) > 0$ because $z^{\mathrm{T}}(u)s(u) > z^{\mathrm{T}}(u)\mathbf{r}$ for any $\mathbf{r} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$, including $\mathbf{r} = 0$.

Consider the line $\hat{S}$ which passes through both βu and $\beta u - s(u)$. This line must intersect the surface of $\mathcal{V}$ at exactly two points because it is not tangent to the surface $(z^{\mathrm{T}}(u)s(u) > 0)$. We know that one of these points is βu by construction. Call the other point $\hat{s}$. If the distance between βu and $\hat{s}$ is δ, then we know that δ grows proportionately with β; it simply scales with the size of $\mathcal{V}$. We also know that $\beta u - s(u)$ and $\hat{s}$ lie on the same side of $\hat{Z}(\beta u)$ and that the distance between βu and $\beta u - s(u)$ is fixed, so eventually after β is larger than some $\beta_m(u)$, $\beta u - s(u)$ will lie in the interior of $\mathcal{V}$. We can then pick a value ϵ_u small enough and a $\beta_m(u)$ large enough so that $\| \beta u - s(u) \| \leq \beta - \epsilon_u$ for all $\beta > \beta_m(u)$.

Putting this all together, we have

$$\begin{aligned} V(\beta u - \mathbf{s}^*(u)) - V(\beta u) &= \| \beta u - \mathbf{s}^*(u) \| - \| \beta u \| \\ &\leq \| \beta u - s(u) \| - \| \beta u \| \\ &\leq -\epsilon_u \end{aligned}$$

for $\beta > \beta_m(u)$ where $\mathbf{s}^*(u)$ is the point in $\bar{B} \cdot \mathcal{S}$ closest to βu under the norm $\| \cdot \|$.

This reasoning holds for all u, so we pick $\beta_{max} = \max_u \beta_m(u)$ and $\epsilon = \min_u \epsilon_u$ and we satisfy the deterministic drift condition for $\ell^+ \notin \{v \mid \| v \| \leq \beta_{max}\} = \mathcal{D}(0, \beta_{max})$ and $\epsilon > 0$. This is true if we assume that $\bar{\mathbf{r}} = 0$. For general $\bar{\mathbf{r}}$, we simply translate so that $\ell^+ \notin \mathcal{D}(\bar{\mathbf{r}}, \beta_{max})$. This tells us that $\ell \notin \mathcal{D}(0, \beta_{max})$ and thus $\mathcal{C} = \mathcal{D}(0, \beta_{max})$. ■

Theorem B.4 (Stability of the 2-Norm for Independent Inputs) *Given dynamics*

$$\ell[n+1] = \ell[n] - B[n+1]\mathbf{s}[n] + \mathbf{r}[n+1] \ ,$$

if we choose $\mathbf{s}[n]$ *so that*

$$\mathbf{s}[n] = \arg\min_{s \in \mathcal{S}} \mathbf{E}\left[\| \ell[n] - B[n+1]\mathbf{s} + \mathbf{r}[n+1] \|_2^2 \mid \ell[n]\right] \ ,$$

and if $B[n]$ *and* $\mathbf{r}[n]$ *are IID and have finite means and second moments such that* $\bar{\mathbf{r}} \in \mathrm{ch}(\bar{B} \cdot \mathcal{S})$, *then the Markov chain* $\ell[n]$ *satisfies the stochastic drift condition (4.5–4.6).*

Proof: Define $V(\ell) = \| \ell \|_2^2$. In order to prove the theorem, we will show that the following inequality holds:

$$\mathbf{E}_\ell[V(\ell[n+1])] - V(\ell[n]) \leq -\epsilon$$

for all ℓ such that $\| \ell \|_2 > \alpha^*$ for some positive bounded α^* and that

$$\mathbf{E}_\ell[V(\ell[n+1])] - V(\ell[n]) \leq A < \infty$$

for all ℓ such that $\| \ell \|_2 \leq \alpha^*$. Then we will have satisfied the stochastic drift condition with $\mathcal{C} = \{\ell \mid \| \ell \|_2 > \alpha^*\}$.

We can expand the above inequality as

$$\begin{aligned} \mathbf{E}_\ell\left[\| \ell \|_2^2 - 2\ell^{\mathrm{T}}(B[n+1]\mathbf{s}[n] - \mathbf{r}[n+1]) + \| B[n+1]\mathbf{s}[n] - \mathbf{r}[n+1] \|_2^2\right] - \| \ell \|_2^2 &\leq -\epsilon \\ \mathbf{E}_\ell\left[\| B[n+1]\mathbf{s}[n] - \mathbf{r}[n+1] \|_2^2\right] - 2\ell^{\mathrm{T}}\left(\bar{B}\mathbf{s}[n] - \bar{\mathbf{r}}\right) &\leq -\epsilon \ . \end{aligned}$$

The first term in the inequality can be bounded by some positive number γ independent of ℓ, so we just need to show that the second term can become sufficiently negative. To show this it will be sufficient to show that for ℓ outside of some bounded region,

$$\ell^{\mathrm{T}}\left(\bar{B}\mathbf{s}[n]-\bar{\mathbf{r}}\right) \geq \gamma \geq \frac{1}{2}\left[\mathbf{E}_{\ell}\left[\| B[n+1]\mathbf{s}[n]-\mathbf{r}[n+1] \|_2^2\right]+\epsilon\right] .$$

If we show that $\ell^{\mathrm{T}}(\bar{B}\mathbf{s}[n]-\bar{\mathbf{r}}) \geq \rho'$ for positive ρ' and all ℓ such that $\| \ell \|_2^2 = 1$ then $\alpha^* \ell^{\mathrm{T}}(\bar{B}\mathbf{s}[n]-\bar{\mathbf{r}}) \geq \alpha^*\rho'$. If we choose α^* to be large enough, we have $\alpha^*\rho' \geq \gamma$.

