Stochastic Analysis of Maintenance and Routing Policies in Queueing Systems

Sherwin Doroudi

April 2016

Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mustafa Akan, Co-chair, Carnegie Mellon University
Mor Harchol-Balter, Co-chair, Carnegie Mellon University
Hayriye Ayhan, Georgia Tech
Mohammad Mousavi, University of Pittsburgh
Alan Scheller-Wolf, Carnegie Mellon University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was funded by NSF-CMMI-1334194 as well as a Computational Thinking grant from Microsoft Research.

Keywords: Queueing Theory, Stochastic Processes, Markov Chains, Computer Security, Malware, Call Centers, Game Theory, Routing, Server Farms

For my parents, Aladdin and Azar, and my wife, Shirin.

Abstract

This dissertation focuses on reexamining traditional management problems that emerge in service systems where customers or jobs queue for service.

In the first essay, we study a large class of quasi-birth-and-death Markov Chains that can model a variety of problems in computing, service, and manufacturing systems. These Markov chains consist of an infinite number of repeating levels and a finite number of phases, with transitions that are skip-free in level and unidirectional in phase. We present a procedure, which we call Clearing Analysis on Phases (CAP), for determining the limiting probabilities of such Markov chains exactly.

In the second essay, we apply the CAP method to evaluate maintenance policies in a setting where an online customer-facing service is vulnerable to persistent malware infections. These infections can cause performance degradation and facilitate data theft, both of which have monetary repercussions. Infections can go undetected and can only be removed by a time-consuming cleanup procedure, which takes the service offline and causes all existing jobs to be discarded without service. We ask how often and in response to what observable events cleanups should happen. In order to quantify the efficiency of various cleanup (maintenance) policies, we propose a revenue model which incorporates delay-based pricing and data theft costs. We model malware infections as a stochastic process that can evolve in stages. The CAP method enables the analysis and comparison of various cleanup policies.

Traditionally, routing in service systems attempts to minimize customer response times under the assumption that all servers work at a fixed rate. In reality, many service systems, such as call centers, are staffed by people who respond to workload incentives. In the third essay, we use a game-theoretic framework to model servers as agents who can choose how fast they work in response to the decisions of their coworkers (i.e., the other servers) and their manager (who sets the routing policy). We introduce a utility model where these strategic servers choose their service rate in order to maximize a tradeoff between an "effort cost" and a "value of idleness." Under this behavior, we find that some traditional policies such as routing to the fastest or slowest free server do not admit symmetric equilibria, while random routing or routing to the server that was most recently idle do admit such equilibria.

The fourth essay addresses routing (or dispatching) jobs in web server farms. Unlike human servers, computer servers are not strategic, but routing in web server farms requires sending jobs to a server immediately upon arrival. Once a job is sent to a server, the job is served according to the processor-sharing service discipline, and it cannot be moved to another server in the future. Our work models each arrival as having a randomly distributed value parameter. Given such value heterogeneity, the correct metric is not the minimization of response time, but rather, *value-weighted response time*. In this context, we ask "what is a good routing policy for minimizing the value-weighted response time metric?" and proceed to propose, analyze, and compare a number of new routing policies that are motivated by the goal of minimizing this metric.

Acknowledgments

First and foremost I cannot thank my dissertation advisors—Mustafa Akan and Mor Harchol-Balter—enough. Mustafa provided invaluable guidance that always kept me headed in the right direction. His deep insight into strategic models made it especially rewarding to have him as an advisor. Mor was not only an instrumental collaborator on Chapters 2, 3, and 5, but her mentorship helped me develop as a researcher, collaborator, writer, presenter, and teacher. Mor's infectious enthusiasm for research made her a joy to work with and her unending encouragement and support helped ensure that I kept things moving during my PhD. I am also indebted to Alan Scheller-Wolf, who I consider my "third advisor." I fondly recall our many meetings, where we would either tackle challenging theoretical problems together, or he would pass on sage advice on navigating life in academia.

Next, I would like to thank my two external committee members, Hayriye Ayhan and Mohammad Mousavi. They never hesitated in offering useful feedback on my work and both were extremely helpful with my job search.

Moreover, I would like to thank my collaborators for their essential contributions to the work presented in my dissertation: Thanassis Avgerinos, Brian Fralix, Raga Gopalakrishnan, Esa Hyttiä, Amy Ward, and Adam Wierman. I learned an immeasurable amount from these great researchers. I want to thank Adam in particular for introducing me to queueing theory and encouraging me to pursue a PhD.

Throughout my PhD, I greatly benefited from having regular opportunities to present (and receive feedback on) my ongoing research during SQUALL lunches. My thanks go out to the participants of these lunches, including Daniel Berger, Mohammad Delasay, Michele Dufalla, Anshul Gandhi, Kristy Gardner, Varun Gupta, Leela Nageswaran, Katsunobu Sasanuma, Siddharth Singh, Ying Xu, and Timothy Zhu. In particular, Anshul and Kristy were my collaborators on a variety of projects that were just as vital to my PhD experience as the chapters found in this dissertation.

I would like to express my gratitude for all the hard work put in by Lawrence Rapp in making my experience as a PhD student a smooth one. I am also grateful for the administrative support provided by Nancy Conway and Nicole Stenger.

The friendships I forged in the PhD program made my PhD much more manageable. Afshin Nikzad, Aabha Verma, and Xin Wang provided some much-needed camaraderie during the process of preparing for the Tepper qualifying exams (and in Xin's case, also during the job search), and I could always count on Majid Bazarbash for enlightening and inspiring conversation. Of course, the friends I made before coming into the program—Nima Beheshti, Daniel Cullina, and Khalil Mohseni—supported me just as much as the new ones I found in Pittsburgh.

I would not be where I am now or even *who I am now* if not for my infinitely supportive family: my parents Aladdin and Azar and my brother Shayan. It is wonderful knowing that I also have the support of a second family: my father-in-law, Parviz Razavian, and my brothers-in-law Pooya and Payam. Finally, I thank my lovely and loving wife, Shirin, who supported me in more ways than I can count.

Contents

1 Introductory Remarks **1**

2 Clearing Analysis on Phases **4**

2.1 Introduction . . . 4
2.1.1 The matrix-geometric approach . . . 5
2.1.2 Our approach: Clearing Analysis on Phases (CAP) . . . 9
2.1.3 Recursive Renewal Reward, ETAQA, and other techniques . . . 10
2.2 Examples of class $\mathbb{M}$ Markov chains . . . 11
2.2.1 Single server in different power states . . . 11
2.2.2 Server fatigue . . . 12
2.2.3 Server with virus infections . . . 13
2.3 Results . . . 14
2.3.1 A key idea . . . 14
2.3.2 Preliminaries . . . 15
2.3.3 The case where all nonzero bases are distinct . . . 17
2.3.4 The case where all bases agree . . . 21
2.3.5 The case where all bases except r_M agree . . . 23
2.4 Analysis of the M/M/1/clearing model . . . 25
2.4.1 Preliminary results on clearing models . . . 26
2.4.2 Applying clearing model analysis toward proving Theorem 2 . . . 29
2.5 Extending the scope of the CAP Method . . . 32
2.5.1 Chains with “catastrophes” . . . 32
2.5.2 Skipping levels when transitioning between phases . . . 32
2.5.3 Chains with an infinite number of phases . . . 32
2.6 Conclusion . . . 33

3 The Malware Cleanup Problem **35**

3.1 Introduction . . . 35
3.2 Literature Review . . . 37
3.2.1 Machine interference problems . . . 37
3.2.2 Condition-based maintenance . . . 38
3.2.3 Methods for solving quasi-birth-death process Markov chains . . . 39
3.3 The Case of Visible Malware . . . 40
3.3.1 Visible Malware Model . . . 40

3.3.2 Visible Malware Analysis . . . 43
3.3.3 Visible Malware Results . . . 48
3.4 The Case of Hidden Malware . . . 51
3.4.1 Hidden Malware Model . . . 52
3.4.2 Hidden Malware Analysis . . . 57
3.4.3 Hidden Malware Results . . . 62
3.5 Approximate Analysis . . . 69
3.6 Conclusion . . . 73

4 Routing when Servers are Strategic 74
4.1 Introduction . . . 74
4.1.1 Contributions of This Chapter . . . 75
4.1.2 Related Work . . . 76
4.2 A Model for Strategic Servers . . . 77
4.3 The M/M/N Queue with Strategic Servers . . . 79
4.4 Routing to Strategic Servers . . . 81
4.4.1 Idle-Time-Order-Based Policies . . . 81
4.4.2 Rate-Based Policies . . . 84
4.5 Conclusion . . . 92

5 Routing with Heterogeneous Job Values 94
5.1 Introduction . . . 94
5.2 Prior work on value-driven dispatching . . . 97
5.3 Model for PS server system . . . 97
5.4 Description of simple dispatching policies . . . 98
5.4.1 Random dispatching (RND) . . . 98
5.4.2 Join-the-Shortest-Queue dispatching (JSQ) . . . 98
5.4.3 Value-Interval-Task-Assignment (VITA) . . . 99
5.4.4 C-MU . . . 99
5.4.5 Length-and-Value-Aware (LAVA) . . . 100
5.5 Simulation results and intuitions . . . 101
5.6 Analytic results . . . 104
5.6.1 RND and JSQ under high load . . . 104
5.6.2 Stability and instability . . . 105
5.6.3 Results under sharply bimodal distributions . . . 106
5.7 A (sometimes) far better policy: Gated VITA (G-VITA) . . . 110
5.7.1 G-VITA . . . 111
5.7.2 G-VITA simulations . . . 113
5.8 More complex policies via the First Policy Iteration (FPI) . . . 113
5.8.1 FPI policies . . . 115
5.8.2 Enhancing FPI policies using discounting . . . 116
5.8.3 FPI simulations . . . 116
5.9 Conclusion . . . 117

6 Concluding Remarks 118

A Supplement to Chapter 2 120
A.1 An alternative interpretation of the Laplace transform 120
A.2 Complete proof of Theorem 2.3 120
A.3 Negative binomial lemmas 125

B Supplement to Chapter 5 129
B.1 Proof of Proposition 5.4 129
B.2 Proof of Lemma 5.10 131
B.3 Proof of Lemma 5.14. 135
B.4 Proof of Proposition 5.16 136
B.5 Proof of Proposition 5.17 137

Bibliography 139

Chapter 1

Introductory Remarks

Queueing theory, which studies waiting times and queue lengths under stochastic uncertainty, is an area of applied probability which has been studied for just over a hundred years.[1] Despite the age of this discipline, queueing-theoretic modeling and analysis techniques continue to prove useful as new problems where waiting times are a concern are constantly emerging. These techniques are applied to problems emerging from service systems as diverse as manufacturing systems, transportation systems, two-sided markets, healthcare services, web server farms, supercomputing systems, cloud computing platforms, and online matchmaking services, among others. This dissertation in particular considers problems arising from computer security, and from the management of call centers and web server farms.

Contributions to the analysis of queueing models are typically either *descriptive* or *prescriptive* in nature. Descriptive work in this area typically seeks to answer *performance analysis* questions about given queueing systems such as the following:

- What is the average waiting time?
- What percentage of customers will experience a waiting time of less than 10 seconds?
- What is the distribution of the queue length?
- What is the average number of servers that will be utilized at any given time?

Prescriptive work, on the other hand, seeks to answer questions of how queueing systems should be designed to achieve particular goals, which typically revolve around reducing waiting times (or waiting costs) subject to feasibility constraints. Examples of operational levers that can be adjusted in the design of queueing systems include the following:

- Staffing: how many servers should be employed?
- Capacity: how fast should the server(s) operate?
- Scheduling: which customer or job should be processed first?
- Admissions: which customers should be served at all?
- Routing: which server (or queue) should an incoming customer or job be sent to?
- Redundancy: how *many* queues should a job be sent to?

[1]An unpublished manuscript dealing with queueing models due to Tore Olaus Engset (1865–1943) dates back to 1915 [120].

- Availability: when should the service be available?
- Pricing: how much should be charged for the service?

In particular, there has been a growing interest in prescriptive problems such as these in settings that allow for agents (customers, service providers, servers, etc.) that behave strategically.

This dissertation comprises four essays (Chapters 2–5), each of which deals with the topic of queueing systems. Chapter 2 presents a methodological performance to descriptive queueing analysis, while Chapter 3 applies this methodology to address a prescriptive queueing problem with a focus on the tradeoff between *maintenance actions* (which we call cleanup procedures) and *service availability* with applications to computer security. Meanwhile, Chapters and 4 and 5 address prescriptive routing problems in the very different contexts of call centers and web server farms, respectively. The remainder of this introduction presents a more detailed overview of the contents of these chapters.

Chapter 2 introduces a class of infinite Markov chains that are useful in modeling a variety of Markovian queueing systems where the system parameters can change over time in a Markov modulated fashion. An example of a queueing system that can be modeled by such chains is a standard Markovian queue where a human server grow fatigued (or a computer servers declines in efficiency) over time in a random Markovian fashion, which effectively reduces the rate at which the server can process customers. The primary contribution of Chapter 2 is the introduction of the *Clearing Analysis on Phases* method, which can be used to determine the limiting probability distribution of such Markov chains in *exact closed form*, whereas existing methods for solving such chains largely resort to numerical solutions. The contents of this chapter originally appeared in the working paper [43], which was written by the author in collaboration with Brian Fralix (Clemson University) and Mor Harchol-Balter (CMU).

Motivated by omnipresent cybersecurity concerns, Chapter 3 addresses the question of when cleanup procedures should be undertaken to repair an online service system that has potentially been compromised by performance degrading cyber attacks such as malware. This online service system is one in which customers queue for service, and we model the system as becoming compromised by progressively more serious attacks in a (possibly unobservable) Markov modulated fashion. We consider, analyze, and compare a variety of cleanup policies that will take the system offline in response to observable criteria, and return the system online after all persistent effects of the attack have been removed. The comparisons of the policies are made with respect to a revenue function that simultaneously captures the desire to maintain high availability, low waiting times, and minimal security risks associated with being compromised. In particular, the system is modeled in such a way that the analysis of each cleanup policy requires the analysis of a Markov chain is amenable to the Clearing Analysis on Phases method introduced in Chapter 2. This cleanup problem can be viewed as a condition-based maintenance problem with the added complexity of having to track the (potentially unbounded) queue length. Therefore, Chapter 3 (together with the analytic technique introduced in the previous chapter) makes up the "maintenance" portion of the dissertation alluded to in the title. As of the time of the writing of this dissertation, the content presented in Chapter 3, which was written by the author in collaboration with Thanassis Avgerinos (co-Founder of the security company ForAllSecure, Inc.) and Mor Harchol-Balter (CMU), is in preparation for submission to a journal.

Chapter 4 features a reexamination of routing policies implemented in call centers staffed by human servers; it is the first of two chapters making up the “routing” portion of this dissertation alluded to in the title and the only chapter in the dissertation that considers strategic models. Rather than uphold the traditional assumption that all servers in a server farm operate at a fixed service rate, we consider human servers that are free to choose their own service rates. Not only do servers choose their own strategic rates in order to strike a balance between idle times and the effort associated with working at faster speeds, but they make their choices *strategically*, taking the choices of their co-workers into consideration. The strategic behavior of the servers causes undesirable properties to emerge from traditional routing policies such as the Fastest Server First policy, and highlights the need for robust routing policies that perform well even in the presence of strategic behavior. The content contained in Chapter 4 is a subset of that presented in [61], which was written by the author in collaboration Ragavendran Gopalakrishnan (Xerox Research Centre India), Amy R. Ward (USC), and Adam Wierman (Caltech). While Chapter 4 is concerned only with *routing* under the presence of strategic servers, [61] also addresses the question of *staffing* call centers when servers are strategic.

The final essay of the dissertation, Chapter 5, considers a very different routing problem in the context of web server farms composed of multiple servers, with each server serving all of the jobs in their queues according to the processor sharing service discipline. In this setting, jobs are dispatched to servers immediately, rather than being held in a central queue. Our work departs from the preexisting work on routing in web server farms by considering heterogeneity in jobs values. That is, the goal is not to simply minimize mean response times, but rather, *value-weighted response times*, a metric which takes the relative importance of jobs into account. Essentially, the goal of Chapter 5 is to identify heuristic routing policies that perform well with respect to this metric, and to compare such policies across a variety of settings using both analytic techniques, and simulations. Developing strong routing policies in such a setting is challenging as one has no control over the scheduling that occurs at the servers, so one must leverage routing to achieve what would normally be easily accomplished via priority scheduling (e.g. via the famous $c\mu$ rule). The contents of this chapter have been published in [42], which was written by the author in collaboration with Esa Hyttiä (Aalto University) and Mor Harchol-Balter (CMU).

We present concluding remark in Chapter 6, highlighting some directions for future work in all of the areas explored in this dissertation.

Chapter 2

Clearing Analysis on Phases

2.1 Introduction

This chapter studies the stationary distribution of Class $\mathbb{M}$ Markov chains, which are continuous time Markov chains (CTMCs)[1] having the following properties (see Fig. 2.1 and Fig. 2.2):

- The Markov chain has a state space, $\mathcal{E}$, that can be decomposed as $\mathcal{E} = \mathcal{R} \cup \mathcal{N}$, where $\mathcal{R}$ represents the infinite *repeating portion* of the chain, and $\mathcal{N}$ represents the finite *nonrepeating portion* of the chain.[2]
- The repeating portion is given by

$$\mathcal{R} \equiv \{(m, j) : 0 \le m \le M, j \ge j_0\}$$

  where both M and j_0 are finite nonnegative integers. We refer to a state $(m, j) \in \mathcal{R}$ as currently being in phase m and level j. For each $j \ge j_0$, level j is given by

$$L_j \equiv \{(0, j), (1, j), \ldots, (M, j)\}.$$

  Throughout this chapter, we index phases by i, k, m, and u, and we index levels by j and ℓ.
- Transitions between a pair of states in $\mathcal{N}$ may exist with any rate.
- Transitions from states in $\mathcal{N}$ to states in $\mathcal{R}$ may only go into states in L_{j_0}, but may exist with any rate.
- Transitions from states in $\mathcal{R}$ to states in $\mathcal{N}$ may only come from states in L_{j_0}, but may exist with any rate.
- Transitions between two states in $\mathcal{R}$ that are both in the same phase, m, (e.g., the "horizontal" transitions in Fig. 2.1 and Fig. 2.2) are described as follows, with $q(x, y)$ denoting the transition rate from state x to state y:

$$\begin{aligned} \lambda_m &\equiv q((m, j), (m, j+1)) && (0 \le m \le M,\ \ j \ge j_0) \\ \mu_m &\equiv q((m, j), (m, j-1)) && (0 \le m \le M,\ \ j \ge j_0 + 1). \end{aligned}$$

[1]The methodology presented in this chapter can easily be modified to apply to discrete time Markov chains.
[2]We note that this partition is not unique.

- We express transition rates between two states in $\mathcal{R}$, which transition out of a state in phase m to a state in another phase (e.g., the "vertical" transitions in Fig. 2.1 and the "vertical" and "diagonal" transitions in Fig. 2.2) using the notation $\alpha_m\langle\Delta_1;\Delta_2\rangle$, where $\Delta_1 \geq 1$ is the increase in phase from m to $m+\Delta_1$ (i.e., the "vertical" shift) and $\Delta_2 \in \{-1,0,1\}$ is the change in level, if any, from j to $j+\Delta_2$ (i.e., the "horizontal" shift). Note that $\Delta_1 \geq 1$ indicates that only transitions to higher-numbered phases are allowed, while $\Delta_2 \in \{-1,0,1\}$ indicates that each transition may change the level by at most 1 in either direction. More specifically, these transitions are described as follows:

  $$\begin{aligned} \alpha_m\langle i-m;-1\rangle &\equiv q((m,j),(i,j-1)) && (0 \leq m < i \leq M,\ \ j \geq j_0+1) \\ \alpha_m\langle i-m;0\rangle &\equiv q((m,j),(i,j)) && (0 \leq m < i \leq M,\ \ j \geq j_0) \\ \alpha_m\langle i-m;1\rangle &\equiv q((m,j),(i,j+1)) && (0 \leq m < i \leq M,\ \ j \geq j_0). \end{aligned}$$

  We will also use the shorthand notation

  $$\alpha_m = \sum_{i=m+1}^{M} \left(\alpha_m\langle i-m;-1\rangle + \alpha_m\langle i-m;0\rangle + \alpha_m\langle i-m;1\rangle\right)$$

  throughout the chapter to represent the *total outgoing transition rate to other phases* from states in phase m with level $j \geq j_0+1$.
- The Markov chain must be ergodic.