Assume that $\mathbf{s}[n]$ is chosen to minimize $\mathbf{E}_{\ell}\left[\| \alpha\ell - B[n+1]\mathbf{s}[n]+\mathbf{r}[n+1] \|_2^2\right]$ for some given ℓ such that $\| \ell \|_2 = 1$. The following holds for any $s \in \mathcal{S}$:

$$\mathbf{E}_{\ell}\left[\| \alpha\ell - B[n+1]s+\mathbf{r}[n+1] \|_2^2\right] - \mathbf{E}_{\ell}\left[\| \alpha\ell - B[n+1]\mathbf{s}[n]+\mathbf{r}[n+1] \|_2^2\right] \geq 0$$

Now consider a schedule s chosen to maximize $\ell^{\mathrm{T}}\bar{B}s$. This gives us

$$\ell^{\mathrm{T}}\bar{B}s' - \ell^{\mathrm{T}}\bar{B}s \leq 0$$

for any $s' \in \mathcal{S}$. Combining these two inequalities:

$$\begin{aligned} 0 \leq \ \ell^{\mathrm{T}}(\bar{B}s-\bar{B}\mathbf{s}[n]) \ &\leq \frac{\mathbf{E}_{\ell}\left[\| B[n+1]s-\mathbf{r}[n+1] \|_2^2 - \| B[n+1]\mathbf{s}[n]-\mathbf{r}[n+1] \|_2^2\right]}{2\alpha} \\ 0 \leq \ \ell^{\mathrm{T}}(\bar{B}s-\bar{B}\mathbf{s}[n]) \ &\leq \frac{\beta}{2\alpha} , \end{aligned}$$

so $\ell^{\mathrm{T}}\bar{B}\mathbf{s}[n]$ is a bounded distance away from $\ell^{\mathrm{T}}\bar{B}s$, and the distance decreases with α. The quantity β is greater than 0, independent of ℓ and is chosen as $\beta = \max_s \mathbf{E}_{\ell}\left[\| B[n+1]s+\mathbf{r}[n+1] \|_2^2\right] - \min_s \mathbf{E}_{\ell}\left[\| B[n+1]s+\mathbf{r}[n+1] \|_2^2\right]$. If $\bar{\mathbf{r}}$ is in $\mathrm{ch}(\bar{B}\cdot\mathcal{S})$ then we can call $\rho > 0$ the minimum distance between $\bar{\mathbf{r}}$ and the surface of the convex hull. Then we simply choose $\alpha(\ell) > \beta/2\rho$ and we will have that $\ell^{\mathrm{T}}(\bar{B}s - \bar{B}\mathbf{s}[n]) = \beta/2\alpha < \rho$. For any given ℓ we have

$$\rho \leq \ell^{\mathrm{T}}(\bar{B}s-\bar{\mathbf{r}}) = \ell^{\mathrm{T}}(\bar{B}s-\bar{B}\mathbf{s}[n]) + \ell^{\mathrm{T}}(\bar{B}\mathbf{s}[n]-\bar{\mathbf{r}}) = \beta/2\alpha+\rho' .$$

But, since $\beta/2\alpha < \rho$ then $\rho' > 0$. So, if we choose α^* such that

$$\alpha^*\rho' \geq \gamma ,$$

then we have a finite bound on the size of $\ell[n]$ beyond which the expected value of $V(\ell[n])$ decreases by 1 with each step. The only thing left is to show that the expected jump in $V(\ell[n])$ from n to $n+1$ is always finite. The finiteness follows from the boundedness of the moments of $\mathbf{r}[n]$ and the fact that $\| \ell[n] \|_2^2 \leq \alpha^*$. ∎

B.4 Proof of Theorem 6.1

Definition B.1 (Lattice A_N) *The lattice A_N consists of all N-dimensional integer vectors a such that $\sum_i a_i = 0$.*
The notation $\sigma \cdot (A_N + m)$ or $A_N(\sigma, m)$ denotes the set of vectors b such that $b = \sigma a'$ where $\sum_i a'_i = m$.

We will use the shorthand $\mathcal{S}$ when we mean $\mathcal{S}(\mathbf{p}_0)$, the constellation generated by the size vector $\mathbf{p}_0$. According to this definition, $\mathcal{S} \subset A_N(1/M, M)$. Specifically, $\mathcal{S}$ contains only those vectors $b \in A_N(1/M, M)$ such that each component $b_i \geq 0$.

Definition B.2 (Allocation Cell) *The* allocation cell AC($\mathbf{s}$) *of a point* $\mathbf{s} \in \mathcal{S}$ *is the region surrounding* $\mathbf{s}$ *such that for every point t in the cell,*
$\| t - \mathbf{s} \|_\infty < \| t - \mathbf{s}' \|_\infty$ *for any other* $\mathbf{s}' \in \mathcal{S}$.

Definition B.3 (Basic Allocation Cell) *The* basic allocation cell BAC($\mathbf{s}$) *of a point* $\mathbf{s} \in A_N(1/M, M)$ *is the region surrounding* $\mathbf{s}$ *such that for every point t in the cell,*
$\| t - \mathbf{s} \|_\infty < \| t - \mathbf{s}' \|_\infty$ *for any other* $\mathbf{s}' \in A_N(1/M, M)$.

The basic allocation cell is equal to the allocation cell except for the points $\mathbf{s}$ that are on the boundary of $\mathcal{S}$. For these points, the allocation cell is not bounded, but extends outwards, because the constellation is bounded. The union of all allocation cells is the entire space $\mathbb{R}^N$, while the union of all basic allocation cells is localized around the constellation.

Definition B.4 (Root Cell) *The* root cell *of the lattice $A_N(1/M, M)$ is the region of points x such that $\sum_i x_i = 0$ and $\| x \|_\infty < \| x - v \|_\infty$ for any other $v \in A_N(1/M, 0)$.*

We will also use these definitions for all vectors $\mathbf{r} \in \mathbb{S}_N$. The allocation cell of $\mathbf{r}$ is the root cell centered around $\mathbf{r}$, i.e. the region of points ℓ such that $\sum_i \ell_i = 1$ and $\ell - \mathbf{r}$ is closer to 0 than to any other $\mathbf{s}' \in A_N(1/M, 0)$. We denote this by BAC($\mathbf{r}$). We also note here that BAC($\mathbf{s}$) is also the root cell centered around $\mathbf{s} \in \mathcal{S}$ similarly.

Lemma B.3 (Root Cell) *The root cell of $A_N(1/M, M)$ is the region such that for x in the root cell, $\max_i x_i - \min_j x_j < 1/M$.*
Proof: Call $x_M = \max_i x_i$, and $x_m = \min_j x_j$. Show that if $x_M - x_m \geq 1/M$, there is a nonzero point in $A_N(1/M, 0)$ that at least as close to x as is 0. Define v so that $v_k = 0$ except that $v_i = 1/M$ and $v_j = -1/M$. It follows that $v \in A_N(1/M, 0)$. We have to show that $\| x - v \|_\infty \leq \| x \|_\infty$. First, we know that $\| x \|_\infty = \max(|x_M|, |x_m|)$. If $x_M - x_m \geq 1/M$, then $x_m \leq 0$ and $x_M \geq 0$. We also know that $x_m + 1/M \leq x_M$ and $x_M - 1/M \geq x_m$. So,

$$\begin{aligned} \| x - v \|_\infty &\leq \max\left(\left| x_M - \frac{1}{M} \right|, \left| x_m + \frac{1}{M} \right|, \{|x_k|, k \neq m, M\} \right) \\ &\leq \max\left(|x_m|, |x_M|, \{|x_k|, k \neq m, M\} \right) \\ &= \| x \|_\infty . \end{aligned}$$

Now, we need to show that if $x_M - x_m < 1/M$, then every nonzero point in $A_N(1/M, 0)$ is farther from x than 0. To do this, we show that for any nonzero point $v \in A_N(1/M, 0)$, we always increase the distance between x and v if we increase any nonnegative element of v by $1/M$ and decrease some other nonpositive element by $1/M$. Because we can construct any nonzero v in this way, all other points v must be further away from x than 0. Thus, 0 is the minimum.

Given v (could be equal to 0), pick a pair of indices i and j for which $v_i \geq 0 \geq v_j$. Construct v' equal to v except that $v'_i = v_i + 1/M$ and $v'_j = v_j - 1/M$. We have to show that $\| x - v' \| > \| x - v \|$ for the following four cases:

$v_i > 0 > v_j$: No matter what, $|v_i - x_i|$ and $|v_j - x_j|$ will be increased, which can only increase the distance between s' and x.

$v_i = 0 > v_j$: The only complication is if $|v_i - x_i|$ was the maximum distance, and $|s'_i - x_i| < |v_i - x_i|$ But, in this case, $|v_j - x_j| \geq 1 > |v_i - x_i|$, so the distance is increased, anyway.

$v_i > 0 = v_j$: Same as previous case.

$v_i = 0 = v_j$: First, if $x_i \leq 0$ or $x_j \geq 0$, this case is similar to one or the other of the previous two cases. So, we now assume that $x_i > 0$ and $x_j < 0$. All we have to show is that $\max(|x_i|, |x_j|) < \max(|x_i - 1/M|, |x_j + 1/M|)$. It is obvious that $\max(|x_i|, |x_j|) \leq \max(|x_M|, |x_m|)$. It is also clear that $|x_i - 1/M| \geq |x_M - 1/M|$ and $|x_j + 1/M| \geq |x_m + 1/M|$. So, we just have to show that $\max(|x_M|, |x_m|) < \max(|x_M - 1/M|, |x_m + 1/M|)$. This follows because $0 > x_m > x_M - 1/M$ and $0 < x_M < x_m + 1/M$.

■

Corollary B.1 *The basic allocation cell for* $\mathbf{s} \in A_N(1/M, M)$ *is the root cell centered on* $\mathbf{s}$.