Markov chains in class $\mathbb{M}$ are examples of quasi-birth-death processes (QBDs), with increments and decrements in level corresponding to "births" and "deaths," respectively. We say that transitions in class $\mathbb{M}$ chains are *skip-free in level*, in that the chain does not allow for the level to increase or decrease by more than 1 in a single transition. We also say that transitions in class $\mathbb{M}$ chains are *unidirectional in phase*, in that transitions may only be made to states having either the same phase or a higher phase in the repeating portion. Note however that phases may be skipped: for example, transitions from a state in phase 2 to a state in phase 5 may exist with nonzero rate.

Many common queueing systems arising in computing, service, and manufacturing systems can be modeled with CTMCs from class $\mathbb{M}$. For such systems, one often needs to track both the number of jobs in the system and the state of the server(s), where each server may be in one of several states, e.g., working, fatigued, on vacation, etc. When modeling a system with a class $\mathbb{M}$ Markov chain, we often use the level, j, of a state (m,j) to track the number of jobs in the system, and we use the phase, m, to track the state of the server(s) and/or the arrival process. For example, a change in phase could correspond to (i) a policy modification that results in admitting more customers, as captured by an increase in "arrival rate" from λ_m to λ_i, where $\lambda_i > \lambda_m$ or (ii) a change in the state of the servers leading to an increase or decrease in the service rate from μ_m to μ_i. A few examples of systems that can be modeled by Class $\mathbb{M}$ Markov chains are presented in Section 5.5.

2.1.1 The matrix-geometric approach

One way of studying the stationary distribution, π, of a class $\mathbb{M}$ Markov chain is to observe that it exhibits a matrix-geometric structure on $\mathcal{R}$. More specifically, if we let $\vec{\pi}_j$ represent the limiting

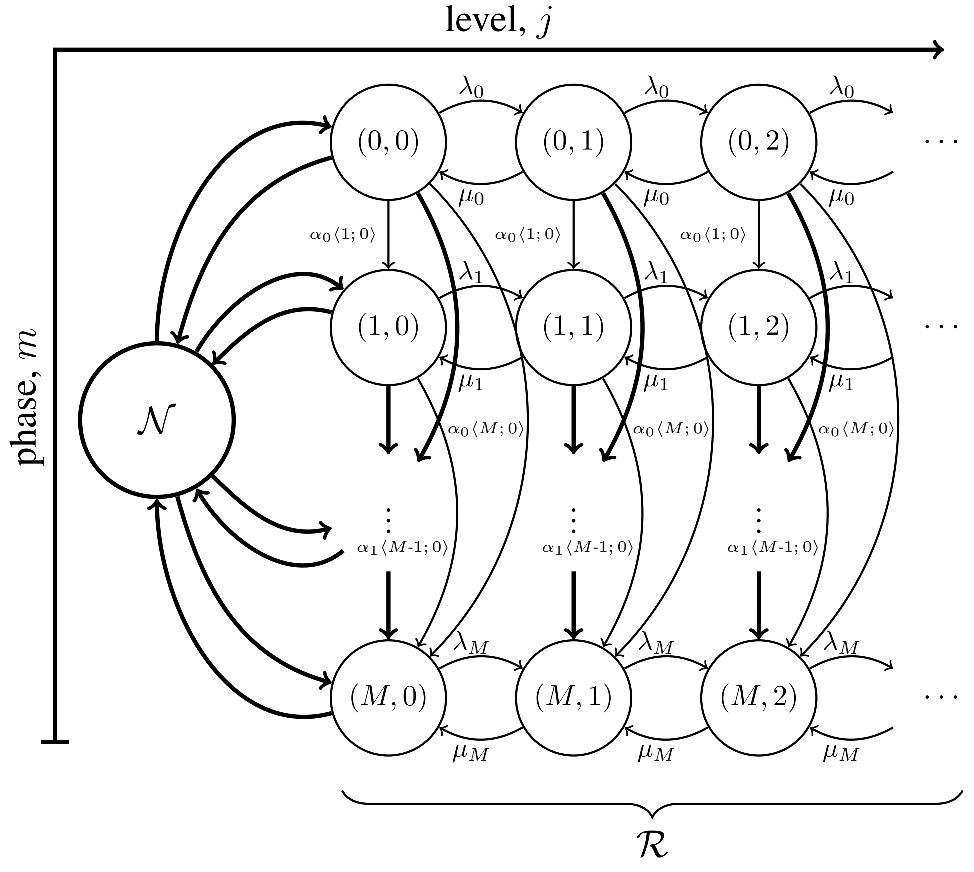


Figure 2.1: The structure of class $\mathbb{M}$ Markov chains. In this case $j_0 = 0$ and, for simplicity, $\alpha_m\langle i - m; \pm 1\rangle = 0$. The chain is made up of a non-repeating portion, $\mathcal{N}$ (shown here as an aggregation of states), and a repeating portion, $\mathcal{R}$. Within $\mathcal{R}$, each phase, m, corresponds to a "row" of states, and each level, j, corresponds to a "column" of states. Transitions between levels in each phase of the repeating portion, $\mathcal{R}$, are *skip-free*: all such transitions move only *one step* to the "left" or "right." Transitions between phases in each level of $\mathcal{R}$ are *unidirectional*: all such transitions move "downward." The thicker arrows denote *sets* of transitions (transitions rates for these sets are omitted from the figure).

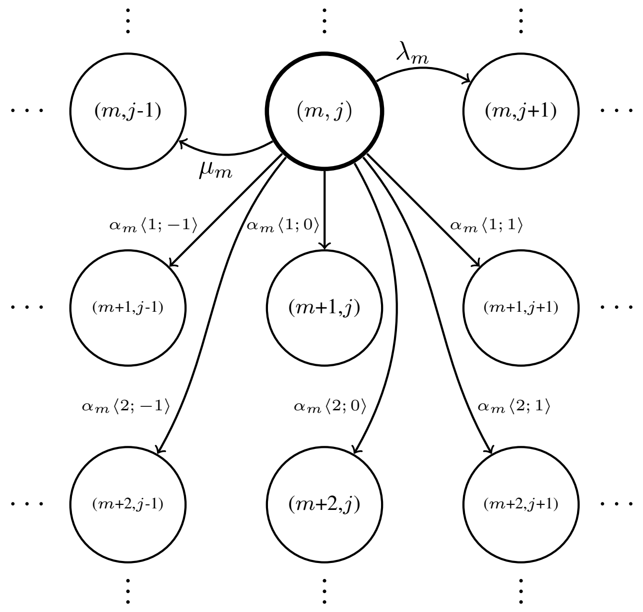

λ_m
μ_m
$(m,j\text{-}1)$
(m,j)
$(m,j\text{+}1)$
$\alpha_m\langle 1;-1\rangle$
$\alpha_m\langle 1;0\rangle$
$\alpha_m\langle 1;1\rangle$
$(m\text{+}1,j\text{-}1)$
$(m\text{+}1,j)$
$(m\text{+}1,j\text{+}1)$
$\alpha_m\langle 2;-1\rangle$
$\alpha_m\langle 2;0\rangle$
$\alpha_m\langle 2;1\rangle$
$(m\text{+}2,j\text{-}1)$
$(m\text{+}2,j)$
$(m\text{+}2,j\text{+}1)$

Figure 2.2: Another more detailed look at the transition structure of class $\mathbb{M}$ Markov chains. For simplicity, only the set of transitions that are possible *from* state (m, j) (where $j \geq j_0 + 1$) to states in phases m, $m + 1$, and $m + 2$ are shown. Note that all transitions from (m, j) are either to the left, to the right, or downward. Furthermore, all transitions can decrease or increase the level by at most one.

probability of the states in L_j, that is, $\vec{\pi}_j \equiv (\pi_{(0,j)}, \pi_{(1,j)}, \ldots, \pi_{(M,j)})$, then for $j \geq j_0$

$$\vec{\pi}_{j+1} = \vec{\pi}_j \mathbf{R}$$

where $\mathbf{R} \in \mathbb{R}^{(M+1)\times(M+1)}$ is referred to as the *rate matrix* associated with the chain. If we let the *sojourn rate* of state x be defined by

$$\nu_x = \sum_{y \neq x} q(x, y),$$

then we can describe the elements of $\mathbf{R}$ probabilistically as follows: the element, $R_{i,m}$, in row i, column m of $\mathbf{R}$ can be interpreted as $\nu_{(i,j)}$ times the expected cumulative amount of time the chain spends in state $(m, j+1)$ before making a transition into a level strictly below $j+1$, given the chain starts in state (i, j). For most QBDs, one cannot derive an exact expression for each element of $\mathbf{R}$, but there are many ways to compute an approximation of **R** numerically: see for example [22, 94]. Readers interested in further details should consult the matrix-analytic texts of Neuts [105], Latouche and Ramaswami [93], and He [75]. Queueing textbooks of a broader scope that also discuss matrix-analytic methods include Asmussen [13] and Harchol-Balter [70]. Once $\mathbf{R}$—or good approximations for $\mathbf{R}$—have been found, then $\vec{\pi}_j = \vec{\pi}_{j_0} \mathbf{R}^{j-j_0}$ for $j \geq j_0$, and so all remaining limiting probabilities, π_x, for $x \in \mathcal{N}$, can be found using the balance equations and the normalization constraint.

There are many examples of QBDs with a rate matrix, **R**, that can be computed exactly through a finite number of operations. One class of QBDs having a closed-form rate matrix is presented in Ramaswami and Latouche [109], with an extension to Markov chains of $GI/M/1$-type given in Liu and Zhao [98]. Other classes of QBDs having explicitly computable rate matrices are considered in the work of van Leeuwaarden and Winands [125] and van Leeuwaarden et al. [126], with both of these studies being much closer to our work, since most (but not all) of the types of Markov chains studied in [125], and all of the chains discussed in [126] belong to class $\mathbb{M}$. In [125, 126] combinatorial techniques are used to derive expressions for each element of **R** that can be computed exactly after a finite number of operations, but their methods are not directly applicable to all class $\mathbb{M}$ Markov chains as they further assume that λ_m and μ_m are the same for $0 \leq m \leq M-1$, and they also assume that for each $0 \leq m \leq M-1$, any transitions leaving phase m must next enter phase $m+1$ (i.e., they assume *phase* transitions are *skip-free*—in addition to being unidirectional—within the repeating portion of the chain).

Even closer to our work is the work of Van Houdt and van Leeuwaarden [124], which presents an approach for the explicit calculation of the rate matrix for a broad class of QBDs including those in class $\mathbb{M}$. This approach involves solving higher order (scalar) polynomial equations, the solutions to which are expressed as infinite sums, which typically cannot be computed in closed-form. However, [124] also gives an approach for calculating closed-form rate matrices for a class of Markov chains called tree-like QBDs. Tree-like QBDs neither contain nor are contained by class $\mathbb{M}$, although there is significant overlap between the two. Transitions between phases (within a level) in tree-like QBDs form a *directed tree*, while transitions between phases in class $\mathbb{M}$ Markov chains form a *directed acyclic graph.* Specifically, unlike class $\mathbb{M}$ chains, tree-like QBDs *do not* allow for a pair of phases $i \neq k$ to both have transitions to the same phase m (i.e., tree-like QBDs do not allow for both $\alpha_i\langle m-i; \Delta\rangle > 0$ and $\alpha_k\langle m-k; \Delta'\rangle > 0$ when $i \neq k$ and $\Delta, \Delta' \in \{-1, 0, 1\}$).

2.1.2 Our approach: Clearing Analysis on Phases (CAP)

In this study we introduce the CAP (Clearing Analysis of Phases) method for evaluating the stationary distribution of class $\mathbb{M}$ Markov chains. This method proceeds iteratively among the *phases*, by first expressing all $\pi_{(0,j)}$ probabilities, for $j \geq j_0$, in terms of π_x probabilities for $x \in \mathcal{N}$. Once each element $\pi_{(m,j)}$ for a fixed phase m, $j \geq j_0$ has been expressed in terms of $\{\pi_x\}_{x\in\mathcal{N}}$, we then do the same for all $\pi_{(m+1,j)}$ terms. After each $\pi_{(M,j)}$ expression has been determined, we use the balance equations and normalization constraint to solve for the remaining $\{\pi_x\}_{x\in\mathcal{N}}$ probabilities. CAP takes its name from the fact that, between two phase transitions, class $\mathbb{M}$ Markov chains behave like an M/M/1/clearing model, that is, each phase is likened to a birth-death process that experiences "clearing" or catastrophic events in accordance to an independent Poisson process. In our model, these "clearings" corresponds to a change in phase.

One major advantage of the CAP method is that it avoids the task of finding the complete rate matrix, **R**, entirely, while yielding expressions for $\pi_{(m,j)}$ that only involve raising $M+1$ *scalars* to higher powers. There exists one such scalar, r_m, for each phase, $m \in \{0, 1, \ldots, M\}$. These scalars, referred to throughout as *base terms*, are actually the diagonal elements of the rate matrix, **R**, i.e.,

$$r_m = R_{m,m}, \qquad (0 \leq m \leq M)$$

and the transition structure of class $\mathbb{M}$ Markov chains makes these elements much easier to compute than any of the other nonzero elements of **R**. Furthermore, the structure of $\pi_{(m,j)}$ depends entirely on the number of base terms that agree with one another. For example, when all nonzero base terms are distinct, one can show that

$$\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0}, \tag{2.1}$$

for $0 \leq m \leq M$, where the $\{c_{m,k}\}_{0\leq k\leq m\leq M}$ values are constants that do not vary with j, and can be computed exactly by solving a linear system of $O(M^2 + |\mathcal{N}|)$ linear equations.

In the case where all base terms agree, we instead find that

$$\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} \binom{j-(j_0+1)+k}{k} r_0^{j-j_0}, \tag{2.2}$$

where again, the $c_{m,k}$ terms can be computed by solving a linear system.

In retrospect, it is of no surprise that $\pi_{(m,j)}$ can be expressed as a linear combination of scalars, each raised to the power of $j - j_0$, as in Equations (2.1) and (2.2): $\mathbf{R}$ must be upper-triangular for class $\mathbb{M}$ chains. This follows by observing that $R_{i,m}$ is $\nu_{(i,j_0)}$ times the expected cumulative amount of time spent in state (m, j_0+1) before returning to L_{j_0}, given initial state (i, j_0), and this value is 0 when $i > m$. Since **R** is upper-triangular, its eigenvalues are simply its diagonal elements, which are also the diagonal elements of the Jordan normal form of $\mathbf{R}$—see e.g., Chapter 3 of Horn and Johnson [77]—from which we know that $\pi_{(m,j)}$ can be expressed as a linear combination of scalars, each raised to the power of $j - j_0$. Although in theory, our solution form could be recovered by first computing $\mathbf{R}$ and then numerically determining $\mathbf{R}$ in Jordan normal form, such a procedure is often inadvisable. The structure of the Jordan normal form of a matrix can be extremely sensitive to small changes in one or more of its elements, particularly when some of its eigenvalues have algebraic multiplicity larger than one, as is the case for all of the models discussed in [125, 126]. Fortunately, the CAP method can handle these cases as well with little additional difficulty.

The statement and proofs of this chapter's main results are presented in Section 2.3. This proof relies on some results regarding M/M/1/clearing models; the proofs of these results are deferred to Section 2.4. In Section 2.5 we briefly touch upon how the CAP method may be applied to chains beyond those in class $\mathbb{M}$.

2.1.3 Recursive Renewal Reward, ETAQA, and other techniques

We briefly review existing techniques for solving QBDs beyond the matrix-geometric approach and comment on their connection to the CAP method.

Gandhi et al. [53, 54] use renewal theory to determine exact mean values and z-transforms of various metrics for a subclass of chains in $\mathbb{M}$ via the Recursive Renewal Reward (RRR) method. The class of chains they study do not allow for "diagonal" transitions (i.e., $\alpha_m\langle i-m;\pm 1\rangle = 0$). Unlike our method, RRR *cannot* be used to determine a formula for a chain's limiting probability distribution in finitely many operations. While there is overlapping intuition and flavor between CAP and RRR—both methods make use of renewal reward theory—CAP is *not* an extension of RRR and does not rely on any of the results from [53, 54].

The Efficient Technique for the Analysis of QBD-processes by Aggregation (ETAQA), first proposed by Ciardo and Simirni [33], combines ideas from matrix analytic and state aggregation approaches in order to compute various exact values (e.g., mean queue length) for a wide class of Markov chains. By design, ETAQA yields the limiting probability of the states in the non-repeating portion, $\mathcal{N}$, along with the limiting probabilities of the states in the first level (or first few levels) of the repeating portion, $\mathcal{R}$. The limiting probabilities of the remaining states (i.e., higher level states) are aggregated, which allows for the speedy computation of exact mean values and higher moments of various metrics of interest. In particular, ETAQA involves solving a system of only $O(|\mathcal{N}|+M)$ linear equations. Although originally applicable to a narrow class of chains (see [33, 35] for details), ETAQA can be generalized so as to be applicable to M/G/1-type, GI/M/1-type, and QBD Markov chains, including those in class $\mathbb{M}$ (see the work of Riska and Smirni [112, 113]). Stathopoulos et al. [119] show that ETAQA is also well suited for numerical computations; ETAQA can be adapted to avoid the numerical problems alluded to in Section 2.1.1. Unlike the CAP method, ETAQA (like RRR) *cannot* be used to determine a formula for a chain's limiting probability distribution (across all states) in finitely many operations.

For certain class $\mathbb{M}$ Markov chains, one can also manipulate generating functions to derive limiting probabilities, such as in the work of Levy and Yechiali [95] and the work of Phung-Duc [106], where this type of approach is used to solve multi-server vacation and setup models, respectively. This approach is covered in greater generality in a technical report by Adan and Resing [5]. We note that although generating function approaches can yield solutions of a form similar to those found using the CAP method, the two approaches differ in methodology.

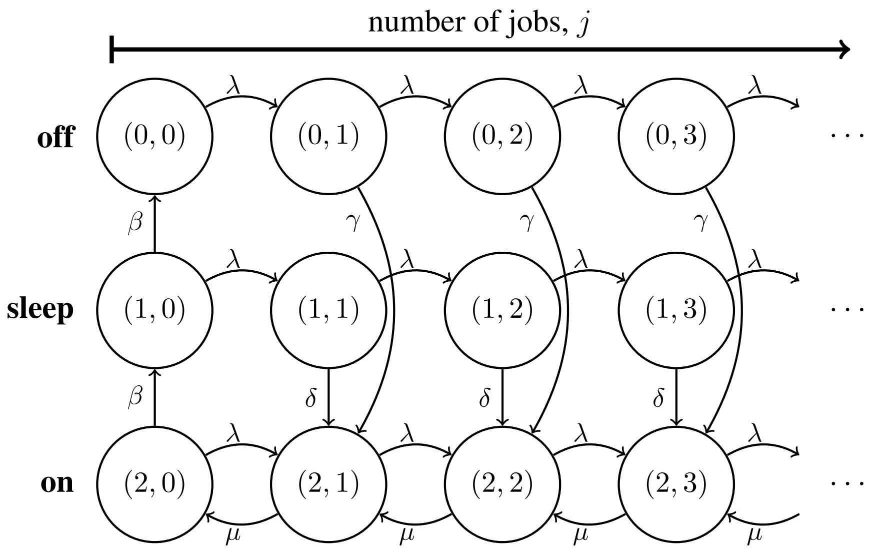


Figure 2.3: The Markov chain for a single server in different power states. State (m, j) indicates that the server is in state m (0=off, 1=sleep, 2=on) with j jobs in the system.

2.2 Examples of class $\mathbb{M}$ Markov chains

In this section we provide several examples of queueing systems which can be modeled by class $\mathbb{M}$ Markov chains. In each example we will use the phase, $m \in \{0, 1, \ldots, M\}$, to track the "state" of the server(s) and/or the arrival process, and the level, j, to track the number of jobs in the system. Of course, there are many systems beyond those covered in this section that can be modeled by class $\mathbb{M}$ Markov chains. For example, class $\mathbb{M}$ chains were recently used to model medical service systems in [40], [31], and [116]. In particular, Chapter 3 of this dissertation focuses on employing class $\mathbb{M}$ chains to model computer systems that are susceptible to performance degrading cyber attacks under a variety of cleanup policies.

2.2.1 Single server in different power states

Consider a computer server that can be in one of three different power states: on, off, or sleep. In the **on** state, the server is fully powered and jobs are processed at rate μ. In the **off** state, the server consumes no power, but jobs cannot be processed. When the server is idle, it is desirable to switch to the off state in order to conserve power, however there is a long setup time, distributed $\text{Exponential}(\gamma)$, needed to turn the server back on when work arrives. Because of this setup time, it is common to switch to a state called the **sleep** state, where the server consumes less power than the **on** state, but where there is a shorter setup time, distributed $\text{Exponential}(\delta)$, for turning the server on. It is also common to purposefully impose a waiting period, distributed $\text{Exponential}(\beta)$, in powering down a server (from on to sleep, and again from sleep to off) once it is idle, which is useful just in case new jobs arrive soon after the server becomes idle. See [52] for more details.

Fig. 2.3 shows a Markov chain representing this setting. This is a class $\mathbb{M}$ chain with $M+1 =$

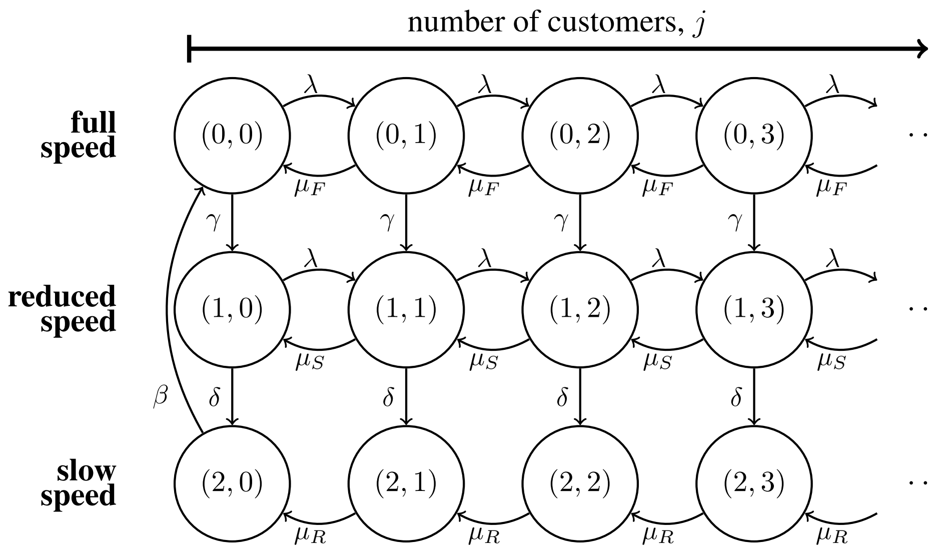


Figure 2.4: The Markov chain for a server susceptible to fatigue. State (m, j) indicates server state m (0=full speed, 1=reduced speed, 2=slow speed) with j customers in the system.

3 phases: **off** ($m = 0$), **sleep** ($m = 1$), and **on** ($m = 2$). For this chain, $j_0 = 1$ and the non-repeating portion of the state space is $\mathcal{N} = \{(0,0), (1,0), (2,0)\}$, while $\lambda_0 = \lambda_1 = \lambda_2 = \lambda$, $\mu_0 = \mu_1 = 0$, $\mu_2 = \mu$, $\alpha_0\langle 2; 0\rangle = \gamma$, and $\alpha_1\langle 1; 0\rangle = \delta > \gamma$ (all other $\alpha_m\langle m - i; \Delta\rangle$ transition rates are zero).

The system becomes much more interesting when there are multiple servers, where each can be in one of the above 3 states. In the case of 2 servers, there will be 6 phases, corresponding to: (off,off), (off,sleep), (off,on), (sleep,sleep), (sleep,on), (on,on). Note than in this case, phase transitions will include transitions with rates 2γ, $\gamma + \delta$, and 2δ, as both servers may be attempting to turn on at the same time. In general, a system with a servers and b server states will have $\binom{a+b-1}{a}$ phases.

2.2.2 Server fatigue

Consider a human server who starts her shift full of energy and works quickly (at rate μ_F). As time passes and fatigue sets in, she gets slower and slower (first she slows down to a reduced rate μ_R and eventually to a very slow rate μ_S, where $\mu_S < \mu_R < \mu_F$). At some point it makes sense to replace her with a fresh human server. However, before we can do that, she needs to finish serving her queue of existing customers, while no longer accepting further arrivals. We assume that the time it takes for the new replacement to start working is distributed $\text{Exponential}(\beta)$.

Fig. 2.4 shows a Markov chain representing this setting. This is a class $\mathbb{M}$ chain with $M+1 = 3$ phases: **full speed** ($m = 0$), **reduced speed** ($m = 1$), and **slow speed** ($m = 2$). For this chain, $j_0 = 1$ and the non-repeating portion of the state space is $\mathcal{N} = \{(0,0), (1,0), (2,0)\}$, while $\lambda_0 = \lambda_1 = \lambda$, $\lambda_2 = 0$, $\mu_0 = \mu_F$, $\mu_1 = \mu_R < \mu_F$, $\mu_2 = \mu_S < \mu_R$, $\alpha_0\langle 1; 0\rangle = \gamma$ and $\alpha_1\langle 1; 0\rangle = \delta$ (all other $\alpha_m\langle m - i; \Delta\rangle$ transition rates are zero).

Again, the system becomes much more interesting when there are multiple servers, where

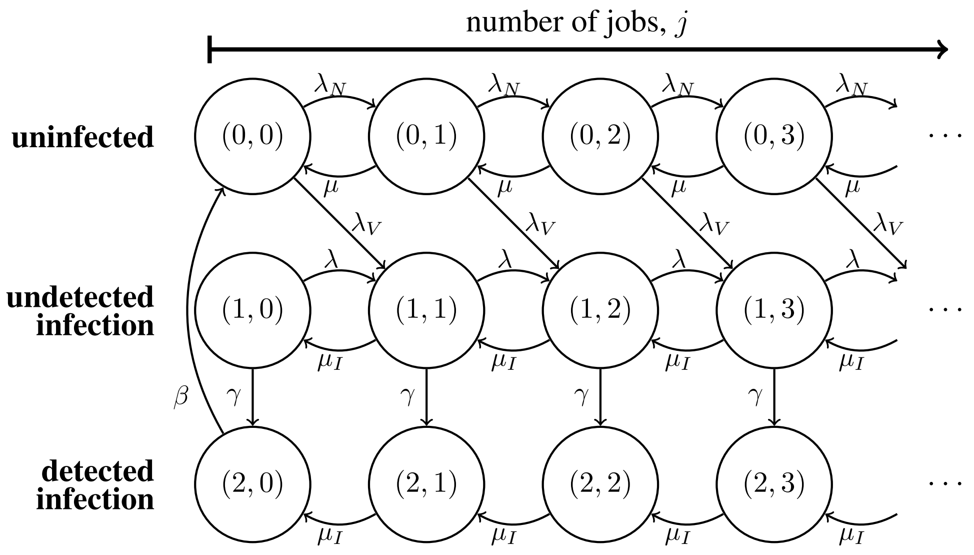


Figure 2.5: The Markov chain for a server vulnerable to viruses. State (m, j) indicates server state m (0=uninfected, 1=undetected infection, 2=detected infection) with j jobs in the system.

each can be in one of the above 3 states.

2.2.3 Server with virus infections

Imagine a computer server that is vulnerable to viruses. We present a stylized model where normally, the server is **uninfected** and receives jobs with rate λ and processes them with rate μ. While most jobs are normal (i.e., not virus carriers), arriving at rate λ_N, every once in a while, one of the arriving jobs brings with it a virus, with rate $\lambda_V = \lambda - \lambda_N$. The virus causes the server to become **infected**, reducing the server's service rate from μ to μ_I. It takes a duration of time distributed $\mathrm{Exponential}(\gamma)$ for the server to detect that it is infected. Once the infection is **detected**, the server stops accepting new jobs, and once all remaining jobs are processed, the server is able to use antivirus software to remove the virus in a duration of time distributed $\mathrm{Exponential}(\beta)$. Once the virus is removed, the server is again uninfected and will resume accepting jobs, processing them at a restored service rate of μ. We model a single server as being in one of 3 states, each of which will make up a *phase* of our Markov chain: **uninfected** ($m = 0$), **undetected infection** ($m = 1$), and **detected infection** ($m = 2$).

Fig. 2.5 shows a class $\mathbb{M}$ Markov that represents this setting. For this chain, $M = 2$, $j_0 = 1$, $\mathcal{N} = \{(0,0), (1,0), (2,0)\}$, $\lambda_0 = \lambda_N$, $\lambda_1 = \lambda = \lambda_N + \lambda_V$, $\lambda_2 = 0$, $\mu_0 = \mu$, $\mu_1 = \mu_2 = \mu_I$, $\alpha_0\langle 1; 1\rangle = \lambda_V$, and $\alpha_1\langle 1; 0\rangle = 0$ (all other $\alpha_m\langle m - i; \Delta\rangle$ transition rates are zero).

In Chapter 3 of this dissertation, class $\mathbb{M}$ Markov chains are used to study alternative models of computer systems that are susceptible to performance degrading cyber attacks.

2.3 Results

In this section we first present a key theorem from the literature that enables the CAP method (Theorem 2.1). We then introduce some preliminary notation, and an original result, Theorem 2.2. Finally, we present the main results of the chapter, Theorems 2.3, 2.5, and 2.7, the proofs of which will depend on both Theorems 2.1 and 2.2.

2.3.1 A key idea

Consider an ergodic CTMC with state space, S, and consider a nonempty proper subset, $A \subsetneq S$, with states $x, z \in A$. The CAP method involves calculating quantities of the form

$$\mathbb{E}_z\left[T_x^A\right] \equiv \mathbb{E}\begin{bmatrix}\text{cumulative time spent in state } x \text{ until next} \\ \text{transition leaving set } A\text{, given initial state } z\end{bmatrix}$$

in order to determine the limiting probabilities of the Markov chain of interest. Theorem 2.1 (from Theorem 5.5.1 of [93]) gives an expression for the limiting probabilities of the Markov chain in terms of the quantities $\mathbb{E}_z\left[T_x^A\right]$.

Theorem 2.1. *Suppose $A \subsetneq S$. Then for each $x \in A$, the limiting probability of being in state x, π_x, can be expressed as*

$$\pi_x = \sum_{y\in A^c}\sum_{z\in A}\pi_y q(y,z)\mathbb{E}_z\left[T_x^A\right],$$

where $q(y,z)$ is the transition rate from state y to state z and $A^c \equiv S\backslash A$.

Proof. See Theorem 5.5.1 of [93]. □

Intuitively, we are expressing the long run fraction of time that we reside in state x, π_x, as a weighted average of the cumulative time spent in state x during uninterrupted visits to states in A, $\mathbb{E}_z\left[T_x^A\right]$, conditioned on the choice of state, $z \in A$, by which we enter A. The weights in this average represent the rate at which visits to A via z occur, which involves conditioning on the states $y \in A^c$ by which one may transition to $z \in A$. We illustrate S, A, y, z, and x in Fig. 2.6.

As an example, consider the simple case where $A = \{x\}$. In this case, Theorem 2.1 yields

$$\pi_x = \sum_{y\in A^c}\sum_{z\in A}\pi_y q(y,z)\mathbb{E}_z\left[T_x^A\right] = \sum_{y\neq x}\pi_y q(y,x)\mathbb{E}_x\left[T_x^{\{x\}}\right] = \frac{\sum_{y\neq x}\pi_y q(y,x)}{\sum_{y\neq x} q(x,y)},$$

and so

$$\pi_x \sum_{y\neq x} q(x,y) = \sum_{y\neq x}\pi_y q(y,x),$$

which is simply the balance equation associated with state x.

Theorem 2.1 is used in [93] to establish the matrix-geometric structure of the stationary distribution of QBD chains. The same argument can be used to establish the matrix-geometric

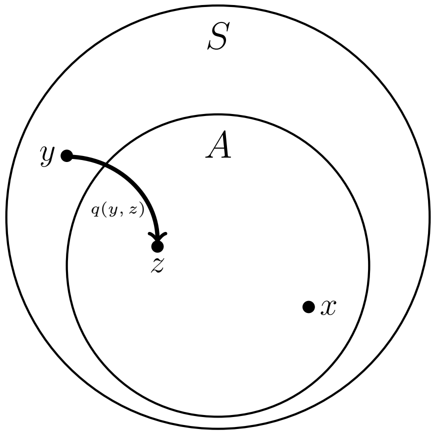


Figure 2.6: For any $x \in A$, Theorem 2.1 gives π_x as a linear combination of quantities $\mathbb{E}_z\left[T_x^A\right]$ by conditioning on the states $y \in A^c$, by which one may transition to states $z \in A$. This figure shows one such (y,z) pair.

structure satisfied by the stationary distribution, π, of a class $\mathbb{M}$ Markov chain on $\mathcal{R}$. Fix a level $j \geq j_0$, and define $A = \bigcup_{\ell \geq j+1} L_\ell$. Then for each state $(m, j+1) \in L_{j+1}$, we have

$$\begin{aligned}\pi_{(m,j+1)} &= \sum_{i=0}^{M}\sum_{k=0}^{M} \pi_{(i,j)} q((i,j),(k,j+1)) \mathbb{E}_{(k,j+1)}\left[T^A_{(m,j+1)}\right] \\ &= \sum_{i=0}^{M} \nu_{(i,j)}\pi_{(i,j)} \sum_{k=0}^{M}\left(\frac{q((i,j),(k,j+1))}{\nu_{(i,j)}}\right)\mathbb{E}_{(k,j+1)}\left[T^A_{(m,j+1)}\right] \\ &= \sum_{i=0}^{M} \pi_{(i,j)} R_{i,m},\end{aligned}$$

thus proving that $\vec{\pi}_{j+1} = \vec{\pi}_j \mathbf{R}$, since $R_{i,m}$ is $\nu_{(i,j)}$ times the expected amount of time the chain spends in state $(m, j+1)$ before returning to L_j, given it starts in state (i,j), and $\mathbf{R}$ is the rate matrix whose (i,m)th element is given by $R_{i,m}$.

We will soon see that the CAP method consists of applying Theorem 2.1 by choosing the set A in a different manner, while simultaneously observing that the resulting expected values of the form $\mathbb{E}_z\left[T_x^A\right]$ can be reinterpreted as tractable expected values associated with an M/M/1/clearing model.

2.3.2 Preliminaries

Our main results, and their proofs, will rely on the following notation:

- $P_m \equiv \{(m, j_0+1), (m, j_0+2), (m, j_0+3), \ldots\}$ is the set of states in phase m with level $j \geq j_0 + 1$ (i.e., the set of states in phase m of $\mathcal{R}$ excluding state (m, j_0)).
- $\rho_m \equiv \lambda_m / \mu_m$.

- $\phi_m(\cdot)$ is the Laplace Transform of the busy period (time to first reach state 0, given that one starts in state 1) of an M/M/1 Markov chain with arrival rate λ_m and departure rate μ_m:
$$\phi_m(s) \equiv \frac{s+\lambda_m+\mu_m-\sqrt{(s+\lambda_m+\mu_m)^2-4\lambda_m\mu_m}}{2\lambda_m}.$$
- The *bases* of our main theorem, r_m, are given by
$$r_m \equiv \begin{cases} \rho_m\phi_m(\alpha_m) & \text{if } \mu_m > 0 \\ \dfrac{\lambda_m}{\lambda_m+\alpha_m} & \text{if } \mu_m = 0, \end{cases} \tag{2.3}$$
recalling that
$$\alpha_m \equiv \sum_{i=m+1}^{M}\sum_{\Delta=-1}^{1}\alpha_m\langle i-m;\Delta\rangle.$$
- For convenience, we define the following quantity, which will appear frequently in our analysis:
$$\Omega_m \equiv \frac{r_m}{\lambda_m(1-r_m\phi_m(\alpha_m))}. \tag{2.4}$$

As a consequence of the ergodicity assumption on class $\mathbb{M}$ Markov chains, we have
- for any phase m, $\lambda_m \geq \mu_m$ implies $\alpha_m > 0$,
- and for any phase m, $\lambda_m = 0$ implies that there exists a phase, $i < m$, and $\Delta \in \{-1,0,1\}$ such that $\alpha_i\langle m-i;\Delta\rangle > 0$.

We also make the following observations:
- $r_m < 1$ for all phases, $m \in \{0,1,\ldots,M\}$.
- $r_m = 0$ if and only if $\lambda_m = 0$.
- $r_m = \rho_m$ whenever $\alpha_m = 0$ (e.g., when $m = M$, as $\alpha_M = 0$). This is because $\alpha_m = 0$ implies that $\mu_m > \lambda_m$ by the ergodicity assumption, which yields $\phi_m(0) = 1$.
- $\phi_m(s) = 0$ for all s whenever $\lambda_m > \mu_m = 0$, which follows from the expression for $\phi_m(s)$. Alternatively, this follows by observing that the busy period of a degenerate (non-ergodic) M/M/1 Markov chain with arrival rate $\lambda_m = 0$ is infinite.

We have the following fundamental result on class $\mathbb{M}$ Markov chains, which together with Theorem 2.1, will enable us to prove the main results of of the chapter (Theorems 2.3, 2.5, and 2.7).

Theorem 2.2. *For any Markov class $\mathbb{M}$ Markov chain, if $\lambda_m, \mu_m > 0$ and $\ell, j \geq j_0+1$, we have*
$$\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = \begin{cases} \Omega_m r_m^{j-\ell}\left(1-(r_m\phi_m(\alpha_m))^{\ell-j_0}\right) & \text{if } \ell \leq j \\ \Omega_m\phi_m(\alpha_m)^{\ell-j}\left(1-(r_m\phi_m(\alpha_m))^{j-j_0}\right) & \text{if } \ell \geq j. \end{cases} \tag{2.5}$$
with r_m as given in (2.3) *and Ω_m as given in* (2.4).

Proof. The proof of this result is deferred to Section 2.4, which is entirely focused on proving this result via clearing model analysis. □

The remainder of this section will present our main results, giving the stationary distribution of class $\mathbb{M}$ chains via the CAP method in three different cases. In Section 2.3.3, we consider the case where all bases, r_m, are distinct whenever they are nonzero. Distinct bases arise in many models where there is no structure connecting the transition rates associated with each phase. For example, the class $\mathbb{M}$ Markov chain representing the "server in different power states" model presented in Section 2.2.1 has distinct bases. In Section 2.3.4 we consider the case where all bases are the same (i.e., $r_0 = r_1 = \cdots = r_m$), while requiring that $\lambda_m, \mu_m > 0$, for simplicity. We study this setting because it is the simplest case featuring repeated nonzero bases. Finally, in Section 2.3.5 we proceed to the case where all bases except for r_M are the same (i.e., $r_0 = r_1 = \cdots = r_{M-1} \neq r_M$). This structure, which is studied in [125, 126], is common in settings where phase transitions are analogous across all phases, except for the final phase where there are no transitions to a further phase before the process transitions to the non-repeating portion. In this case, we again assume that $\lambda_m, \mu_m > 0$, for simplicity. While in principle, the CAP method can be used to determine the limiting probabilities of *any* class $\mathbb{M}$ Markov chain, for simplicity, we do not cover other cases (e.g., $r_1 = r_2 \neq r_3 = r_5 = r_7 \neq r_4 = r_6 \neq r_1$), as the computations become increasingly cumbersome.

2.3.3 The case where all nonzero bases are distinct

We are now ready to present our main result for the case where all nonzero bases, r_m, are distinct. Theorem 2.3 expresses the stationary distribution of such class $\mathbb{M}$ Markov chains as the solution to a finite system of linear equations.