Theorem B.5 *If* $\text{BAC}(\mathbf{r}) \cap \text{BAC}(\mathbf{s}) = \emptyset$ *for all* $\mathbf{s} \in (A_N(1/M, M) - \mathcal{S})$*, then* $\ell[n] \in \text{BAC}(0)$.
Proof: Proof by induction. First, if $\ell[0] = 0$, it is true for $n = 0$. Now, assume it is true for n, and we will show it for $n + 1$. By the definition of the quantizing scheduler, $\ell[n] + \mathbf{r} \in \text{AC}(\mathbf{s}[n])$. By the assumption in the theorem, $\ell[n] + \mathbf{r} \in \text{BAC}(\mathbf{s}[n])$. By the definition of the basic allocation cell, $\ell[n] + \mathbf{r} - \mathbf{s}[n] \in \text{BAC}(0)$ and thus $\ell[n+1] \in \text{BAC}(0)$. ■

All of this tells us that $\ell[n + 1]$ is in the root cell. This means that $\sum_i \ell_i[n + 1] = 0$ and that $\max_j \ell_j[n + 1] - \min_j \ell_j[n + 1] < 1/M$. From this we can see that the largest any $\ell_i[n + 1]$ can be is $((N - 1)/N) \cdot (1/M) = E(\mathbf{p}_0)$. This follows because in order to make one element as large as possible, we must make the others as small as possible. This means putting $N - 1$ elements at $-1/(NM)$ and the last element at $(N-1)/(NM)$. A similar argument establishes that the smallest $\ell_i[n + 1]$ can be is $-(N - 1)/(NM)$.

Corollary B.2 (Lag Bound) *If* $\text{BAC}(\mathbf{r}) \cap \text{BAC}(\mathbf{s}) = \emptyset$ *for all* $\mathbf{s} \in (A_N(1/M, M) - \mathcal{S})$*, then* $\max_j \ell_j[n + 1] - \min_j \ell_j[n + 1] < 1/M$. *This implies that* $|\ell_i[n + 1]| < ((N - 1)/N) \cdot (1/M)$ *for all* i.

This corollary is equivalent to Theorem 6.1, without the specific characterization of the possible values of $\mathbf{r}$. Now, we turn to the specification of the region $\bar{R}$ such that $\text{BAC}(\mathbf{r}) \cap \text{BAC}(\mathbf{s}) = \emptyset$ for all $\mathbf{r} \in \bar{R}$ and $\mathbf{s} \in (A_N(1/M, M) - \mathcal{S})$.

Theorem B.6 *If* $(s_i - r_i) - (s_j - r_j) > 2/M$ *for some* i, j *for all* $\mathbf{s} \in A_N(1/M, M)$, $\mathbf{r} \in \mathbb{S}_N$*, then* $\text{BAC}(\mathbf{r}) \cap \text{BAC}(\mathbf{s}) = \emptyset$.
Proof: We need to show first that $(\mathbf{r} + \mathbf{s})/2 \in \text{BAC}(\mathbf{r}) \cap \text{BAC}(\mathbf{s})$ if the two regions intersect. This can be shown using convexity and symmetry properties. First, fix $\mathbf{s}$ and $\mathbf{r}$ and define a vector $v = \mathbf{s} - \mathbf{r}$ such that $\text{BAC}(\mathbf{r})$ intersects $\text{BAC}(\mathbf{s})$ at at least one point t. Define $x = t - \mathbf{r}$. By symmetry of the root cell, $t' = \mathbf{s} - x$ is also in the intersection. But, because the two cells are convex, all convex

combinations of t and t' must be in the intersection. Thus, $(t+t')/2 = (\mathbf{r}+x+\mathbf{s}-x)/2 = (\mathbf{r}+\mathbf{s})/2$ is in the intersection.

Finally, define $y = \mathbf{r} - (\mathbf{r}+\mathbf{s})/2 = (\mathbf{r}-\mathbf{s})/2$. From the definition of BAC($\cdot$), $y_i - y_j \leq 1/M$ which implies that $(r_i - s_i) - (r_j - s_j) \leq 2/M$. ■

Now, we have a characterization of the set $\bar{R}$. From the above, we know that BAC($\mathbf{r}$)$\cap$BAC($\mathbf{s}$) $= \emptyset$ if there exist indices i and j such that $(s_i - r_i) - (s_j - r_j) > 2/M$. This is equivalent to saying that $\mathbf{r}$ is not in the region defined as a root cell of the lattice $A_N(2/M, 0)$ centered around $\mathbf{s}$.

The basic interpretation of the uniform rate set is that if $\mathbf{r} \in \bar{R}$, then we know that $\ell[n]$ will always be inside the union of all of the basic allocation cells of the points in $\mathcal{S}$.

B.5 Proofs from Chapter 7

The full proofs of Lemmas 7.1, 7.2, 7.3, and 7.4 are given here.

Lemma B.4 (Existence and Completeness of Essential Classes) *Every fair class $C(m)$, for all $N \geq 2$ and $0 \leq m \leq N-1$, is an essential class and there are no other essential classes.*
Proof: To establish that the fair classes are essential, we need to show that every state in $C(m)$ communicates with every other state in $C(m)$ and that no state in $C(m)$ leads to any state not in $C(m)$.

First, we show that if $\mathbf{w}[n_0] \in C(m)$ at time n_0, $\mathbf{w}[n] \in C(m)$ for all $n \geq n_0$. The following facts establish this:

(i) The total wealth in any state in $C(m)$ is $N-(m-1)$.

(ii) Any state reached by transitioning out of any state in $C(m)$ maintains a total wealth of $N-(m-1)$.

(iii) The maximum wealth any individual process can have in any state reached is 2.

(iv) Furthermore, only one process can have a wealth of 2, the others must be 1 or 0.