Theorem 2.3. *For any class $\mathbb{M}$ Markov chain such that all nonzero bases $r_1, r_2, \ldots, r_M$—given in Equation (2.3)—are distinct (i.e., $r_m \neq r_i$ implies either $m \neq i$ or $r_m = \lambda_m = 0$), for all $j \geq j_0 + 1$, we have a limiting probability distribution of the form*

$$\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0},$$

where $\{c_{m,k}\}_{0 \leq k \leq m \leq M}$ are constants with respect to j. Moreover, together with $\{\pi_{(m,j_0)}\}_{0 \leq m \leq M}$ and $\{\pi_x\}_{x \in \mathcal{N}}$, the $\{c_{m,k}\}_{0 \leq k \leq m \leq M}$ values constitute $M(M+5)/2 + |\mathcal{N}| + 2$ "unknown variables" *satisfying the following system of $M(M+5)/2+|\mathcal{N}|+3$ linear equations:*

$$\begin{cases}
c_{m,k} = \dfrac{r_k r_m \left(\sum_{i=k}^{m-1} \sum_{\Delta=-1}^{1} c_{i,k}\alpha_i \langle m-i;\Delta\rangle r_k^{\Delta} \right)}{\lambda_m (r_k - r_m)(1-\phi_m(\alpha_m) r_k)} & (0 \le k < m \le M\colon r_m, r_k > 0) \\
c_{m,k} = \dfrac{\sum_{i=k}^{m-1} \sum_{\Delta=-1}^{1} c_{i,k}\alpha_i \langle m-i;\Delta\rangle r_k^{\Delta}}{\mu_m(1-r_k)+\alpha_m} & (0 \le k < m \le M\colon r_k > r_m = 0) \\
c_{m,k} = 0 & (0 \le k < m \le M\colon r_k = 0) \\
c_{m,m} = \pi_{(m,j_0)} - \sum_{k=0}^{m-1} c_{m,k} & (0 \le m \le M) \\
\pi_{(m,j_0)} = \dfrac{\mu_m \sum_{k=0}^{m} c_{m,k} r_k + \sum_{x\in\mathcal{N}} q(x,(m,j_0))\pi_x + \sum_{i=0}^{m-1}\sum_{\Delta=-1}^{0} \alpha_i\langle m-i;\Delta\rangle \pi_{(i,j_0-\Delta)}}{\lambda_m + \sum_{i=m+1}^{M}\sum_{\Delta=0}^{1} \alpha_m\langle i-m;\Delta\rangle + \sum_{x\in\mathcal{N}} q((m,j_0),x)} & (0 \le m \le M) \\
\pi_x = \dfrac{\sum_{m=0}^{M} q((m,j_0),x)\pi_{(m,j_0)} + \sum_{y\in\mathcal{N}} q(y,x)\pi_y}{\sum_{m=0}^{M} q(x,(m,j_0)) + \sum_{y\in\mathcal{N}} q(x,y)} & (x \in \mathcal{N}) \\
1 = \sum_{x\in\mathcal{N}} \pi_x + \sum_{m=0}^{M}\sum_{k=0}^{m} \dfrac{c_{m,k}}{1-r_k},
\end{cases}$$

where $q(x,y)$ denotes the transition rate from state x to state y.

We note before proving Theorem 2.3 that solving this system of equations *symbolically* will yield closed-form solutions for the limiting probabilities. Alternatively, if all parameter values are fixed and known, an exact numerical solution can be found by solving the system numerically using exact methods. Note that there is one more equation than there are unknowns, as is often the case in representations of limiting equations through balance equations. Although one equation can be omitted from the system, the normalization equation must be used in order to guarantee a unique solution.

It is also worth observing that once the values $\{\pi_x\}_{x\in\mathcal{N}}$ and $\{\pi_{(m,j_0)}\}_{0\le m\le M}$ are known, all other $c_{m,k}$ terms can be computed recursively, without having to apply Gaussian elimination to the entire linear system given in Theorem 2.3.

This recursion may also simplify further for some types of class $\mathbb{M}$ Markov chains. For example, if $\alpha_m\langle\Delta_1;\Delta_2\rangle = 0$ for all $\Delta_1 \ge 2$, $\Delta_2 \in \{-1,0,1\}$, and $0 \le m \le M$, then when all bases are positive, for any $k < m$, we have

$$c_{m,k} = c_{m-1,k}\frac{r_k r_m}{\lambda_m(r_k - r_m)(1-\phi_m(\alpha_m)r_k)} \sum_{\Delta=-1}^{1} \alpha_{m-1}\langle 1;\Delta\rangle r_k^{\Delta}$$

which further implies, for $k < m$,

$$c_{m,k} = c_{k,k}\prod_{\ell=1}^{m-k} \frac{r_k r_{k+\ell}}{\lambda_{k+\ell}(r_k - r_{k+\ell})(1-\phi_{k+\ell}(\alpha_{k+\ell})r_k)} \sum_{\Delta=-1}^{1} \alpha_{k+\ell-1}\langle 1;\Delta\rangle r_k^{\Delta}$$

meaning that only the $\{c_{k,k}\}_{0\le k\le M}$ terms need to be computed recursively.

Proof of Theorem 2.3. For simplicity, we present the proof for the case where $\lambda_m, \mu_m > 0$ for all phases $m \in \{0, 1, 2, \ldots, M\}$. The complete proof that includes the cases where one or both of λ_m and μ_m may be 0 for some phases, m, is given in Appendix A.2.

We prove the theorem via strong induction on the phase, m. Specifically, for each phase m, we will show that $\pi_{(m,j)}$ takes the form $\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0}$ for all $j \geq j_0 + 1$, and show that $\{c_{m,k}\}_{0 \leq k \leq m-1}$ satisfies

$$c_{m,k} = \frac{r_k r_m}{\lambda_m (r_k - r_m)(1 - \phi_m(\alpha_m) r_k)} \left(\sum_{i=k}^{m-1} \sum_{\Delta=-1}^{1} c_{i,k} \alpha_i \langle m - i; \Delta \rangle r_k^{\Delta} \right)$$

while $c_{m,m} = \pi_0 - \sum_{k=0}^{m-1} c_{m,k}$. Finally, after completing the inductive proof, we justify that the remaining linear equations in the proposed system are ordinary balance equations together with the normalization constraint.

Base case:

We begin our strong induction by verifying that the claim holds for the base case (i.e., for $m = 0$). In this case, Equation (2.5) yields

$$\mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right] = \Omega_0 r_0^{j-j_0-1}(1 - r_0 \phi_0(\alpha_0)) = \frac{r_0^{j-j_0}}{\lambda_0}.$$

We can now apply Theorem 2.1, yielding

$$\begin{aligned} \pi_{(0,j)} &= \pi_{(0,j_0)} \lambda_0 \mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right] = \pi_{(0,j_0)} \lambda_0 \left(\frac{r_0^{j-j_0}}{\lambda_0} \right) = \pi_{(0,j_0)} r_0^{j-j_0} \\ &= c_{0,0} r_0^{j-j_0}, \end{aligned}$$

where $c_{0,0} = \pi_{(0,j_0)}$. Hence, $\pi_{(0,j)}$ takes the claimed form. Moreover, $c_{0,0}$ satisfies the claimed constraint as $c_{0,0} = \pi_{(0,j_0)} - \sum_{k=0}^{m-1} c_{m,k} = \pi_{(0,j_0)} - 0 = \pi_{(0,j_0)}$, because the sum is empty when $m = 0$. Note that when $m = 0$, $\{c_{m,k}\}_{0 \leq k < m \leq M}$ is empty, and hence, there are no constraints on these values that require verification.

Helpful computations:

Before proceeding to the inductive step, we compute two useful expressions: First, we have $\lambda_m \mathbb{E}_{(m,j_0+1)}\left[T_{(m,j)}^{P_m}\right] = r_m^{j-j_0}$, which follows from applying Equation (2.5). Next, we have

$$\begin{aligned} \sum_{\ell=1}^{\infty} r_k^{\ell-j_0} \mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] &= \sum_{\ell=j_0+1}^{j} r_k^{\ell-j_0} \mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] + \sum_{\ell=j+1}^{\infty} r_k^{\ell-j_0} \mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] \\ &= \Omega_m \left(\sum_{\ell=j_0+1}^{j} r_k^{\ell-j_0} r_m^{j-\ell} \left(1 - (r_m \phi_m(\alpha_m))^{\ell-j_0}\right) \right. \\ &\qquad \left. + \sum_{\ell=j+1}^{\infty} r_k^{\ell-j_0} \phi_m(\alpha_m)^{\ell-j} \left(1 - (r_m \phi_m(\alpha_m))^{j-j_0}\right) \right) \\ &= \frac{r_k r_m (r_k^{j-j_0} - r_m^{j-j_0})}{\lambda_m (r_k - r_m)(1 - \phi_m(\alpha_m) r_k)}, \end{aligned}$$

where the last equality follows from well known geometric sum identities. Note that this expression is well-defined because $r_k \neq r_m$ by assumption and $r_m \phi_m(\alpha_m) \neq 1$.

Inductive step:

Next, we proceed to the inductive step and assume the induction hypothesis holds for all phases $i \in \{0, 1, \ldots, m-1\}$. In particular, we assume that $\pi_{(i,j)} = \sum_{k=0}^{i} c_{i,k} r_k^{j-j_0}$ for all $i < m$. Applying Theorem 2.1, the induction hypothesis, and our computations above, we have[3]

$$\begin{aligned}
\pi_{(m,j)} &= \pi_{(m,j_0)} \lambda_m \mathbb{E}_{(m,j_0+1)} \left[T_{(m,j)}^{P_m} \right] + \sum_{i=0}^{m-1} \sum_{\ell=1}^{\infty} \sum_{\Delta=-1}^{1} \pi_{(i,\ell-\Delta)} \alpha_i \langle m-i; \Delta \rangle \mathbb{E}_{(m,\ell)} \left[T_{(m,j)}^{P_m} \right] \\
&= \pi_{(m,j_0)} r_m^{j-j_0} + \sum_{i=0}^{m-1} \sum_{\ell=1}^{\infty} \sum_{\Delta=-1}^{1} \alpha_i \langle m-i; \Delta \rangle \left(\sum_{k=0}^{i} c_{i,k} r_k^{\ell-j_0-\Delta} \mathbb{E}_{(m,\ell)} \left[T_{(m,j)}^{P_m} \right] \right) \\
&= \pi_{(m,j_0)} r_m^{j-j_0} + \sum_{k=0}^{m-1} \sum_{i=k}^{m-1} \left(c_{i,k} \sum_{\Delta=-1}^{1} \alpha_i \langle m-i; \Delta \rangle r_k^{\Delta} \right) \left(\sum_{\ell=1}^{\infty} r_k^{\ell-j_0} \mathbb{E}_{(m,\ell)} \left[T_{(m,j)}^{P_m} \right] \right) \\
&= \pi_{(m,j_0)} r_m^{j-j_0} + \sum_{k=0}^{m-1} \sum_{i=k}^{m-1} \left(c_{i,k} \sum_{\Delta=-1}^{1} \alpha_i \langle m-i; \Delta \rangle r_k^{\Delta} \right) \left(\frac{r_k r_m (r_k^{j-j_0} - r_m^{j-j_0})}{\lambda_m (r_k - r_m)(1 - \phi_m(\alpha_m) r_k)} \right) \\
&= \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0},
\end{aligned}$$

where we have collected terms with

$$c_{m,k} = \frac{r_k r_m \left(\sum_{i=k}^{m-1} \sum_{\Delta=-1}^{1} c_{i,k} \alpha_i \langle m-i; \Delta \rangle r_k^{\Delta} \right)}{\lambda_m (r_k - r_m)(1 - \phi_m(\alpha_m) r_k)} \qquad (0 \leq k < m \leq M)$$

and $c_{m,m} = \pi_{(m,j_0)} - \sum_{k=0}^{m-1} c_{m,k}$, as claimed. This completes the inductive step and the proof by induction.

The balance equations and normalization constraint:

The equations with $\pi_{(m,j_0)}$ and π_x in their left-hand sides in our proposed system are ordinary balance equations (that have been normalized so that there are no coefficients on the left-hand side).

It remains to verify that the final equation, which is the normalization constraint:

$$\begin{aligned}
1 &= \sum_{x \in \mathcal{N}} \pi_x + \sum_{m=0}^{M} \pi_{(m,j_0)} + \sum_{m=0}^{M} \sum_{j=j_0+1}^{\infty} \pi_{(m,j)} \\
&= \sum_{x \in \mathcal{N}} \pi_x + \sum_{m=0}^{M} \sum_{k=0}^{M} c_{m,k} + \sum_{m=0}^{M} \sum_{k=0}^{m-1} \sum_{j=j_0+1}^{\infty} c_{m,k} r_k^{j-j_0} \\
&= \sum_{x \in \mathcal{N}} \pi_x + \sum_{m=0}^{M} \sum_{k=0}^{m} \frac{c_{m,k} r_k}{1 - r_k}.
\end{aligned}$$

□

[3] Note that we have also used the fact that $\pi_{(i,j_0)}$ also satisfies the claimed form for all $i < m$, which is true as $c_{i,i} = \pi_{(i,j_0)} - \sum_{k=0}^{i-1} c_{i,k}$ (from the inductive hypothesis) implies that $\pi_{(i,j_0)} = \sum_{k=0}^{i} c_{i,k} = \sum_{k=0}^{i} c_{i,k} r_k^0$.

2.3.4 The case where all bases agree

The CAP method can also be used in cases where some of the base terms coincide. We assume, for the sake of readability, that λ_m and μ_m are both positive for each phase m, but analogous results can still be derived when this is no longer the case.

In order to derive our result, we will make use of the following lemma: we omit the proof, but each formula can be derived using the lemmas contained in Appendix A.3.

Lemma 2.4. *For a class $\mathbb{M}$ Markov chain with all $\lambda_m, \mu_m > 0$ and $r_0 = r_1 = \cdots = r_M$, for each integer $u \geq 0$ and each integer $j \geq j_0 + 1$, we have the following three identities:*

- $$\sum_{\ell=j_0+2}^{\infty} \binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0} \mathbb{E}_{(m,\ell-1)}\left[T_{(m,j)}^{P_m}\right] = \sum_{k=1}^{u+1} \frac{\Omega_m r_0}{(1-r_0\phi_m(\alpha_m))^{u+1-k}} \binom{j-(j_0+1)+k}{k} r_0^{j-j_0},$$

- $$\sum_{\ell=j_0+1}^{\infty} \binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0} \mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = \Omega_m \binom{j-(j_0+1)+u+1}{u+1} r_0^{j-j_0} + \sum_{k=1}^{u} \frac{\Omega_m r_0 \phi_m(\alpha_m)}{(1-r_0\phi_m(\alpha_m))^{u+1-k}} \binom{j-(j_0+1)+k}{k} r_0^{j-j_0},$$

- $$\sum_{\ell=j_0+1}^{\infty} \binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0} \mathbb{E}_{(m,\ell+1)}\left[T_{(m,j)}^{P_m}\right] = \frac{\Omega_m}{r_0} \binom{j-(j_0+1)+u+1}{u+1} r_0^{j-j_0} - \binom{j-(j_0+1)+u}{u} \frac{r_0^{j-j_0}}{\lambda_m} + \sum_{k=1}^{u} \frac{\Omega_m r_0 \phi_m(\alpha_m)^2}{(1-r_0\phi_m(\alpha_m))^{u+1-k}} \binom{j-(j_0+1)+k}{k} r_0^{j-j_0}.$$

Theorem 2.5. *For a class $\mathbb{M}$ Markov chain with all $\lambda_m, \mu_m > 0$ and $r_0 = r_1 = \cdots = r_M$, for all $0 \leq m \leq M$, $j \geq j_0$, we have*

$$\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} \binom{j-(j_0+1)+k}{k} r_0^{j-j_0}$$

where the $\{c_{m,k}\}_{0\le k\le m\le M}$ *values satisfy the system of linear equations*

$$\begin{cases} c_{m,0} = \pi_{(m,j_0)}, & (0 \le m \le M) \\ c_{m,k} = \Omega_m r_0 \sum_{u=k}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\left[\sum_{\Delta=-1}^{1} \frac{\alpha_i\langle m-i;\Delta\rangle \phi_m(\alpha_m)^{\Delta+1}}{(1-r_0\phi_m(\alpha_m))^{u+1-k}}\right] & \\ \qquad - \frac{1}{\lambda_m}\sum_{i=k}^{m-1} c_{i,k}\alpha_i\langle m-i;1\rangle & \\ \qquad + \Omega_m \sum_{i=k-1}^{m-1} c_{i,k-1}\left[\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_0^{-\Delta}\right] & (1\le k\le m-1) \\ c_{m,m} = c_{m-1,m-1}\Omega_m \sum_{\Delta=-1}^{1}\alpha_{m-1}\langle 1;\Delta\rangle r_0^{\Delta} & (1\le m\le M), \end{cases}$$

together with the usual balance equations and normalization constraint.

Proof. Starting with phase 0, we observe as before that, for $j \ge j_0+1$,

$$\pi_{(0,j)} = \pi_{(0,j_0)}\lambda_0 \mathbb{E}_{(0,j_0+1)}\left[T^{P_0}_{(0,j)}\right] = \pi_{(0,j_0)} r_0^{j-j_0}$$

and this equality is clearly also valid when $j = j_0$.

We now proceed by induction. Assuming the result holds for $\pi_{(i,\ell)}$ for $0 \le i \le m-1$, $\ell \ge j_0$, we have

$$\begin{aligned} \pi_{(m,j)} &= \pi_{(m,j_0)}\lambda_m \mathbb{E}_{(m,j_0+1)}\left[T^{P_m}_{(m,j)}\right] + \sum_{i=0}^{m-1}\sum_{\ell=j_0+2}^{\infty} \pi_{(i,\ell)}\alpha_i\langle m-i;-1\rangle \mathbb{E}_{(m,\ell-1)}\left[T^{P_m}_{(m,j)}\right] \\ &\quad + \sum_{i=0}^{m-1}\sum_{\ell=j_0+1}^{\infty} \pi_{(i,\ell)}\alpha_i\langle m-i;0\rangle \mathbb{E}_{(m,\ell)}[T^{P_m}_{(m,j)}] \\ &\quad + \sum_{i=0}^{m-1}\sum_{\ell=j_0}^{\infty} \pi_{(i,\ell)}\alpha_i\langle m-i;1\rangle \mathbb{E}_{(m,\ell+1)}[T^{P_m}_{(m,j)}] \\ &= \pi_{(m,j_0)} r_0^{j-j_0} + \sum_{i=0}^{m-1} c_{i,0}\Omega_m\left[\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;-1\rangle r_0^{-\Delta}\right]\binom{j-(j_0+1)+1}{1} r_0^{j-j_0} \\ &\quad + \sum_{u=1}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\alpha_i\langle m-i;-1\rangle \sum_{\ell=j_0+2}^{\infty}\binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0}\mathbb{E}_{(m,\ell-1)}\left[T^{P_m}_{(m,j)}\right] \\ &\quad + \sum_{u=1}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\alpha_i\langle m-i;0\rangle \sum_{\ell=j_0+1}^{\infty}\binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\ &\quad + \sum_{u=1}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\alpha_i\langle m-i;1\rangle \sum_{\ell=j_0+1}^{\infty}\binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0}\mathbb{E}_{(m,\ell+1)}\left[T^{P_m}_{(m,j)}\right] \end{aligned}$$

and after applying Lemma 2.4 and simplifying, we conclude that

$$\begin{aligned}\pi_{(m,j)} &= \pi_{(m,j_0)} r_0^{j-j_0} \\ &+ \sum_{k=0}^{m-1}\sum_{i=k}^{m-1} c_{i,k}\Omega_m \left[\sum_{\Delta=-1}^{1} \alpha_i\langle m-i;\Delta\rangle r_0^{-\Delta}\right] \binom{j-(j_0+1)+k+1}{k+1} r_0^{j-j_0} \\ &+ \sum_{k=1}^{m-1}\sum_{u=k}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\Omega_m r_0 \left[\sum_{\Delta=-1}^{1} \frac{\alpha_i\langle m-i;\Delta\rangle \phi_m(\alpha_m)^{-\Delta+1}}{(1-r_0\phi_m(\alpha_m))^{u+1-k}}\right] \binom{j-(j_0+1)+k}{k} r_0^{j-j_0} \\ &- \frac{1}{\lambda_m}\sum_{k=1}^{m-1}\sum_{i=k}^{m-1} c_{i,k}\alpha_i\langle m-i;1\rangle \binom{j-(j_0+1)+k}{k} r_0^{j-j_0} \\ &= \sum_{k=0}^{m} c_{m,k}\binom{j-(j_0+1)+k}{k} r_0^{j-j_0},\end{aligned}$$

where we have collected terms so that for $1 \le k \le m-1$ we have

$$\begin{aligned}c_{m,k} &= \sum_{u=k}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\Omega_m r_0 \left[\sum_{\Delta=-1}^{1} \frac{\alpha_i\langle m-i;\Delta\rangle \phi_m(\alpha_m)^{-\Delta+1}}{(1-r_0\phi_m(\alpha_m))^{u+1-k}}\right] \\ &\quad - \frac{1}{\lambda_m}\sum_{i=k}^{m-1} c_{i,k}\alpha_i\langle m-i;1\rangle \\ &\quad + \Omega_m \sum_{i=k-1}^{m-1} c_{i,k-1}\left[\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_0^{-\Delta}\right],\end{aligned}$$

while $c_{m,0} = \pi_{(m,j_0)}$ and $c_{m,m} = c_{m-1,m-1}\Omega_m\left[\sum_{\Delta=-1}^{1}\alpha_{m-1}\langle 1;\Delta\rangle r_0^{-\Delta}\right]$, as claimed. □

2.3.5 The case where all bases except r_M agree

We conclude this section by considering the case where $r_0 = r_1 = \cdots = r_{M-1} \neq r_M$, as this case is satisfied by the Markov chains studied in [125, 126]. The following lemma can be used to compute the limiting probability distribution. The proof is again omitted, but as with Lemma 2.4, each formula can be derived using the lemmas contained in Appendix A.3.

Lemma 2.6. *For a class $\mathbb{M}$ Markov chain with all $\lambda_m, \mu_m > 0$ and $r_0 = r_1 = \cdots = r_{M-1} \neq r_M$, for each integer $u \ge 0$, we have the following three identities:*

- $$\begin{aligned}&\sum_{\ell=j_0+2}^{\infty}\binom{\ell-(j_0+1)+u}{u} r_0^{\ell-j_0}\mathbb{E}_{(M,\ell-1)}\left[T_{(M,j)}^{P_M}\right] \\ &= -\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1}} - \frac{1}{(1-\frac{r_0}{r_M})^{u+1}}\right] r_M^{j-j_0} \\ &\quad + \sum_{k=0}^{u}\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1-k}} - \frac{1}{(1-\frac{r_0}{r_M})^{u+1-k}}\right]\binom{j-(j_0+1)+k}{k} r_0^{j-j_0},\end{aligned}$$

- $$\sum_{\ell=j_0+1}^{\infty}\binom{\ell-(j_0+1)+u}{u}r_0^{\ell-j_0}\mathbb{E}_{(M,\ell)}\left[T_{(M,j)}^{P_M}\right]$$
$$= -\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1}}-\frac{1}{r_M(1-\frac{r_0}{r_M})^{u+1}}\right]r_M^{j-j_0}$$
$$+\sum_{k=0}^{u}\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1-k}}-\frac{1}{r_M(1-\frac{r_0}{r_M})^{u+1-k}}\right]\binom{j-(j_0+1)+k}{k}r_0^{j-j_0},$$

- $$\sum_{\ell=j_0+1}^{\infty}\binom{\ell-(j_0+1)+u}{u}r_0^{\ell-j_0}\mathbb{E}_{(M,\ell+1)}\left[T_{(M,j)}^{P_M}\right]$$
$$= -\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1}}-\frac{1}{r_M^2(1-\frac{r_0}{r_M})^{u+1}}\right]r_M^{j-j_0}-\binom{j-(j_0+1)+u}{u}\frac{r_0^{j-j_0}}{\lambda_M}$$
$$+\sum_{k=0}^{u}\Omega_M r_0\left[\frac{1}{(1-r_0)^{u+1-k}}-\frac{1}{r_M^2(1-\frac{r_0}{r_M})^{u+1-k}}\right]\binom{j-(j_0+1)+k}{k}r_0^{j-j_0}.$$

Our next theorem gives an expression for the stationary distribution of a class $\mathbb{M}$ Markov chain when $r_0 = r_1 = \cdots = r_{M-1} \neq r_M$. As the proof is similar to those of Theorems 2.3 and 2.5, we omit the proof.

Theorem 2.7. *Suppose a class $\mathbb{M}$ Markov chain has all $\lambda_m, \mu_m > 0$ and $r_0 = r_1 = \cdots = r_{M-1} \neq r_M$. Then, for all $0 \leq m \leq M-1$, $j \geq j_0$,*

$$\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k}\binom{j-(j_0+1)+k}{k}r_0^{j-j_0},$$
$$\pi_{(M,j)} = \sum_{k=0}^{M-1} c_{M,k}\binom{j-(j_0+1)+k}{k}r_0^{j-j_0} + c_{M,M}r_M^{j-j_0}$$

where the $\{c_{m,k}\}_{0\le k\le m\le M}$ *values satisfy the system of linear equations*

$$\begin{cases}
c_{m,0} = \pi_{(m,j_0)} & (0 \le m < M) \\
c_{m,k} = \Omega_m r_0 \sum_{u=k}^{m-1}\sum_{i=u}^{m-1} c_{i,u}\left[\sum_{\Delta=-1}^{1} \frac{\alpha_i\langle m-i;\Delta\rangle \phi_m(\alpha_m)^{\Delta+1}}{(1-r_0\phi_m(\alpha_m))^{u+1-k}}\right] & \\
\quad + \Omega_m \sum_{i=k-1}^{m-1} c_{i,k-1}\left[\sum_{\Delta=-1}^{1} \alpha_i\langle m-i;\Delta\rangle) r_0^{-\Delta}\right] & \\
\quad - \frac{1}{\lambda_m}\sum_{i=k}^{m-1} c_{i,k}\alpha_i\langle m-i;1\rangle & (1 \le k < m < M) \\
c_{m,m} = c_{m-1,m-1}\Omega_m\left[\sum_{\Delta=-1}^{1}\alpha_{m-1}\langle 1;\Delta\rangle r_0^{\Delta}\right] & (1 \le m < M) \\
c_{M,0} = \sum_{i=0}^{M-1} c_{i,0}\Omega_M r_0\left[\sum_{\Delta=-1}^{1}\left[\frac{1}{1-r_0} - \frac{1}{r_M^{\Delta+1}(1-\frac{r_0}{r_M})}\right]\alpha_i\langle m-i;\Delta\rangle\right] & \\
\quad + \sum_{u=1}^{M-1}\sum_{i=u}^{M-1} c_{i,u}\Omega_M r_0\left[\sum_{\Delta=-1}^{1}\alpha_i\langle M-i;\Delta\rangle\left[\frac{1}{(1-r_0)^{u+1}}\right.\right. & \\
\quad\quad \left.\left. - \frac{1}{r_M^{\Delta+1}(1-\frac{r_0}{r_M})^{u+1}}\right]\right] & \\
c_{M,k} = -\sum_{i=k}^{M-1} c_{i,k}\frac{\alpha_i\langle M-i;1\rangle}{\lambda_M} & \\
\quad + \sum_{u=k}^{M-1}\sum_{i=u}^{M-1} c_{i,u}\Omega_M r_0\left[\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle\left[\frac{1}{(1-r_0)^{u+1-k}}\right.\right. & \\
\quad\quad \left.\left. - \frac{1}{r_M^{\Delta+1}(1-\frac{r_0}{r_M})^{u+1-k}}\right]\right] & (1 \le k < M) \\
c_{M,M} = \pi_{(M,j_0)} - c_{M,0}, &
\end{cases}$$

together with the usual balance equations and normalization constraint.

2.4 Analysis of the M/M/1/clearing model

In this section we present an analysis of the M/M/1/clearing model Markov chain in order to prove Theorem 2.2 (presented in Section 2.3.2), which we used in the proof of Theorems 2.3 2.5, and 2.7. This analysis provides the framework on which the CAP method is built.

Like the ordinary M/M/1 model, the M/M/1/clearing model Markov chain (see Fig. 2.7) has state space $\{0, 1, 2, 3, \ldots\}$, with an arrival rate of $\lambda \equiv q(j, j+1)$ (for all $j \ge 0$) and a departure rate of $\mu \equiv q(j, j-1)$ (for all $j \ge 2$). In addition, all nonzero states in the M/M/1/clearing model have an additional transition to state 0 representing a *clearing* (also known as a catastrophe or disaster). All clearing transitions occur with the same rate $\alpha \equiv q(j, 0)$ (for all $j \ge 2$), which we call the *clearing rate*. Note that from state 1, there are two "ways" of transitioning to state 0—a departure or a clearing—and hence, $q(1, 0) = \mu + \alpha$. We observe that each phase, m, of

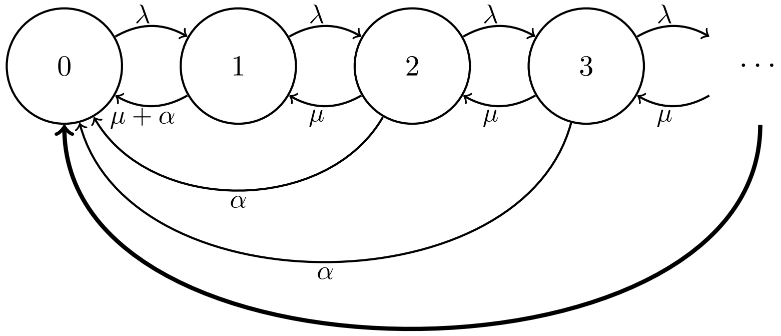


Figure 2.7: Markov chain for the M/M/1/clearing model. For any state $j \geq 0$, there is a clearing rate with rate α. Note that the transition rate from state 1 to state 0 is $\mu + \alpha$ as either a departure or a clearing can cause this transition. The thicker arrow denotes a *set* of transitions.

a class $\mathbb{M}$ Markov chain (for levels $j \geq j_0 + 1$) behaves like an M/M/1/clearing Markov chain, with clearing rate

$$\alpha_m \equiv \sum_{i=m+1}^{M} \sum_{\Delta=-1}^{1} \alpha_m \langle i - m; \Delta \rangle,$$

except with "clearings" transitioning to a different phase.

2.4.1 Preliminary results on clearing models

In this section we present two preexisting results from the literature that will aid us in proving Theorem 2.2. Our first result gives the limiting probability distribution of the M/M/1/clearing model: see e.g., Corollary 4.2.2 of [4], as well as Exercise 10.7 of [70].

Lemma 2.8. *In an M/M/1/clearing model with arrival, departure, and clearing rates λ, μ, and α, respectively, the limiting probability distribution is given by*

$$\pi_j = (1 - \rho\phi(\alpha))(\rho\phi(\alpha)^j),$$

where $\rho = \lambda/\mu$ and $\phi(\cdot)$ is the Laplace transform of the busy period of an M/M/1 system:

$$\phi(s) = \frac{s + \lambda + \mu - \sqrt{(s + \lambda + \mu)^2 - 4\lambda\mu}}{2\lambda}.$$

Proof. See the proof of Corollary 4.2.2 of [4].

□

The next result is also known, and gives an expression for a probability that is useful in computing values of the form $\mathbb{E}_\ell \left[T_j^A\right]$ in the M/M/1 clearing model. A similar result, presented in the context of Brownian motion, is given in Problems 22 and 23 from Chapter 7 of [87].

Lemma 2.9. *In an M/M/1/clearing model with arrival, departure, and clearing rates λ, μ, and α, respectively, the probability that one reaches state $j > 0$ before state 0, given that one starts in state $\ell > 0$, is given by*

$$p_{\ell\to j} = \begin{cases} \dfrac{(\rho\phi(\alpha))^{j-\ell}(1-(\rho\phi(\alpha)^2)^{\ell})}{1-(\rho\phi(\alpha)^2)^{j}} & \textit{if } \ell \le j \\ \phi(\alpha)^{\ell-j} & \textit{if } \ell \ge j. \end{cases}$$

Proof. First, note that in the degenerate case where $\ell = j$, we are already at state j from the start, and so we reach state j before reaching state 0 *surely*, yielding $p_{\ell\to j} = 1$. Substituting $\ell = j$ in either branch of the claimed expression for $p_{\ell\to j}$ yields 1, validating the claim in this case.

Next, we consider the case where $\ell > j$, which will be the simpler of the two remaining cases. In this case, $p_{\ell\to j}$ can be viewed as the probability that the sum of $\ell - j$ independent M/M/1 busy periods (without clearing), $B_1, B_2, \ldots, B_{\ell-j}$, do not exceed the exponentially distributed "clearing" random variable ζ_α:

$$p_{\ell\to j} = \mathbb{P}\left(\sum_{n=1}^{j-\ell} B_n \le \zeta_\alpha\right) = \mathbb{E}\left[e^{-\alpha\sum_{n=1}^{j-\ell} B_n}\right] = \phi(\alpha)^{j-\ell}$$

as claimed, with the next-to-last equality following from the alternate interpretation of the Laplace Transform (see Appendix A.1 for details).

Now let us consider the remaining case where $\ell < j$. In this case, it will be helpful to consider two Poisson processes, one associated with arrivals, occurring with rate λ, and the other associated with departures, occurring with rate μ. Departures can happen even at state 0, although at state 0 departures *do not cause a change of state*. Let $N_A(t)$ and $N_D(t)$ be the number of such arrivals and departures during time interval $[0, t]$, assuming that we are in state ℓ at time 0.

Next, let $\tau_0 = \inf\{t\colon \ell + (N_A(t) - N_D(t)) = 0\}$ be the first time after 0 until we have ℓ departures in excess of arrivals, and let $\tau_j = \inf\{t\colon \ell + (N_A(t) - N_D(t)) = j\}$ be the first time after 0 until we have $j - \ell$ arrivals in excess of departures. Although there may be positive probability that one of of these two events may never happen (i.e., $\max\{\tau_0, \tau_j\} = +\infty$), at least one of these events will happen almost surely. Moreover, if either of these events happens before a clearing, which will occur at time $\zeta_\alpha \sim \text{Exponential}(\alpha)$ (independent of both τ_0 and τ_j), then τ_0 and τ_j describe the first time that we will reach state 0 and j, respectively.

Given this notation, we can express $p_{\ell\to j}$, the probability that one next reaches state $j > \ell$ before state 0 in an M/M/1/clearing model, given that one starts in state $\ell > 0$, by

$$p_{\ell\to j} = \mathbb{P}(\tau_j \le \min\{\tau_0, \zeta_\alpha\}) = \mathbb{E}[e^{-\alpha\tau_0} \cdot I\{\tau_0 < \tau_j\}],$$

where $I\{\cdot\}$ is the indicator function. Similarly, if we let $p_{\ell\not\to j}$ be the probability that we reach 0—via departures, rather than via a clearing—before reaching j and before a clearing, we have

$$p_{\ell\not\to j} = \mathbb{P}(\tau_0 \le \min\{\tau_j, \zeta_\alpha\}) = \mathbb{E}[e^{-\alpha\tau_j} \cdot I\{\tau_j < \tau_0\}].$$

At this point, it will be useful to compute the quantities $\mathbb{E}[e^{-\alpha\tau_0}]$ and $\mathbb{E}[e^{-\alpha\tau_j}]$. Observe that τ_0 is the time until we first have ℓ departures in excess of arrivals. We can think of each time "departures minus arrivals" increments by one as the completion of an M/M/1 busy period. Hence,

τ_0 corresponds to the time until we have completed ℓ consecutive independent busy periods. Meanwhile, τ_j is the time until we first have $j - \ell > 0$ arrivals in excess of departures. Just as we can think each time "departures minus arrivals" increments by one as the completion of an M/M/1 busy period, we can also think of the each time "arrivals minus departures" increments by one as the completion of an M/M/1 busy period where we think of arrivals as "departures" occurring with rate λ and departures as "arrivals" occurring with rate μ. Hence, τ_j corresponds to the time until we have completed $j - \ell$ consecutive independent busy periods with arrival rate μ and departure rate λ. Consequently

$$\mathbb{E}[e^{-\alpha\tau_0}] = \phi(\alpha)^\ell, \qquad \mathbb{E}[e^{-\alpha\tau_j}] = \eta(\alpha)^{j-\ell},$$

where $\phi(\cdot)$ and $\eta(\cdot)$ are the Laplace transforms of the M/M/1 busy periods with arrival and departure rate pairs (λ, μ) and (μ, λ), respectively. We observe that for all $s > 0$,

$$\begin{aligned}\eta(s) &= \frac{s+\mu+\lambda-\sqrt{(s+\mu+\lambda)^2-4\mu\lambda}}{2\mu}\\ &= \left(\frac{\lambda}{\mu}\right)\left(\frac{s+\lambda+\mu-\sqrt{(s+\lambda+\mu)^2-4\lambda\mu}}{2\lambda}\right)\\ &= \rho\phi(s).\end{aligned}$$

Note that in the case that $\rho \neq 1$, we must have $\eta(0) \neq \phi(0)$, and in particular one of these transforms will not evaluate to 1. This is not a problem as if $\rho < 1$ (respectively, $\rho > 1$), the underlying random variable of η (respectively, ϕ) has positive probability mass at infinity, and will thus not satisfy the "usual" condition of Laplace transforms evaluating to 1 at 0.

We proceed to use these expectations to determine $p_{\ell\to j}$:

$$\begin{aligned}\phi(\alpha)^\ell &= \mathbb{E}[e^{-\alpha\tau_0}]\\ &= \mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_0<\tau_j\}] + \mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_j<\tau_0\}]\\ &= \mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_0<\tau_j\}] + \phi(\alpha)^j\cdot\mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_j<\tau_0\}]\\ &= p_{\ell\not\to j} + \phi(\alpha)^j p_{\ell\to j},\\ (\rho\phi(\alpha))^{j-\ell} &= \mathbb{E}[e^{-\alpha\tau_j}]\\ &= \mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_0<\tau_j\}] + \mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_j<\tau_0\}]\\ &= (\rho\phi(\alpha))^j\cdot\mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_0<\tau_j\}] + \mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_j<\tau_0\}]\\ &= (\rho\phi(\alpha))^j(p_{\ell\not\to j}) + p_{\ell\to j}.\end{aligned}$$

We justify $\mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_j<\tau_0\}] = \phi(\alpha)^j\cdot\mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_j<\tau_0\}]$ by observing that given that $\tau_j < \tau_0$, we reach state j before state 0 (ignoring clearings), so we can only reach state 0 by performing j consecutive busy periods after reaching j. We justify the analogous equality $\mathbb{E}[e^{-\alpha\tau_j}\cdot I\{\tau_0<\tau_j\}] = (\rho\phi(\alpha))^j\cdot\mathbb{E}[e^{-\alpha\tau_0}\cdot I\{\tau_0<\tau_j\}]$ by observing that given that $\tau_0 < \tau_j$, we reach state 0 before state j (ignoring clearings), so we can only reach state j by performing j consecutive "busy" periods in an M/M/1 model with arrival rate μ and departure rate λ.

We now have a system of two linear equations in the two unknowns, $p_{\ell\to j}$ and $p_{\ell\not\to j}$. Solving the system for $p_{\ell\to j}$ and simplifying, we find that

$$p_{\ell\to j} = \frac{(\rho\phi(\alpha))^{j-\ell} - (\rho\phi(\alpha))^{j}\phi(\alpha)^{\ell}}{1-(\rho\phi(\alpha))^{j}\phi(\alpha)^{j}} = \frac{(\rho\phi(\alpha))^{j-\ell}(1-(\rho\phi(\alpha)^2)^{\ell})}{1-(\rho\phi(\alpha)^2)^{j}},$$

which proves the claim. □

2.4.2 Applying clearing model analysis toward proving Theorem 2

We now use Lemmas 2.8 and 2.9 to compute $\mathbb{E}_\ell\left[T_j^A\right]$ in an M/M/1/clearing model, where A is the set of nonzero states. This result is presented in Lemma 2.10. Finally, we will recast Lemma 2.10 in the context of class $\mathbb{M}$ Markov chains, allowing us to prove Theorem 2.2 from Section 2.3.2.