All of these facts follow easily from the definition of $C(m)$ if you think of the auction in two stages: first, some process wins the auction and pays the second-highest bid (which is always equal to 1 in $C(m)$), and second, one more unit of wealth is given to some process. After the first step, no process can have more than 1 unit of wealth because the highest bidder could only have bid at most 2, and must always pay 1. This leaves some number (≥ 2) of processes with 1 unit of wealth, and some with none. After the second step, either $N-(m-1)$ processes have 1 unit of wealth and the rest have none, or $N-m$ have 1, one has 2 and the rest have none, which is exactly the definition of $C(m)$. So, once we are in $C(m)$, we can only transition into another state in $C(m)$.

We now show that all states in $C(m)$ communicate. The following three statements establish this:

(i) For a given $\mathbf{k}(m)$, all $\omega_i^{\mathbf{k}(m)}$ are fully connected. The only difference between these states is i, the process with a wealth of 2. $\mathbf{w}[n]$ will transition from $\omega_j^{\mathbf{k}(m)}$ to $\omega_i^{\mathbf{k}(m)}$ for any $i, j \notin \mathbf{k}(m)$, when process i receives the unit of wealth (probability p_i) after j wins the next auction.

(ii) Given any two $\omega_0^{\mathbf{k}(m-1)}$ and $\omega_0^{\ell(m-1)}$, there is at least one path between them as follows: consider the two sets $\Omega_{k\ell} = \mathbf{k}(m-1) - \ell(m-1)$ and $\Omega_{\ell k} = \ell(m-1) - \mathbf{k}(m-1)$ (denote the size of $\Omega_{k\ell}$ and $\Omega_{\ell k}$ by c). If the processes in $\Omega_{k\ell}$ win the next c auctions in order, and the processes in $\Omega_{\ell k}$ receive the next c units of wealth in order, we will be left in state $\omega_0^{\ell(m-1)}$ if we started in $\omega_0^{\mathbf{k}(m-1)}$.

(iii) Any $\omega_i^{\mathbf{k}(m)}$ reaches at least one $\omega_0^{\mathbf{k}(m-1)}$ whenever some process $j \in \mathbf{k}(m)$ receives a dollar after an auction, and any state $\omega_0^{\mathbf{k}(m-1)}$ reaches some $\omega_i^{\mathbf{k}(m)}$ ($\mathbf{k}(m-1) \subset \mathbf{k}(m)$) if process $k = \mathbf{k}(m) - \mathbf{k}(m-1)$ wins the auction and some process $i \notin \mathbf{k}(m)$ receives a dollar after the auction.

We have shown that once we enter a state in $C(m)$, we always stay in states in $C(m)$, and that all states in $C(m)$ communicate, so $C(m)$ is an essential class.

We can see that these are the only essential classes because there is a finite length path between any other state and a state in one of the sets $C(m)$. This means that all other states are inessential because they lead to some state which which they do not communicate [9]. ∎

Lemma B.5 (Inevitability of Fair Classes) *If n_τ is the first round during which $\mathbf{w}[n_\tau] \in C(m)$ for some m, then $\mathbf{E}[n_\tau] < \infty$ for any given $\mathbf{w}[0]$.*
Proof: Define n_i for $i = 1, \ldots, \tau - 1$ as the rounds during which $\mathbf{w}[n_i] = w[n_i] \cdot \mathbf{e}_{u[n_i]}$ where $\mathbf{e}_u$ is the vector whose u-th component is 1 and all other components are 0. The value τ is defined such that $\mathbf{w}[n_\tau]$ is first in one of the fair classes $C(m)$. We define $n_0 = 0$. The states $w \cdot \mathbf{e}_u$ are of interest because they are the only ones from which the maximum wealth over all users can increase. We begin by showing that $\mathbf{E}[\tau] < \infty$ for any initial condition $\mathbf{w}[0]$. Then, define $\eta_i = n_i - n_{i-1}$ and we will show that $\mathbf{E}[\sum_{i=1}^{\tau} \eta_i] = \mathbf{E}[n_\tau] < \infty$.

We can define a Markov chain $\mathbf{W}[m] = \mathbf{w}[n_m]$ with appropriate transition probabilities determined from the transition probabilities of $\mathbf{w}[n]$. The chain $\mathbf{W}[m]$ operates on a smaller state space, consisting only of vectors of the form $w \cdot \mathbf{e}_u$ for $w \geq 0$ and $1 \leq i \leq N$ augmented by a single state ω representing all of the states in the fair classes $C(m)$. Thus, $\mathbf{W}[\tau + 1] = \omega$.

In order to make $\mathbf{W}[m]$ an irreducible chain, which is technically needed for the following proof of positive recurrence, we will assume that the chain $\mathbf{W}[m]$ moves from ω to the state 0 with probability 1. To show that $\mathbf{W}[m]$ is irreducible, we note that there is a finite length path (of positive probability) between any two possible states of $\mathbf{W}[m]$.

To establish that $\mathbf{E}[\tau] < \infty$, we will show that the chain $\mathbf{W}[m]$ is positive recurrent. For an irreducible Markov chain on a countable state space to be positive recurrent, we only need to verify that the drift condition

$$\mathbf{E}[V(\mathbf{W}[m+1]) - V(\mathbf{W}[m]) \mid V(\mathbf{W}[m]) = v] \leq -\epsilon \qquad v \notin \mathcal{R}$$

holds for some stochastic Lyapunov function $V(\mathbf{w})$, $\epsilon > 0$, and some finite set $\mathcal{R}$ [53]. We also must show that the drift is bounded for $v \in \mathcal{R}$.

Define the Lyapunov function to be the maximum wealth over all users $V(\mathbf{W}[m]) = \max_i W_i[m] = w[n_m]$. We will now find an upper bound $\bar{w}$ on the first term in the drift condition that still satisfies the condition outside of a finite set of states.

Given that the chain is currently in some state $\mathbf{W}[m] = w[n_m]\mathbf{e}_{u[n_m]} = w\mathbf{e}_u$, there are two possibilities. The chain can either transition to $(w+1)\mathbf{e}_u$ and increase the maximum wealth or to some other point $w'\mathbf{e}_{u'}$ for some other pair of w' and u'. This leaves us with

$$\begin{aligned}
\mathbf{E}[V(\mathbf{W}[m+1])] &= p_u(w+1) + \mathbf{E}[w'] \\
&= p_u(w+1) + \sum_{w',u'} w' \cdot \Pr[\mathbf{W}[m] \to w'\mathbf{e}_{u'}] \\
&\leq p_u(w+1) + \bar{w} \sum_{w',u'} \Pr[\mathbf{W}[m] \to w'\mathbf{e}_{u'}] \\
&= p_u(w+1) + (1-p_u)\bar{w}
\end{aligned}$$

for some $\bar{w}$ which is an upper bound on the possible values of w'.

The fastest way (which also results in the largest remaining w') to get from $\mathbf{W}[m] = \mathbf{w}[n_m] = w\mathbf{e}_u$ to $\mathbf{W}[m+1] = \mathbf{w}[n_{m+1}] = w'\mathbf{e}_{u'}$ is for the u'-th user to receive income at every round until $w_{u'}[n+k] = w_u[n+k]$ at which point the u'-th user wins and $\mathbf{W}[m+1] = w'\mathbf{e}_{u'} = \bar{w}\mathbf{e}_{u'}$. When this happens, $w - \sum_{j=1}^{k} j = k = w' - 1$. In other words the first time that the wealth for the two users meet or cross, they are equal and the next step brings us to $w'e_{u'}$. This means that user u' gains 1 dollar every round after n_m and ends up with k dollars when both users have the same wealth. Then at round n_{m+1}, user u' has $k+1$ dollars, so $w' = k+1 = \eta_{m+1}$. So,

$$\begin{aligned}
w' &\leq w + 1 - \sum_{k=1}^{w'-1} k \\
\frac{(w')^2}{2} + \frac{w'}{2} - 1 &\leq w \\
w' &\leq \sqrt{2w + \frac{7}{4}} - \frac{1}{2} \\
w' &\leq \left\lceil \sqrt{2w + \frac{7}{4}} - \frac{1}{2} \right\rceil = \bar{w} \, .
\end{aligned}$$

We can now write

$$\begin{aligned}
\mathbf{E}[V(\mathbf{W}[m+1])] &\leq p_u(w+1) + (1-p_u)\bar{w} \\
&= p_u(w+1) + (1-p_u)\left\lceil \sqrt{2w + \frac{7}{4}} - \frac{1}{2} \right\rceil \, .
\end{aligned}$$

It is clear from this that we can pick a $w = w(u)$ large enough for the upper bound on the right side of this last equation to be smaller than $w - \epsilon$ and thus satisfy the drift condition. So, we define $\mathcal{R}$ to be the set of vectors $w\mathbf{e}_u$ such that $w < w(u)$ for all u. The boundedness of $V(\cdot)$ within $\mathcal{R}$ is obvious.

We have now established that $\mathbf{W}[m]$ is positive recurrent and thus that $\mathbf{E}[\tau] < \infty$. All that is left is to show that $\mathbf{E}[\sum_{i=1}^{\tau} \eta_i] < \infty$. Assume we start at some state $\mathbf{w}[0]$. The expected value of η_1 is finite because either we start in an appropriate state $w[0]e_u$ or the maximum wealth deterministically decreases until we reach an appropriate state. The maximum wealth strictly decreases when the second highest wealth is greater than 1; if it is equal to 1, then we have to wait some number of rounds until the user with the second highest wealth has 2 dollars. But this number is just a geometric random variable, and so has a finite expectation.

Let us assume that the maximum starting wealth $\max_i w_i[0] = \omega$ and fix τ at some value. Then, the largest maximum wealth we can possibly have during the evolution of the chain is less than

$\omega + \tau$ because we can only increase the maximum wealth by at most 1 per round and we can only do it at most τ times before we have to return to one of the essential classes. Thus, the largest number of decreasing steps we can take is $\omega + \tau$. In addition to decreasing steps we have a number of steps wasted waiting for the second highest wealth to move from 1 to 2 at most $\omega + \tau$ times (once at every decrease, a very conservative estimate). The expected time waiting in each of these states is bounded above by $N\mu$, where μ is the expected value of a geometric random variable with distribution $p(n) = (1 - p_m)p_m^n$ where p_m is the smallest probability of receiving income over all of the users.