Lemma 2.10. *In an M/M/1/clearing model with arrival, departure, and clearing rates λ, μ, and α, respectively, if $A = \{1,2,3,\ldots\}$ denotes the set of nonzero states of the state space of the underlying Markov chain, then*

$$\mathbb{E}_\ell\left[T_j^A\right] = \begin{cases} \dfrac{(\rho\phi(\alpha))^{j-\ell+1}\left(1-(\rho\phi(\alpha)^2)^{\ell}\right)}{\lambda(1-\rho\phi(\alpha)^2)} & \text{if } \ell \le j \\ \\ \dfrac{\rho\phi(\alpha)^{\ell-j+1}\left(1-(\rho\phi(\alpha)^2)^{j}\right)}{\lambda(1-\rho\phi(\alpha)^2)} & \text{if } \ell \ge j. \end{cases}$$

Proof. We first consider the case where $\ell \le j$. We claim that

$$\mathbb{E}_1\left[T_j^A\right] = (p_{1\to\ell})\mathbb{E}_\ell\left[T_j^A\right], \tag{2.6}$$

recalling that $p_{1\to\ell}$ is the probability that one reaches state ℓ before state 0 given initial state 1. Equivalently, in our setting, we may interpret $p_{1\to\ell}$ to be the probability that one reaches state ℓ before leaving A, given initial state 1, as 0 is the only state not in A. The claim in Equation (2.6) follows from conditional expectation and the fact that given that we start in state 1, we either

- reach state ℓ before leaving A, in which case the the expected cumulative time spent in state j before leaving A is $\mathbb{E}_\ell\left[T_j^A\right]$ (note that no time is spent in j before reaching ℓ, as $\ell \le j$),
- or we do not reach state ℓ before leaving A, in which case we also do not reach state j, and hence we spend 0 time in state j before leaving A.

From Lemma 2.9, we know that for $\ell \le j$, we have

$$p_{\ell\to j} = \frac{(\rho\phi(\alpha))^{j-\ell}\left(1-(\rho\phi(\alpha)^2)^{\ell}\right)}{1-(\rho\phi(\alpha)^2)^{j}}. \tag{2.7}$$

Hence, in order to determine $\mathbb{E}_\ell\left[T_j^A\right]$ from Equation (2.6), we need only determine $\mathbb{E}_1\left[T_j^A\right]$. We compute this quantity via the renewal reward theorem. Let us earn reward in state j at rate 1, and consider a cycle from state 0 until one returns to 0 again (after leaving 0). We also use the fact from Lemma 2.8 that the limiting probability of being in state j in an M/M/1/clearing model is given by $(1-\rho\phi(\alpha))(\rho\phi(\alpha))^j$. Hence, by the renewal reward theorem, we have

$$\frac{\mathbb{E}_1\left[T_j^A\right]}{\mathbb{E}[B_C]+1/\lambda} = (1-\rho\phi(\alpha))(\rho\phi(\alpha))^j, \tag{2.8}$$

where B_C denotes the busy period of an M/M/1/clearing model. To determine $\mathbb{E}[B_C]$, observe that $B_C = \min\{B, \zeta_\alpha\}$, where B is an independent random variable distributed like the busy period of an M/M/1 model *without* clearing, and $\zeta_\alpha \sim \text{Exponential}(\alpha)$ is an exponentially distributed clearing time. Taking the expectation, we have

$$\begin{aligned}\mathbb{E}[B_C] &= \mathbb{E}[\min(B,\zeta_\alpha)] = \int_0^\infty \mathbb{P}(B>t)\mathbb{P}(\zeta_\alpha>t)\,dt = \int_0^\infty \mathbb{P}(B\ge t)e^{-\alpha t}\,dt \\ &= \frac{1}{\alpha}\int_0^\infty \mathbb{P}(B\ge t)\left(\alpha e^{-\alpha t}\right)\,dt = \frac{\mathbb{P}(B>\zeta_\alpha)}{\alpha} = \frac{1-\mathbb{P}(B\le\zeta_\alpha)}{\alpha} \\ &= \frac{1-\phi(\alpha)}{\alpha},\end{aligned}$$

where the final step follows from an alternate interpretation of the Laplace transform (see Appendix A.1 for details), noting that $\phi(\cdot)$ is the Laplace transform of B.

Returning to Equation (2.8), we find that

$$\mathbb{E}_1\left[T_j^A\right] = \left(\frac{1-\phi(\alpha)}{\alpha}+\frac{1}{\lambda}\right)(1-\rho\phi(\alpha))(\rho\phi(\alpha))^j = \frac{(\rho\phi(\alpha))^j}{\lambda}, \tag{2.9}$$

where we make use of the identity

$$\left(\frac{1-\phi(\alpha)}{\alpha}+\frac{1}{\lambda}\right)(1-\rho\phi(\alpha)) = \frac{1}{\lambda}$$

in our simplification. This identity can be verified algebraically by using the explicit form of $\phi(s)$. Alternatively, let $\mathbb{E}_0[T_0]$ be the expected duration of time spent in state 0 in a cycle starting from state 0, and ending with a return to state 0 from a nonzero state. Then by the renewal reward theorem,

$$\mathbb{E}_0[T_0] = \left(\mathbb{E}[B_C]+\frac{1}{\lambda}\right)(1-\rho\phi(\alpha)) = \left(\frac{1-\phi(\alpha)}{\alpha}+\frac{1}{\lambda}\right)(1-\rho\phi(\alpha)).$$

We can also observe that during such a cycle, the only time spent in state 0 is during the initial residence, as a revisit to state 0 ends the cycle, so $\mathbb{E}_0[T_0] = 1/\lambda$. Setting these quantities equal to one another yields the claimed identity directly.

We proceed to use Equation (2.6) in determining $\mathbb{E}_\ell\left[T_j^A\right]$ (in the case where $\ell \le j$), by substituting in values from Equations (2.7) and (2.9):

$$\begin{aligned}\mathbb{E}_\ell\left[T_j^A\right] &= \frac{\mathbb{E}_1\left[T_j^A\right]}{p_{1\to\ell}} = \left(\frac{(\rho\phi(\alpha))^j}{\lambda}\right)\left(\frac{1-(\rho\phi(\alpha)^2)^\ell}{(\rho\phi(\alpha))^{\ell-1}(1-\rho\phi(\alpha)^2)}\right) \\ &= \frac{(\rho\phi(\alpha))^{j-\ell+1}\left(1-(\rho\phi(\alpha)^2)^\ell\right)}{\lambda(1-\rho\phi(\alpha)^2)}.\end{aligned}$$

Next, we consider the case where $\ell \geq j$ (note that the two branches in the claimed expression coincide when $\ell = j$). We again use conditional expectation, this time obtaining

$$\begin{aligned}\mathbb{E}_\ell\left[T_j^A\right] &= (p_{\ell\to j})\,\mathbb{E}_j\left[T_j^A\right] = \left(\phi(\alpha)^{\ell-j}\right)\left(\frac{\rho\phi(\alpha)(1-(\rho\phi(\alpha)^2)^j)}{\lambda(1-\rho\phi(\alpha)^2)}\right)\\ &= \frac{\rho\phi(\alpha)^{\ell-j+1}(1-(\rho\phi(\alpha)^2)^j)}{\lambda(1-\rho\phi(\alpha)^2)},\end{aligned}$$

which completes the proof of the claim. Note that we have obtained $\mathbb{E}_j\left[T_j^A\right]$ by substituting $\ell = j$ into the expression for $\mathbb{E}_\ell\left[T_j^A\right]$, which we found for $\ell \leq j$, and we have also used the fact from Lemma 2.9 that $p_{\ell\to j} = \phi(\alpha)^{\ell-j}$ whenever $\ell \geq j$. □

Finally, we use Lemma 2.10 to prove Theorem 2.2.

Theorem 2.2. *For any class $\mathbb{M}$ Markov chain, if $\lambda_m, \mu_m > 0$ and $\ell, j \geq j_0 + 1$, we have*

$$\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = \begin{cases}\Omega_m r_m^{j-\ell}\left(1-(r_m\phi_m(\alpha_m))^{\ell-j_0}\right) & \text{if } \ell \leq j\\ \Omega_m\phi_m(\alpha_m)^{\ell-j}\left(1-(r_m\phi_m(\alpha_m))^{j-j_0}\right) & \text{if } \ell \geq j.\end{cases} \tag{2.10}$$

Proof. Observe that the time spent in state (m, j) before leaving phase m, given initial state (m, ℓ) in a class $\mathbb{M}$ Markov chain with $\lambda_m, \mu_m > 0$ is stochastically identical to the time spent in state $j - j_0$ before reaching state 0, given initial state $\ell - j_0$ in an M/M/1/clearing model with arrival, departure, and clearing rates λ_m, μ_m, and α_m, respectively. That is,

$$\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = \mathbb{E}_\ell\left[T_j^A\right],$$

where the quantity on the left-hand side is associated with the class $\mathbb{M}$ Markov chain, and the quantity on the right-hand side with the M/M/1/clearing model Markov chain (with the appropriate transition rate parameters and $A = \{1, 2, 3, \ldots\}$).

We proceed to complete the proof by applying Lemma 2.10. Recall when $\lambda_m, \mu_m > 0$, we have the notation $\rho_m = \lambda_m/\mu_m$, $r_m = \rho_m\phi_m(\alpha_m)$, and $\Omega_m = r_m/(\lambda_m(1 - r_m\phi_m(\alpha_m)))$. Applying Lemma 2.10 when $\ell \leq j$ yields

$$\begin{aligned}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] &= \frac{(\rho_m\phi_m(\alpha_m))^{(j-j_0)-(\ell-j_0)+1}\left(1-(\rho_m\phi_m(\alpha_m)^2)^{\ell-j_0}\right)}{\lambda_m(1-\rho_m\phi_m(\alpha_m)^2)}\\ &= \frac{r_m^{j-\ell+1}\left(1-(r_m\phi_m(\alpha_m))^{\ell-j_0}\right)}{\lambda_m(1-r_m\phi_m(\alpha_m))}\\ &= \Omega_m r_m^{j-\ell}\left(1-(r_m\phi_m(\alpha_m))^{\ell-j_0}\right),\end{aligned}$$

while when $\ell \geq j$, Lemma 2.10 yields

$$\begin{aligned}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] &= \frac{\rho_m\phi_m(\alpha_m)^{(\ell-j_0)-(j-j_0)+1}\left(1-(\rho_m\phi_m(\alpha_m)^2)^{j-j_0}\right)}{\lambda_m(1-\rho_m\phi_m(\alpha_m)^2)}\\ &= \frac{r_m\phi_m(\alpha_m)^{\ell-j}\left(1-(r_m\phi_m(\alpha_m))^{j-j_0}\right)}{\lambda_m(1-r_m\phi_m(\alpha_m))}\\ &= \Omega_m\phi_m(\alpha_m)^{\ell-j}\left(1-(r_m\phi_m(\alpha_m))^{j-j_0}\right),\end{aligned}$$

as claimed. □

2.5 Extending the scope of the CAP Method

In this section we briefly touch upon ways in which the CAP method can be extended beyond class $\mathbb{M}$ Markov chains.

2.5.1 Chains with "catastrophes"

Recall that the M/M/1 clearing model is used to model a system where there can be a catastrophe from any nonzero state causing an immediate transition to state 0. Similarly, we can consider a modification of a class $\mathbb{M}$ Markov chain where from any state (m, j) with $j \geq j_0 + 1$, a catastrophe can occur taking one to state $x \in \mathcal{N}$ with rate $\alpha_m\langle x\rangle \equiv q((m, j), x)$.[4] That is, each phase can have several catastrophe rates, one for each state in the non-repeating portion. In this case, it will be useful to redefine α_m as follows:

$$\alpha_m \equiv \sum_{x\in\mathcal{N}} \alpha_m\langle x\rangle + \sum_{i=m+1}^{M} \sum_{\Delta=-1}^{1} \alpha_m\langle i-m; \Delta\rangle.$$

The CAP method can easily be modified to give limiting probabilities for these types of Markov chains.

2.5.2 Skipping levels when transitioning between phases

Although the assumption that transitions from state (m, j) to state (m, ℓ) can only occur only if $\ell = j \pm 1$ is essential to the CAP method, the assumption that transitions from state (m, j) to state (i, ℓ) (where $i > m$) can only occur if $\ell = j \pm 1$ is much less important. That is, the CAP method may be extended to allow for nonzero transition rates of the form $\alpha_m\langle\Delta_1; \Delta_2\rangle$ with $d \leq \Delta_2 \leq D$ for some $d, D \in \mathbb{Z}$. However, it is advisable to treat the levels $L_{j_0}, L_{j_0+1}, \ldots, L_{j_0+\max\{|d|,|D|\}-1}$ as special cases, just as L_{j_0} was treated as a special case in the analysis presented throughout this chapter.

2.5.3 Chains with an infinite number of phases

Consider a chain with the structure of a class $\mathbb{M}$ chain, except with infinitely many *phases* (i.e., $m \in \{0, 1, 2, \ldots\}$), and a possibly infinite non-repeating portion, $\mathcal{N}$. The CAP method may be used to determine the $\{c_{m,k}\}_{0\leq k\leq m}$ values in terms of $\{\pi_x\}_{x\in\mathcal{N}}$ for the first K phases by solving a system of at most $O(K^2)$ equations. This is because the CAP method provides recurrences such that each $\{c_{m,k}\}_{0\leq k\leq m}$ value can be expressed in terms of $\{c_{i,k}\}_{0\leq k\leq i\leq m-1}$ values; that is, only information about lower-numbered phases (and the non-repeating portion) is needed to compute each $c_{m,k}$. We can first express such values for phase $m = 0$, then phase $m = 1$, and so on. Once these values—along with the easily determined corresponding base terms—have been obtained, we can use the CAP method to find the limiting probabilities for all states in the first K phases as long as we know the $\{\pi_x\}_{x\in\mathcal{N}}$ values.

[4]Whether or not catastrophes can also occur in states (m, j_0) will not change the analysis as arbitrary transitions from states (m, j_0) to states $x \in \mathcal{N}$ are already allowed in class $\mathbb{M}$ Markov chains.

Such a procedure is typically not useful, as the $\{\pi_x\}_{x\in\mathcal{N}}$ values are usually determined via the normalization constraint, which requires expressing limiting probabilities, $\pi_{(m,j)}$, in terms of $\{\pi_x\}_{x\in\mathcal{N}}$ for *all* phases, rather than for only the first K phases. However, there are settings where sufficient information about the structure of $\{\pi_x\}_{x\in\mathcal{N}}$ may be obtained via other analytic approaches, allowing for the CAP method to compute the limiting probability of the first K phases (where K can be as high as desired, subject to computational constraints). For example, a two-class priority queue can be modeled by an infinite phase variant of a class $\mathbb{M}$ Markov chain. In that setting, queueing-theoretic analysis provides sufficient information about the structure of the limiting probabilities in the non-repeating portion (see [118]), making the CAP method a useful tool for that problem.

2.6 Conclusion

This chapter presents a study of the stationary distribution of quasi-birth-death (QBD) continuous time Markov chains in class $\mathbb{M}$. Class $\mathbb{M}$ Markov chains are ergodic chains consisting of a finite nonrepeating portion and an infinite repeating portion. The repeating portion of a class $\mathbb{M}$ chain consists of an infinite number of levels and a finite number of phases. Moreover, transitions in such chains are *skip-free in level*, in that one can only transition between consecutive levels, and *unidirectional in phase*, in that one can only transition from lower-numbered phases to higher-numbered phases. Despite these restrictions, class $\mathbb{M}$ Markov chains are used extensively in modeling computing, service, and manufacturing systems, as they allow for keeping track of both the number of jobs in a system (via levels), and the state of the server(s) and/or the arrival process to the system (via phases).

This chapter develops and introduces a novel technique, Clearing Analysis on Phases (CAP), for determining the limiting probabilities of class $\mathbb{M}$ chains exactly. This method proceeds iteratively among the phases, by first determining the form of the limiting probabilities of the states in phase 0, then proceeding to do the same for the states in phase 1, and so on. As suggested by its name, the CAP method uses clearing model analysis to determine the structure of the limiting probabilities in each phase.

Unlike most existing techniques for solving QBDs, which rely upon the matrix-geometric approach, the CAP method avoids the task of finding the complete rate matrix, $\mathbf{R}$, entirely. Instead, the CAP method yields the limiting probabilities of each state, (m, j), in the repeating portion of the Markov chain as a linear combination of scalar *base terms* (with weights dependent on the phase, m), each raised to a power corresponding to the level, j. These *base terms* turn out to be the diagonal elements of the rate matrix, $\mathbf{R}$. The weights of these linear combinations can be determined by solving a finite system of linear equations. We also observe that the structure of the weights of these linear combinations can depend on the multiplicity structure of the base terms.

The CAP method can be applied to Markov chains beyond those in class $\mathbb{M}$, as discussed in Section 2.5. For example, the CAP method can be used to determine limiting probabilities in chains where one or more phases allow for immediate "catastrophe" transitions to states in the non-repeating portion. As another example, the CAP method can also be applied to Markov chains where transitions between phases can be accompanied with a change in level exceeding

1. The CAP method can also be used to study some chains with an infinite number of phases. There is ample room for future work to extend the CAP method in a variety of directions.

The CAP method and the solution form it provides offer several impactful advantages. First, while many existing methods for determining the limiting probabilities of QBDs exploit the relationship between successive *levels*, the CAP method exploits the relationship between successive *phases*, thereby offering complementary probabilistic intuition on the structure and steady state behavior of class $\mathbb{M}$ Markov chains. This method also provides an additional tool for practitioners who are studying systems that can be modeled by class $\mathbb{M}$ Markov chains. Depending on the application domain, the scalar solution form of the CAP method may have advantages over other solution forms for computing certain metrics of interest (e.g., mean values, higher moments, tail probabilities, etc.). While this chapter does not cover using the solution of the CAP method to derive metrics of interest, as such metrics are often application specific, we hope that future work can find novel uses for the CAP method in a variety of settings.

Chapter 3

The Malware Cleanup Problem

3.1 Introduction

Cybercrime is an increasingly costly problem for individuals, corporations, and governments alike. The total cost of cybercrime in 2014 is estimated at \$375–\$575 billion worldwide, with the average country experiencing cyber crime costs amounting to 0.5% of its GDP (see [29]). In this chapter, we focus in particular on cyber attacks with *persistent effects* that target an online service provider. The effects of such attacks do not cease until some cleanup actions are undertaken by the service provider. These attacks often take the form of *malware* (malicious software), including viruses, code injections, worms, trojan horses, etc. For simplicity, throughout we will refer to these persistent attacks as malware.

Malware attacks are concerning primarily for two reasons: attacks pose a **security** threat, while also potentially compromising system **performance**; both are costly. For an online service provider, security threats include the direct theft of money as well as data breaches, which expose customers to fraud. These breaches lead to substantial legal expenses, tarnish the service provider's reputation, and where applicable, negatively affect the parent company's stock price [58]. We refer to these combined monetary losses as *security losses*. Meanwhile, malware also often leads to performance degradation [80], which effectively reduces the rate at which a service provider can serve its customers' requests (hereafter, *jobs*). Service rate reductions lead to lengthier response times, create an inferior service experience for the customers, and reduce their willingness to pay for the service. The removal of malware, although necessary, leads to additional performance costs: reboots and lengthier cleanup procedures require temporarily taking the system offline, causing discarded jobs and downtimes [21, 41, 99]. Malware infections also often act as a scaffolding for even more serious malware attacks. Malware infections often develop in stages that grow worse over time, and hence security and performance costs can increase sharply if infections are left unaddressed [24, 79]. More severe malware can also take longer to remove, leading to even longer downtimes during cleanup procedures [121].

Consider a system administrator that is providing an online service. At a certain point in time, she receives an automated warning that her server is currently overloaded. This could be a consequence of (i) a traffic spike, (ii) a software bug, or (iii) a security breach that consumes system resources. The system administrator can take several actions: she could do nothing, which could have negative effects on user experience, user privacy, and ultimately revenues. Alternatively, she could reboot the system, at the cost of existing clients, with some chance of resolving the issue. Or she could take the system down, investigate what went wrong, and take time-consuming steps to maximize the likelihood that the issue is resolved. These dilemmas are faced by system administrators on a regular basis. System administrators are not the only ones that face such security-availability tradeoffs; similar dilemmas exist in every deployed system where resources are limited [17] and figuring the right balance is currently an open research question (see for example, [38]).