Finally we have that the expected waiting time $\sum_{i=1}^{\tau} \eta_i$, given τ and ω is less than $(N\mu + 1) \cdot (\tau + \omega) + \tau$. Thus we have:

$$\begin{aligned}
\mathbf{E}[n_\tau] = \mathbf{E}\left[\sum_{i=1}^{\tau} \eta_i\right] &\leq \mathbf{E}[(N\mu + 1)\cdot(\tau + \omega) + \tau] \\
&= (N\mu + 1)\cdot(\mathbf{E}[\tau] + \omega) + \mathbf{E}[\tau] < \infty .
\end{aligned}$$

■

Lemma B.6 (Limiting Probabilities) *Given* $\mathbf{w}[n]$ *is in a fair class* $C(m)$*, the limiting probabilities of the states in the class are given by*

$$\begin{aligned}
\pi_i^{\mathbf{k}(m)} &= p_i \cdot \frac{r_m^{\mathbf{k}(m)}}{r_m} \\
\pi_0^{\mathbf{k}(m-1)} &= (N - (m-1)) \cdot \frac{r_{m-1}^{\mathbf{k}(m-1)}}{r_m}
\end{aligned}$$

(Note that for $m = 0$*, there are no* $\pi_0^{\mathbf{k}(m-1)}$ *states, and for* $m = 1$*, there is only one,* π_0*.)*

Proof: We need to verify two things: first, that the probabilities sum to 1, and second, that they satisfy the equation $\Pi(m)\mathbf{P}(m) = \Pi(m)$, where $\Pi(m)$ is a row vector of limiting probabilities and $\mathbf{P}(m)$ is the matrix of transition probabilities between states in $C(m)$.

So, to show that they sum to 1, just verify the following equation:

$$\begin{aligned}
\sum_{\mathbf{k}(m)} \sum_{i \notin \mathbf{k}(m)} \pi_i^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m-1)} \pi_0^{\mathbf{k}(m-1)} &= 1 \\
\sum_{\mathbf{k}(m)} \sum_{i \notin \mathbf{k}(m)} p_i \cdot \frac{r_m^{\mathbf{k}(m)}}{r_m} + \sum_{\mathbf{k}(m-1)} (M - (m-1)) \cdot \frac{r_{m-1}^{\mathbf{k}(m-1)}}{r_m} &= 1 \\
\sum_{\mathbf{k}(m)} \sum_{i \notin \mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m-1)} (M - (m-1)) r_{m-1}^{\mathbf{k}(m-1)} &= r_m \\
\sum_{\mathbf{k}(m)} r_m^{\mathbf{k}(m)} \sum_{i \notin \mathbf{k}(m)} p_i + \sum_{\mathbf{k}(m)} \sum_{i \in \mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} &= r_m \\
\sum_{\mathbf{k}(m)} r_m^{\mathbf{k}(m)} \left(\sum_{i \notin \mathbf{k}(m)} p_i + \sum_{i \in \mathbf{k}(m)} p_i \right) &= r_m \\
\sum_{\mathbf{k}(m)} r_m^{\mathbf{k}(m)} &= r_m .
\end{aligned}$$

The last statement follows by the definition of r_m.

Next, we need to verify that these probabilities are actually limiting probabilities, i.e. they satisfy $\Pi(m)\mathbf{P}(m) = \Pi(m)$. The equations that need to be satisfied are in two parts. The first set of equations sets the limiting probability of being in state $\omega_i^{\mathbf{k}(m)}$ equal to the sum of two expressions. The first expression is a sum over all states $\omega_j^{\mathbf{k}(m)}$; they can be reached if j receives the unit of wealth after i wins the next auction when starting in state $\omega_j^{\mathbf{k}(m)}$. The second expression is a sum over all states $\omega_0^{\mathbf{k}(m-1)}$ that can reach $\omega_i^{\mathbf{k}(m)}$. The only possible states are the ones for which $\mathbf{k}(m)$ and $\mathbf{k}(m-1)$ differ by one process, $j = \mathbf{k}(m) - \mathbf{k}(m-1)$; we reach $\omega_i^{\mathbf{k}(m)}$ if j wins the next auction and i receives the next unit of wealth (which happens with probability $\frac{p_i}{N-(m-1)}$).

First, for all $\mathbf{k}(m)$, and for all $i \notin \mathbf{k}(m)$ we need:

$$\begin{aligned}
\pi_i^{\mathbf{k}(m)} &= \sum_{j \notin \mathbf{k}(m)} p_i \pi_j^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m-1):\mathbf{k}(m-1)\subset\mathbf{k}(m)} \frac{p_i}{N-(m-1)} \pi_0^{\mathbf{k}(m-1)} \\
p_i \cdot r_m^{\mathbf{k}(m)} &= \sum_{j \notin \mathbf{k}(m)} p_i p_j \cdot r_m^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m-1):\mathbf{k}(m-1)\subset\mathbf{k}(m)} \frac{p_i}{N-(m-1)} \cdot (N-(m-1)) r_{m-1}^{\mathbf{k}(m-1)} \\
r_m^{\mathbf{k}(m)} &= \sum_{j \notin \mathbf{k}(m)} p_j \cdot r_m^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m-1):\mathbf{k}(m-1)\subset\mathbf{k}(m)} r_{m-1}^{\mathbf{k}(m-1)} .
\end{aligned} \tag{B.3}$$

Now, we just need to show

$$\sum_{j \in \mathbf{k}(m)} p_j \cdot r_m^{\mathbf{k}(m)} = \sum_{\mathbf{k}(m-1):\mathbf{k}(m-1)\subset\mathbf{k}(m)} r_{m-1}^{\mathbf{k}(m-1)} , \tag{B.4}$$

and we will be done. The LHS of Equation (B.4) has m terms in the summation, one for each element of $\mathbf{k}(m)$. The RHS also only has m terms, because there are only m possible values of $\mathbf{k}(m-1)$ such that $\mathbf{k}(m-1) \subset \mathbf{k}(m)$. We now show that for any term in the summation on the LHS, there is a corresponding equal term on the RHS. Start with some $j \in \mathbf{k}(m)$. Then, choose $\mathbf{k}(m-1) = \mathbf{k}(m) - \{j\}$. By definition, $r_{m-1}^{\mathbf{k}(m-1)} = p_j r_m^{\mathbf{k}(m)}$, and we have a correspondence between the terms on the LHS and RHS of Equation (B.4).