In this chapter we investigate how an online service provider should respond to observable changes in the system—whether *direct evidence* of a malware infection or performance degradation that *merely suggests the possibility* of an infection—in order to maximize revenue after accounting for security losses. In essence, we develop a mathematical model that addresses the question of "what level of threat necessitates a response?" At a first glance, it may appear that removing malware as early as possible is always revenue-optimal: early cleanups reduce average security costs, minimize performance degradation, and lead to less time-consuming cleanup times. However, frequent cleanup procedures can be detrimental, as they can increase the frequency at which jobs are discarded and lead to more downtime (even if individual downtimes are shorter).

We quantify this tradeoff by presenting a stylized, but detailed Markovian stochastic model of the service provider's operations and vulnerability to malware. Customers wait in a queue for service and pay a price that depends on the system's historic average response time. Over time, the system becomes infected by malware in *stages*. Each successive malware state causes greater security losses, further slows down service, and takes longer to clean up. Undertaking a cleanup action requires discarding all jobs currently in the system (the customers are compensated), and taking the system offline for a prolonged (random) period of time, before resuming service. Using exact stochastic analysis, we quantify the revenue rate associated with cleaning up the system at each stage and determine the optimal cleanup policy.

In reality, malware is often difficult to detect, so we also consider the case where performance degradation can occur due to reasons other than malware, and only the service rate is visible to the service provider; the malware state is hidden. By observing the service rate, the service provider can infer probabilistic beliefs regarding malware infection and take cleanup actions based on *the duration of time spent in the current performance degradation state*.

Both the visible and hidden malware models provide an analytic challenge. In order to quantify the revenue rate under each potential policy, we must understand not only the relative proportion of time spent in each malware state, but also the mean number of jobs in the system. We determine this quantity by finding the exact limiting probability distribution of a continuous time Markov chain that simultaneously tracks the number of jobs in the system (an unbounded quantity) and the system's current malware state. This is a *quasi-brith-death process* Markov chain with a two-dimensional state space that is infinite in one dimension. While chains of this form are notoriously difficult to analyze, we employ a novel adaptation of the Clearing Analysis on Phases (CAP) methodology—developed in Chapter 2—to obtain the exact limiting probability distribution, by which we obtain exact revenues under various cleanup policies.

While problems closely related to malware cleanup have been addressed in the literature (see, for example, [79]), to date work in this area has focused on heuristic solutions. We are the first to introduce and solve a mathematical model for the malware cleanup problem. Aspects of our model resemble models studied in the literature on the machine interference problem and condition-based maintenance, but with several distinguishing features (see Section 3.2 for details).

Our primary contributions are four-fold: (i) on the modeling front, we provide the first stochastic model for determining when an online service provider should perform cleanups; (ii) on the theoretical front, we derive the revenue rate for various cleanup policies in closed form – this requires solving a complex quasi-birth-death process, which requires first developing an adaptation of the CAP method; (iii) on the practical front, we provide a decision tool that allows practitioners to evaluate optimal cleanup policies for their systems; (iv) on the case study front, we use our decision tool to highlight some interesting cases and insights by studying parameter sets provided by the security company ForAllSecure, Inc.

The remainder of the chapter is structured as follows. In Section 3.2, we review the related literature. In Sections 3.3 and 3.4 we present, solve, and show results for the visible and hidden malware models, respectively. In Section 3.5, we highlight the importance of exact analysis by comparing the effectiveness of our technique to a technique making use of a simplifying approximation. We conclude in Section 3.6.

3.2 Literature Review

To date, problems very closely related to the malware cleanup problem have been addressed in the computer security literature only in terms of heuristic approaches. A notable example is in the work of [79], which proposes a policy where a subset of the servers being used by service provider are rotated out for cleaning, while ensuring that enough servers are running at any given time. This policy is not a result of stochastic analysis. By contrast, our model assumes a single server system, or a system where the servers work together and are prone to becoming simultaneously compromised (e.g., due to unknown exploitable bugs that are common to all servers).

The bulk of the literature on the stochastic analysis of malware and intrusions focuses on either malware propagation (see for example, [56]) or intrusion detection (see for example, [132]). Neither stream of work is applicable to our problem, where the focus is on maintaining and removing potential infections from a single vulnerable system.

Although the application is different, the literature on the repair and maintenance of machines is analytically related to our work. We briefly review the related literature on machine interference problems and condition-based maintenance in Sections 3.2.1 and 3.2.2, respectively. Finally, in Section 3.2.3, we provide a little background on the CAP method for solving quasi-birth-death chains, and we explain why alternative analytical methods in the literature are not suitable for our problem.

3.2.1 Machine interference problems

The classical *machine interference problem*, also known as the *machine repairman problem* (see [67] for a survey), features n machines and r workers. Machines occasionally break down, and a worker can spend some time with a machine and restore it. If the number of machines that have broken down at any point exceeds the number of workers r, then some machines will have to wait until they go into repair (e.g., the machines interfere with one another). When machines exhibit heterogeneity, the problem is to identify which machines should be repaired in any given situation.

In contrast, our problem involves only a single server (machine), or equivalently a set of servers that work together and become compromised together. Many successful cyber attacks involve exploiting a fundamental software-level problem, and therefore an entire system might become infected, or otherwise degrade in performance, at the same time. Therefore, the malware cleanup problem asks the question of *when* to repair a system, rather than *which* of many subsystems (machines) to repair. Nonetheless, there is a small body of work within this stream of literature which, like our work, features potentially unbounded queues. The analysis of problems with multiple machines subject to breakdowns often requires the analysis of a quasi-birth-death process Markov chain. Our Markov chain has certain features, however, that make it distinct from the works cited below. For example, our Markov chain features infinite collections of states which transition directly to one of several states in the finite "non-repeating portion" of the Markov chain, corresponding to the fact that all customer requests (jobs) must be discarded when a cleanup procedure is initiated.

In [30], the machines are servers in a multi-server queueing system, each contributing its own service rate to serving the jobs, and each with its own breakdown and repair rates. The goal is to repair servers in such a manner than the total service rate is maximized. The analysis of this problem requires solving for the limiting probability distribution of a quasi-birth-death process Markov chain. In general, the chains required for this problem do not have closed-form solutions, as the solutions require solving higher-order polynomials. The authors use the spectral expansion method to obtain numerical solutions in this setting. A similar model is considered in [128], with each server serving its own parallel queue.

In [44], the machines play the role of jobs in a single server queueing system, coming from two classes, one from a finite population (forming a closed system), and the other from an infinite population (forming an open system), with the infinite population having priority over the finite population. The descriptive analysis of this model requires solving for the limiting probability distribution of a quasi-birth-death process Markov chain, falling within the category of chains that can be solved via the CAP method. The limiting probability distribution is determined explicitly using eigenvalues, which the CAP method invokes implicitly.

3.2.2 Condition-based maintenance

Condition-based maintenance problems ask *when* one or more failure-prone components of a system should be repaired. These components will degrade in performance over time. One must determine at which condition level one or more machines must be repaired. There is a natural tradeoff between repairing components early and often (preventative maintenance) and repairing components only when necessary (corrective maintenance). The former can yield more frequent downtimes, while the latter leads to more severely degraded components and "unplanned downtime." For a recent overview of work in this area, see the Ph.D. thesis [133].

The malware cleanup problem proposed in this paper can be thought of as a type of condition-based maintenance problem. The fundamental difference between our problem and the literature in this domain is that we consider a single component system, where the component serves jobs in a potentially unbounded queue. Therefore, our problem has an infinite state space, whereas the work in this area primarily focuses on finite state spaces. Maintenance actions (cleanup procedures) require the removal of all jobs in the queue, triggering an additional loss of revenue. We can avoid tracking queue in our model, and instead resort to steady-state analysis at each performance state, but this leads to an approximation that is often woefully imprecise (see Section 3.5).

Prior work, however, has considered condition-based maintenance in the presence of hidden variables. For example, in [23] decisions and inferences are made in a condition-based maintenance setting using a Hidden Markov Model (HMM), while [96] use a modified HMM where the failure state is completely observable. In [102], an optimal stopping time framework is used to approach a similar problem. Modeling differences and complexities that arise in our revenue rate function, due to dealing with an infinite state space problem, make the methods used in these papers unsuitable for our setting.

3.2.3 Methods for solving quasi-birth-death process Markov chains

Quasi-birth-death (QBD) processes are used in modeling a variety of phenomena. Despite the difficulties associated with determining the limiting probability distributions of such Markov chains, a variety of techniques are available for analyzing various subclasses of QBDs, with the matrix-geometric methods being the most common (see the surveys in [105] and [93]). These methods are typically implemented numerically and do not generally allow for closed-form solutions.

Within the matrix-geometric stream of literature, there are special classes of Markov chains that allow for closed-form solutions. One such method, given in [124], can be used to solve for the limiting probability distribution of the chain describing our *visible* malware model, but is not directly applicable to our *hidden* malware model, as phase transitions in the *hidden* case exhibit a *directed acyclic graph* (DAG) structure, whereas the method in [124] is presented in the context of *tree-like* phase transition structures.

Additional methods exist for solving QBD Markov chains in closed form. For example, in [5], a general method is given for finding explicit solutions to a large class of chains, but this method is again not directly applicable to chains exhibiting a DAG phase transition structure. Another solution method is Recursive Renewal Reward (RRR) (see [53] and [54]). While RRR can find closed-form solutions for chains with a DAG phase transition structure, it only provides means and transforms, rather than complete distributions.

In this paper we employ the Clearing Analysis on Phases (CAP) method, first introduced in [43]. The CAP method yields exact closed-form solutions for the chain's entire limiting probability distribution by viewing each phase of a QBD as acting like an M/M/1 model with clearing events. This makes the CAP method a natural fit for our model, as it features clearing events (correspond to the start of a cleanup procedure, which requires discarding all jobs). To the best of our knowledge, this is the first applied work which makes use of the CAP method.

3.3 The Case of Visible Malware

When a host becomes infected by malware, there are two possibilities: (i) the system administrator knows of the infection and needs to make a decision regarding whether a cleanup process should be initiated and (ii) the system administrator does not know that the host has been infected. In this section, we deal with the first case, where malware infections are always visible. While this case is certainly simpler, it is of interest for two reasons: (i) it provides a stylized simplification of the malware problem, which will lend itself to easier analysis and interpretation and (ii) it is an appropriate model for "unsophisticated malware," which serves as more of a nuisance, than a real threat (i.e., malware that does not pose serious security threats, but still leads to considerable performance degradation and service disruptions).

3.3.1 Visible Malware Model

In the visible malware model, the system can be in one of four states depending on the level of malware, if any, that the system is infected with: **normal** (uninfected), **bad**, **worse**, and **dead**. In each successive state, (i) the security cost rises, (ii) the performance level, or service rate, becomes slower, and (iii) the amount of time necessary to clean the malware increases. For tractability, all transition rates in this system are Markovian (i.e., memoryless).

Queueing model.

We consider a revenue-maximizing online service provider, facing a stream of incoming customers. Customer requests (hereafter, *jobs*) arrive to a first-come-first-serve queue according to a Poisson process with rate λ, and are served by a single server[1] at some service rate μ (i.e., it takes an average of $1/\mu$ seconds to serve a job), which can change due to the state of the system; service times are exponentially distributed. We let T be the response time experienced by a job (i.e., T is the time elapsed from when the job first arrives until it either receives service or it is discarded). We assume that the steady-state average response time, $\mathbb{E}[T]$—which will be based on the service provider's decisions—is known to both the service provider and the customers (e.g., via historically available data).

Visible malware evolution.

The server can become infected by malware in stages. Initially, the system is in the **normal** state, but will become infected by **bad** malware after an amount of time that is exponentially distributed with rate α_{bad} (i.e., on average such a transition takes $1/\alpha_{\text{bad}}$ units of time to occur). Similarly, there is a transition from **bad** to **worse** with rate α_{worse} and from **worse** to **dead** with rate α_{dead}. We restrict attention to the case where malware infections occur *in sequential stages*, and no stage can be skipped.[2] The model's accuracy can be increased by considering more than

[1]The single-server assumption is for simplicity; our methodology can accommodate multiple servers.

[2]Our methodology can accommodate skipping malware states, and can even allow for multiple types of malware that can simultaneously infect the system. The sequential escalation formulation of malware simplifies the exposition of the base technique and captures the real-world phenomena where "tamer" malware that is more likely to go

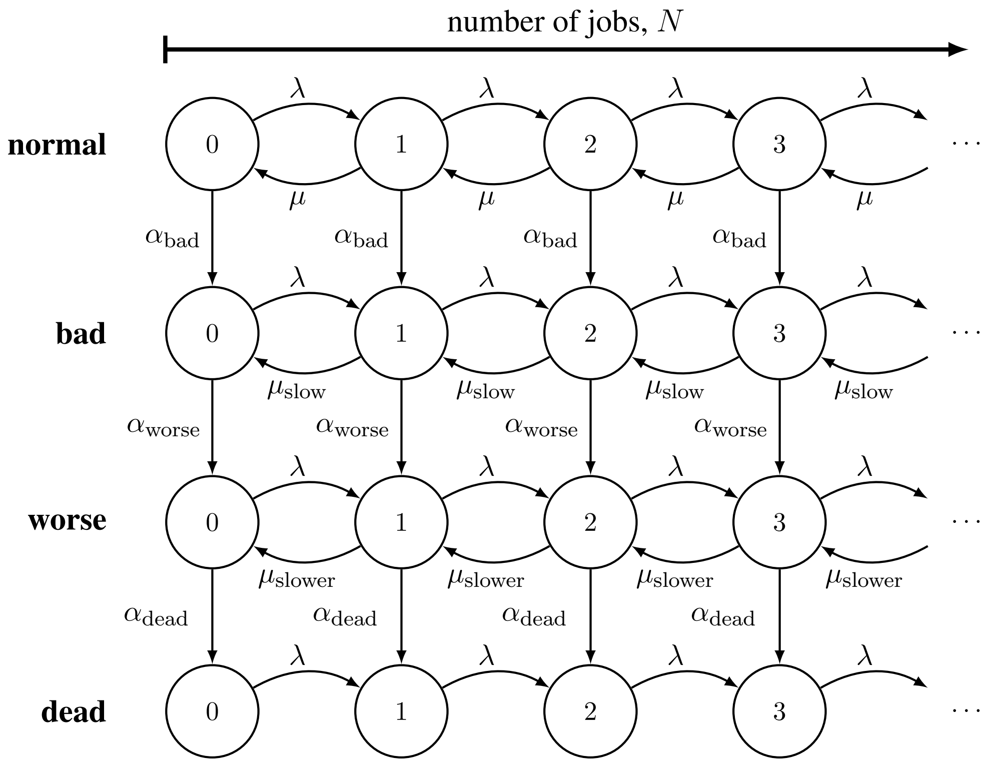


Figure 3.1: The continuous time Markov chain (CTMC) tracking the malware state and the number of jobs in the system, assuming no system administration. Transitions across malware states preserve the number of jobs. The service rate drops with each successive malware state. Note that since there are no cleanups, this chain is *not* ergodic.

these four states, but we maintain that this number of states allows for an appropriate level of detail, given our aim of providing a stylized model.

Malware causes the service provider to incur losses (e.g., monetary theft, data theft, loss of good will, etc.) due to having an insecure system. The average rate (in dollars per second) at which the server incurs losses is 0, ℓ_{bad}, ℓ_{worse}, and ℓ_{dead}, in the **normal**, **bad**, **worse**, and **dead** states, respectively, where $0 \leq \ell_{\text{bad}} \leq \ell_{\text{worse}} \leq \ell_{\text{dead}}$. Malware also causes a decrease in the service rate: the service rate is μ_{fast}, μ_{slow}, μ_{slower}, and $\mu_{\text{dead}} = 0$ in the **normal**, **bad**, **worse**, and **dead** states, respectively, where $\mu_{\text{fast}} \geq \mu_{\text{slow}} \geq \mu_{\text{slower}} > \mu_{\text{dead}} = 0$.[3]

The Markovian transition structures allow us to model the entire system as a continuous time Markov chain (CTMC) tracking the malware state of the system along with the number of jobs in the system. The CTMC for our model, assuming no cleanup events, is shown in Figure 3.1. Note that without cleanup events, the resulting Markov chain is non-ergodic, as the number of jobs will grow without bound once we reach the **dead** state. We next introduce the cleanup policies

unnoticed is used as a scaffolding to deploy more disruptive software.

[3]Being in the **dead** state means that the service provider (and the customers) are unable to use the system, and so the service rate has dropped to zero. Moreover, we assume that in all non-**dead** states, the system is stable: that is $\mu_{\text{fast}}, \mu_{\text{slow}}, \mu_{\text{slower}} > \lambda$.

that are available to us, which will lead to modifications of this chain that will make the chain ergodic.

Visible malware cleanup.

A system infected by malware can be restored to full speed by a cleanup procedure during which all existing jobs are discarded (refused service and refunded q), and the system stops admitting jobs for a random duration of time. When the cleanup is complete, the system is restored to the **normal** state and resumes accepting jobs. The length of time devoted to the cleanup procedure depends on the current malware state, as more severe malware takes longer to remove. The cleanup procedure lasts an amount of time that is exponentially distributed with rate β_{bad}, β_{worse}, or β_{dead} (i.e., average length $1/\beta_{\text{bad}}$, $1/\beta_{\text{worse}}$, or $1/\beta_{\text{dead}}$) when initiated in the **bad**, **worse**, or **dead** states, respectively.[4] The more severe the malware, the more time consuming it is to remove (e.g., a simple reboot is not enough to restore a **dead** system back to **normal**). For simplicity, we assume that there are no security losses during the cleanup procedure.[5]

In the setting with visible malware, we have the three following natural cleanup policies that the service provider can employ to maintain their system:

- **clean@bad**: Clean once the system transitions to the **bad** malware state.
- **clean@worse**: Clean once the system transitions to the **worse** malware state.
- **clean@dead**: Clean once the system transitions to the **dead** malware state.

Each of these policies initiates a cleanup procedure *only* when the target malware state is reached. More sophisticated policies are studied when we address hidden malware in Section 3.4.

Pricing and revenue model.

We assume that each customer is willing to pay a price

$$p \equiv q - c \cdot \mathbb{E}[T], \tag{3.1}$$

in order for her job to be served, where

- q is the hypothetical price that customers would pay (in dollars) for a zero delay service,
- c is a parameter capturing customer delay sensitivities (in dollars per second),
- and $\mathbb{E}[T]$ is the expected response time (in seconds).

Since the service provider seeks to maximize revenue, the provider charges each customer the full amount, p, upon arrival. Should a job be discarded (either while waiting in the queue or while in service), the customer is refunded q (i.e., she pays nothing and keeps $q - p = c \cdot \mathbb{E}[T]$ as

[4]We note that the cleanup durations can actually take on any other distribution (e.g., deterministic or uniform). The analysis and results will remain unchanged, as long as the mean cleanup duration is given by $1/\beta_{\text{bad}}$, $1/\beta_{\text{worse}}$, and $1/\beta_{\text{dead}}$ when initiated in the **bad**, **worse**, and **dead** state, respectively.

[5]Our model can accommodate the case where there *are* security losses during the cleanup procedure without additional difficulty. From a modeling perspective, security losses during the cleanup procedure could be associated with good-will losses due to the service being down, tarnishing the reputation of the service provider.

compensation for waiting some time before being informed that her job will not be served) and the firm incurs a revenue loss of $c \cdot \mathbb{E}[T]$ for that customer.[6]

The objective of the service provider is to implement a malware cleanup policy so as to maximize the rate at which it earns revenue; we can express the service provider's revenue rate by observing that the service provider earns q for each customer that completes service, while incurring a loss of c per second for each job in the system (whether or not that customer is ultimately served; recall that customers whose jobs are not served are still compensated $c \cdot \mathbb{E}[T]$), and potentially incurring additional security losses each second. Hence, the server's revenue rate is

$$\mathcal{R} \equiv q\chi - c \cdot \mathbb{E}[N] - \mathcal{L}, \tag{3.2}$$

where q and c are as previously defined and

- χ is the system's throughput, or average rate at which jobs are served,
- $\mathbb{E}[N]$ is the steady-state expected number of jobs in the system, and
- $\mathcal{L}$ is the average rate at which the service provider incurs security losses.