Now, going back to Equation (B.3), we have

$$\begin{aligned}
r_m^{\mathbf{k}(m)} &= \sum_{j \notin \mathbf{k}(m)} p_j \cdot r_m^{\mathbf{k}(m)} + \sum_{j \in \mathbf{k}(m)} p_j \cdot r_m^{\mathbf{k}(m)} \\
r_m^{\mathbf{k}(m)} &= r_m^{\mathbf{k}(m)} \left(\sum_{j \notin \mathbf{k}(m)} p_j + \sum_{j \in \mathbf{k}(m)} p_j \right) = r_m^{\mathbf{k}(m)} .
\end{aligned}$$

We now look at the second set of equations that must be verified. The LHS of the first equation below is the limiting probability of being in state $\omega_0^{\mathbf{k}(m-1)}$. The first summation on the RHS adds up all of the possible ways of entering this state from states of the form $\omega_i^{\mathbf{k}(m)}$. If we are in state $\omega_i^{\mathbf{k}(m)}$ and some $j \notin \mathbf{k}(m-1)$ receives the next unit of wealth, we end up in state $\omega_0^{\mathbf{k}(m-1)}$. The second summation on the RHS adds up all the possible ways of getting to $\omega_0^{\mathbf{k}(m-1)}$ from some other state $\omega_0^{\ell(m-1)}$. The only possible transitions are ones such that $\ell(m-1)$ and $\mathbf{k}(m-1)$ differ only in at most two processes (j and ℓ in the summation). From state $\omega_0^{\ell(m-1)}$, if ℓ wins the auction (with probability $\frac{1}{N-(m-1)}$) and j receives the next unit of wealth (with probability p_j), we end up in state $\omega_0^{\mathbf{k}(m-1)}$.

For every possible $\mathbf{k}(m-1)$,

$$
\begin{aligned}
\pi_0^{\mathbf{k}(m-1)} &= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} \left[\sum_{i\notin\mathbf{k}(m)} p_j \pi_i^{\mathbf{k}(m)} + p_j \sum_{\ell\in\mathbf{k}(m)} \frac{1}{N-(m-1)} \pi_0^{(\mathbf{k}(m)-\{\ell\})} \right] \\
(N-(m-1)) r_{m-1}^{\mathbf{k}(m-1)} &= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} p_j \left[\sum_{i\notin\mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} + \sum_{\ell\in\mathbf{k}(m)} r_{m-1}^{(\mathbf{k}(m)-\{\ell\})} \right] \\
&= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} p_j \left[\sum_{i\notin\mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} + \sum_{\ell\in\mathbf{k}(m)} p_\ell r_m^{\mathbf{k}(m)} \right] \\
&= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} p_j r_m^{\mathbf{k}(m)} \left[\sum_{i\notin\mathbf{k}(m)} p_i + \sum_{\ell\in\mathbf{k}(m)} p_\ell \right] \\
&= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} p_j r_m^{\mathbf{k}(m)} \\
&= \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} r_{m-1}^{\mathbf{k}(m-1)} \\
&= r_{m-1}^{\mathbf{k}(m-1)} \cdot \sum_{\mathbf{k}(m):\mathbf{k}(m)-\mathbf{k}(m-1)=\{j\}} 1 \\
&= (N-(m-1)) r_{m-1}^{\mathbf{k}(m-1)} .
\end{aligned}
$$

■

Lemma B.7 (Winning Probabilities) *Given* $\mathbf{w}[n]$ *is in a fair class* $C(m)$*, the probability that process* i *wins an auction at some arbitrary time* n *is equal to* p_i*.*

Proof: Given the limiting probabilities from Lemma 7.3, we simply need to make sure that the sum of the probabilities that process i wins the auction over all states is equal to p_i. We can break these states into two sets: ones in which i wins because it has the most wealth ($\omega_i^{\mathbf{k}(m)}$ for all $\mathbf{k}(m)$ such that $i \notin \mathbf{k}(m)$) and ones in which i wins a tie breaker ($\omega_0^{\mathbf{k}(m-1)}$ for all $\mathbf{k}(m-1)$ such that $i \notin \mathbf{k}(m-1)$). We simply need to verify the following equation

$$
\begin{aligned}
\sum_{\mathbf{k}(m):i\notin\mathbf{k}(m)} \pi_i^{\mathbf{k}(m)} + \frac{1}{N-(m-1)} \sum_{\mathbf{k}(m-1):i\notin\mathbf{k}(m-1)} \pi_0^{\mathbf{k}(m-1)} &= p_i \\
\sum_{\mathbf{k}(m):i\notin\mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} + \frac{1}{N-(m-1)} \sum_{\mathbf{k}(m-1):i\notin\mathbf{k}(m-1)} (N-(m-1)) r_{m-1}^{\mathbf{k}(m-1)} &= r_m p_i \\
\sum_{\mathbf{k}(m):i\notin\mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} + \sum_{\mathbf{k}(m):i\in\mathbf{k}(m)} p_i r_m^{\mathbf{k}(m)} &= r_m p_i \\
p_i \sum_{\mathbf{k}(m)} r_m^{\mathbf{k}(m)} &= r_m p_i
\end{aligned}
$$

The last equation follows by the definition of r_m. ■

6597-20