We shall see that all three of χ, $\mathbb{E}[N]$, and $\mathcal{L}$ will depend on the service provider's decisions for combating malware and maximizing revenue.

3.3.2 Visible Malware Analysis

In order to analyze $\mathcal{R}$, it is helpful to define some additional notation. We define π_{normal}, π_{bad}, π_{worse}, and π_{dead} to be the limiting probability associated with the (i.e., the long-run steady-state proportion of time spent in) **normal**, **bad**, **worse**, and **dead** states (ignoring time spent in cleanup procedures), respectively, under any of the policies of interest. From the definition of our policies, we can immediately observe that $\pi_{\text{bad}} = 0$ under the **clean@bad** policy, $\pi_{\text{worse}} = 0$ under both the **clean@bad** and **clean@worse** policies, while $\pi_{\text{dead}} = 0$ under all policies. Recalling that ℓ_{bad}, ℓ_{worse}, and ℓ_{dead} are the average rates at which the server incurs losses in the **bad**, **worse**, and **dead** states, respectively, the overall average security loss rate, $\mathcal{L}$, may be reformulated as

$$\mathcal{L} = \ell_{\text{bad}}\pi_{\text{bad}} + \ell_{\text{worse}}\pi_{\text{worse}} + \ell_{\text{dead}}\pi_{\text{dead}}.$$

Using the new notation defined above, we can write $\mathcal{R}$ as follows:

$$\begin{aligned}\mathcal{R} &= q\chi - c \cdot \mathbb{E}[N] - \mathcal{L} \\ &= q\chi - c \cdot \mathbb{E}[N] - (\ell_{\text{bad}}\pi_{\text{bad}} + \ell_{\text{worse}}\pi_{\text{worse}} + \ell_{\text{dead}}\pi_{\text{dead}}) \\ &= q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}},\end{aligned}$$

establishing the fact that ℓ_{dead} is irrelevant to the revenue rate under our assumption that a system will *never* be permitted to persist in the **dead** state under any policy. In order to determine $\mathcal{R}$ for each of the policies of interest, it remains to determine χ, $\mathbb{E}[N]$, π_{bad}, and π_{worse}, under these

[6]Equivalently, each customer might be refunded an amount such that the customer nets $c \cdot T$, where T is the *exact* amount of time the customer waited before her job was terminated. Since we are only concerned with the service provider's *expected* revenue rate, this modified assumption would leave the analysis unchanged.

policies. While π_{bad} and π_{dead} can be found by analyzing a finite-state Markov chain tracking only the *malware state* of a system, the exact determination of χ and $\mathbb{E}[N]$ require analyzing a two-dimensional infinite state Markov chain.

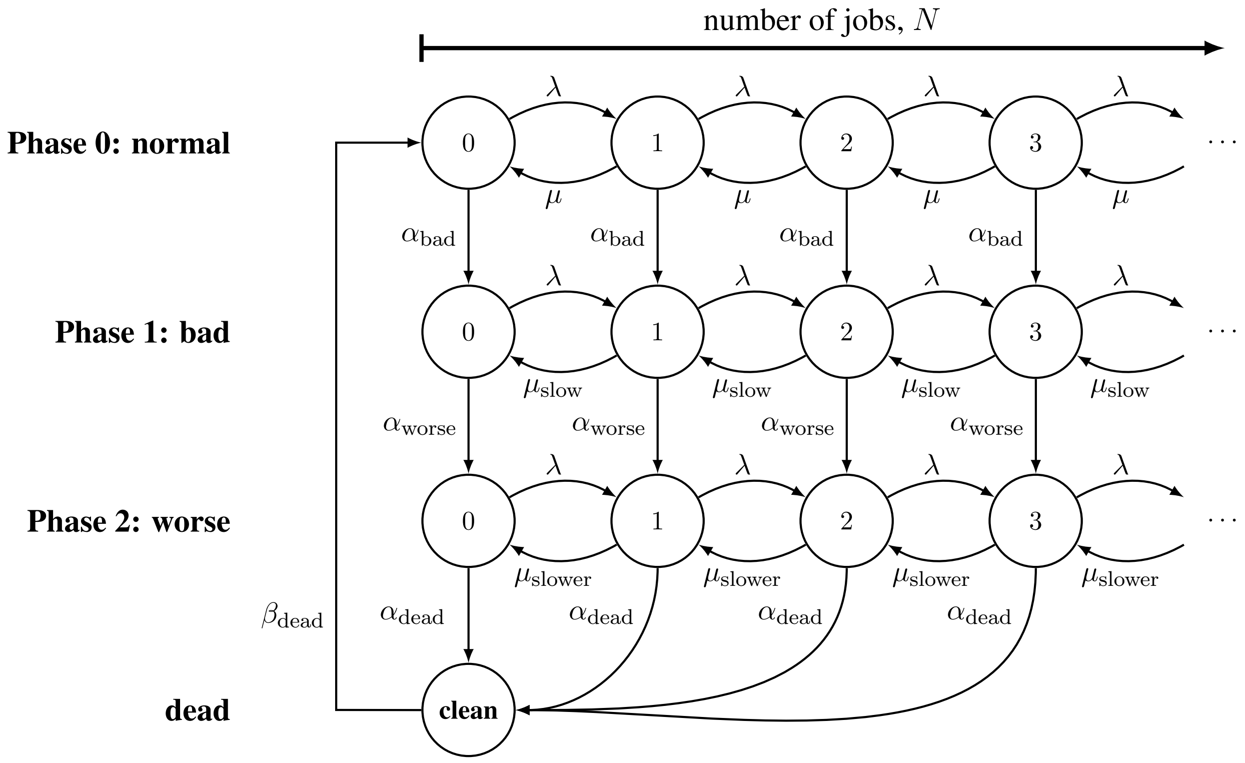


Figure 3.2: The continuous time Markov chain (CTMC) tracking both the malware state and the number of jobs under the **clean@dead** policy. Each *phase* (pictured as a "row" of states) is an infinite collection of states (tracking the number of jobs, N) associated with one of the malware states. Transitions from one phase to another preserve the number of jobs and always move to a higher-numbered phase (pictured as "downward"). Rather than transitioning to another phase, the states in Phase 2 transition to the **clean** state, where all jobs are discarded.

In order to find these quantities, we use the Clearing Analysis on Phases (CAP) method. In the interest of brevity, we primarily restrict attention to analyzing the most complicated policy, **clean@dead**. The analysis of the **clean@dead** policy requires examining more malware states than the the other policies, and so the analysis of the simpler **clean@bad** and **clean@worse** policies follow from straightforward variants of the same approach. For each cleanup policy we view our model as a CTMC with a state space tracking the malware state *and* the number of jobs present in the system. Figure 3.2 shows the CTMC corresponding to the **clean@dead** policy.

Each malware state corresponds to a *phase* of the CTMC, an infinite collection of states that comprise a birth-death process (with arrivals with rate λ and departures with some rate μ, depending on the phase), but with additional transition rates to other phases. For example, in the **clean@dead** CTMC, Phases 0, 1, and 2 correspond to the **normal**, **bad**, and **worse** malware states, respectively. We also introduce the **clean** malware state, which captures when the system is undergoing a cleanup procedure. The **clean** state is a single state, rather than an entire phase, as there are no jobs to track (i.e., $N = 0$) when cleaning.

We call attention to the fact that transitions only exist from lower-numbered phases to higher-numbered phases, which is essential for obtaining exact solutions via the CAP method. The CAP method treats each phase as an M/M/1 *clearing model*—an M/M/1 chain with identical rates from each state leading to a *clearing event*—where *clearing events* either lead to a higher-numbered phase or directly to the **clean** state.

In order to apply the CAP method, it is helpful to introduce the following notation:

- $\pi_{(m,j)}$ is the limiting probability of being in Phase m with $j \geq 0$ jobs in the system,
- π_{clean} is the limiting probability of being in the **clean** state,
- μ_m is the service rate in Phase m (i.e., $\mu_0 = \mu_{\text{fast}}$, $\mu_1 = \mu_{\text{slow}}$, $\mu_2 = \mu_{\text{slower}}$), and
- α_m is the rate at which the system *leaves* Phase m (i.e., $\alpha_0 = \alpha_{\text{bad}}$, $\alpha_1 = \alpha_{\text{worse}}$, $\alpha_2 = \alpha_{\text{dead}}$).

We will solve for limiting probabilities in the following form

$$\pi_{(m,j)} = \sum_{k=0}^{m} a_{m,k} r_k^j,$$

- where r_k is shorthand notation for the following *base term* associated with Phase k:[7]

$$r_k \equiv \frac{\lambda + \mu_k + \alpha_k - \sqrt{(\lambda + \mu_k + \alpha_k)^2}}{2\mu_k};$$

- $a_{m,k}$ is a quantity capturing the relationship between Phases m and k, where $0 \leq k \leq m \leq 2$.

This form is particularly useful as the probability of being in a particular malware state with j jobs in the system is represented as a linear combination of base terms, r_k (one associated with each of the preceding and current malware states), each raised to the power of j, where the $a_{m,k}$ coefficients are the weights of this linear combination. Informally, the $a_{m,k}$ coefficients capture a relationship between Phase m and Phase k. Note that neither $a_{m,k}$, nor r_k depend on j. This form also allows for the convenient computation of $\mathbb{E}[N]$ and χ, via the use of standard infinite series.

Determining the complete limiting distribution of the **clean@dead** CTMC requires only determining the $a_{m,k}$ coefficients, together with π_{clean}. These variables, together with the redundant $\pi_{(m,0)}$ variables, are the solutions to a system of equations, which are a combination of balance equations and relationships which are derived via the CAP method. In [43], the CTMCs of interest do not feature clearing events which return directly to the non-repeating portion (in the case of the **clean@dead** policy, these are the transitions with rate α_{dead} from Phase 2 to the **clean** state), although it is mentioned that the CAP method can be extended to cover such transitions. We adapt the CAP method to allow it to apply to such CTMCs by modifying the balance equation associated with π_{clean} to take into account the additional transitions into π_{clean} and by considering

[7]Our method requires that all r_k be distinct, which is the case for all but a zero measure set of parameter settings (for the application of the CAP method when this is not the case, see [43]).

α_{dead} as a component of α_2, the total outgoing transition rate leaving Phase 2 (in fact, in this case, we simply have $\alpha_{\text{dead}} = a_2$, although this is a simplification that will not necessarily apply in all settings, such as when malware can be hidden). This adaptation of the CAP method yields the following system of equations that must be satisfied by the limiting probability distribution and the $a_{m,j}$ coefficients:

$$\begin{cases} a_{0,0} = \pi_{(0,0)} \\ a_{1,0} = \dfrac{r_0 r_1 \alpha_0 a_{0,0}}{(\lambda - \mu_1 r_0 r_1)(r_0 - r_1)} \\ a_{1,1} = \pi_{(1,0)} - a_{1,0} \\ a_{2,0} = \dfrac{r_0 r_2 \alpha_1 a_{1,0}}{(\lambda - \mu_2 r_0 r_2)(r_0 - r_2)} \\ a_{2,1} = \dfrac{r_1 r_2 \alpha_1 a_{1,1}}{(\lambda - \mu_2 r_1 r_2)(r_1 - r_2)} \\ a_{2,2} = \pi_{(2,0)} - a_{2,0} - a_{2,1} \\ \pi_{(0,0)} = \dfrac{\beta_{\text{dead}} \pi_{\text{clean}} + \mu_0 r_0 a_{0,0}}{\lambda + \alpha_0} \\ \pi_{(1,0)} = \dfrac{\mu_1 (r_0 a_{1,0} + r_1 a_{1,1}) + \alpha_0 \pi_{(0,0)}}{\lambda + \alpha_1} \\ \pi_{(2,0)} = \dfrac{\mu_2 (r_0 a_{2,0} + r_1 a_{2,1} + r_2 a_{2,2}) + \alpha_1 \pi_{(1,0)}}{\lambda + \alpha_2} \\ \pi_{\text{clean}} = \dfrac{\alpha_2}{\beta_{\text{dead}}} \left(\dfrac{a_{2,0}}{1 - r_0} + \dfrac{a_{2,1}}{1 - r_1} + \dfrac{a_{2,2}}{1 - r_2} \right) \\ 1 = \pi_{\text{clean}} + \dfrac{a_{0,0} + a_{1,0} + a_{2,0}}{1 - r_0} + \dfrac{a_{1,1} + a_{2,1}}{1 - r_1} + \dfrac{a_{2,2}}{1 - r_2}. \end{cases}$$

Closed-form solutions for the limiting probability distribution of the CTMC can be obtained by symbolically solving the system above, although the resulting expressions will not be concise. Most importantly, the solutions will be exact, rather than approximations. With the limiting probabilities determined in a convenient (if not concise) form, we can compute $\mathbb{E}[N]$ and χ in terms of π_{clean} and the $a_{m,k}$ coefficients. Starting with $\mathbb{E}[N]$, we use straightforward sum identities to find that

$$\begin{aligned}
\mathbb{E}[N] &= \sum_{j=0}^{\infty} j \cdot \mathbb{P}(N = j) \\
&= \sum_{j=0}^{\infty} j(\pi_{(0,j)} + \pi_{(1,j)} + \pi_{(2,j)}) \\
&= \sum_{j=0}^{\infty} j\left((a_{0,0}r_0^j) + (a_{1,0}r_0^j + a_{1,1}r_1^j) + (a_{2,0}r_0^j + a_{2,1}r_1^j + a_{2,2}r_2^j)\right) \\
&= \sum_{j=0}^{\infty} \left((a_{0,0} + a_{1,0} + a_{2,0})jr_0^j + (a_{1,1} + a_{2,1})jr_1^j + (a_{2,2})jr_2^j\right) \\
&= \frac{(a_{0,0} + a_{1,0} + a_{2,0})r_0}{(1 - r_0)^2} + \frac{(a_{1,1} + a_{2,1})r_1}{(1 - r_1)^2} + \frac{a_{2,2}r_2}{(1 - r_2)^2}.
\end{aligned}$$

Next, we must compute χ, the rate at which jobs are served, *excluding* those jobs that are *discarded* from service, due to being in the system at the start of a cleanup procedure. We observe that every job that enters the system is eventually either served or discarded. Moreover, jobs enter the system only when the system is not in the **clean** state, so the long run average arrival rate is $\lambda(1 - \pi_{\text{clean}})$. Hence, if η is the rate at which jobs are discarded from the system, then $\chi = \lambda(1 - \pi_{\text{clean}}) - \eta$: the rate at which jobs are served is the rate at which jobs enter the system less the rate at which jobs are discarded.

In order to compute χ, we must compute η: we observe that since jobs are only discarded when the system transitions to **clean**, η is simply the rate at which the system transitions to **clean** multiplied by the expected number of jobs in the system at that time. Under the **clean@dead** policy, this rate is $\alpha_{\text{dead}} \cdot \pi_{\text{worse}}$: the product of the probability that the system is in the **worse** state and the rate with which the system transitions from **worse** to **clean**. Meanwhile, the expected number of jobs at the time of clearing is $\mathbb{E}[N|\textbf{worse}]$, the expected number of jobs in the system conditioned on being in the **worse** state. Consequently, we have

$$\begin{aligned}
\chi &= \lambda(1 - \pi_{\text{clean}}) - \eta \\
&= \lambda(1 - \pi_{\text{clean}}) - \alpha_{\text{dead}}\pi_{\text{worse}} \cdot \mathbb{E}[N|\textbf{worse}] \\
&= \lambda(1 - \pi_{\text{clean}}) - \alpha_{\text{dead}} \sum_{j=0}^{\infty} j\pi_{(2,j)} \\
&= \lambda(1 - \pi_{\text{clean}}) - \alpha_{\text{dead}} \left(\frac{a_{2,0}r_0}{(1 - r_0)^2} + \frac{a_{2,1}r_1}{(1 - r_1)^2} + \frac{a_{2,2}r_2}{(1 - r_2)^2}\right).
\end{aligned}$$

Computing $\mathcal{R}$ also requires finding π_{bad} and π_{worse}. These values can be found by analyzing a finite state Markov chain (that cyclically alternates between the **normal**, **bad**, **worse**, and **clean** states with the appropriate rates), but we can also express these quantities, as well as π_{normal}, in terms of the $a_{m,k}$ coefficients as follows:

$$\begin{aligned}
\pi_{\text{normal}} &= \sum_{j=0}^{\infty} \pi_{(0,j)} = \frac{a_{0,0}}{1-r_0}, \\
\pi_{\text{bad}} &= \sum_{j=0}^{\infty} \pi_{(1,j)} = \frac{a_{1,0}+a_{1,1}}{1-r_1}, \\
\pi_{\text{worse}} &= \sum_{j=0}^{\infty} \pi_{(2,j)} = \frac{a_{2,0}+a_{2,1}+a_{2,2}}{1-r_2}.
\end{aligned}$$

Finally, we have the following exact determination of $\mathcal{R}$ under the **clean@dead** policy, again in terms of π_{clean} and the $a_{m,k}$ coefficients:

$$\begin{aligned}
\mathcal{R} &= q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}} \\
&= \lambda q(1-\pi_{\text{clean}}) - q\alpha_{\text{dead}}\left(\frac{a_{2,0}r_0}{(1-r_0)^2} + \frac{a_{2,1}r_1}{(1-r_1)^2} + \frac{a_{2,2}r_2}{(1-r_2)^2}\right) \\
&\quad - c\left(\frac{(a_{0,0}+a_{1,0}+a_{2,0})r_0}{(1-r_0)^2} + \frac{(a_{1,1}+a_{2,1})r_1}{(1-r_1)^2} + \frac{a_{2,2}r_2}{(1-r_2)^2}\right) \\
&\quad - \ell_{\text{bad}}\left(\frac{a_{1,0}+a_{1,1}}{1-r_1}\right) - \ell_{\text{worse}}\left(\frac{a_{2,0}+a_{2,1}+a_{2,2}}{1-r_2}\right).
\end{aligned}$$

We can now use our exact expression for the revenue rate, $\mathcal{R}$, in order to investigate the effect of various parameters on both revenue and on the optimal choice of clean-up policy.

3.3.3 Visible Malware Results

In this section, we use our exact analyses of the revenue rate, $\mathcal{R}$, under the **clean@bad**, **clean@worse**, and **clean@dead** policies in order to determine the impact of various parameters on the revenue and on the choices of optimal policy. Our calculations are exact modulo numerical precision (including measures taken to avoid badly condition matrices).

Due to our large parameter space (12 dimensions after normalizing time and money), we cannot exhaustively study the impact of all parameters on $\mathcal{R}$. While no single parameter set is representative of all systems, after consulting with the security company ForAllSecure, Inc., we chose a "default" parameter set that would be realistic in actual deployments. The default parameter set allows us to keep most parameters fixed while varying one parameter at a time, in order to observe the impact of various real-world phenomena on our system. This parameter set, **P1** (presented in Table 3.1), corresponds to the case of a mid-sized service provider facing frequent customers that has intermediate susceptibility to attacks (on the order of hours, days, and weeks for the three levels of infection, with cleanups taking minutes, hours, or days). Security loss rates are relatively low, as expected for visible malware, which is often a nuisance rather than extremely dangerous. Under the unaltered **P1** parameter set, the **clean@bad** policy dominates the **clean@worse** policy, which dominates the **clean@dead** policy.

Visible Malware Default Parameter Set														
	1/second										$	$/second		
	λ	μ_{fast}	μ_{slow}	μ_{slower}	α_{bad}	α_{worse}	α_{dead}	β_{bad}	β_{worse}	β_{dead}	q	c	ℓ_{bad}	ℓ_{worse}
P1	10	30	14	10.1	10^{-4}	10^{-5}	10^{-6}	10^{-2}	10^{-4}	10^{-5}	1	.05	.01	.05

Table 3.1: Table of the default parameter set, **P1**, used in this section. Recall the various families of parameters: λ (arrival rate), μ (service rates), α (malware infection rates), β (cleanup rates), q (hypothetical price of delay-free service), c (waiting cost rate), and ℓ (security loss rates).

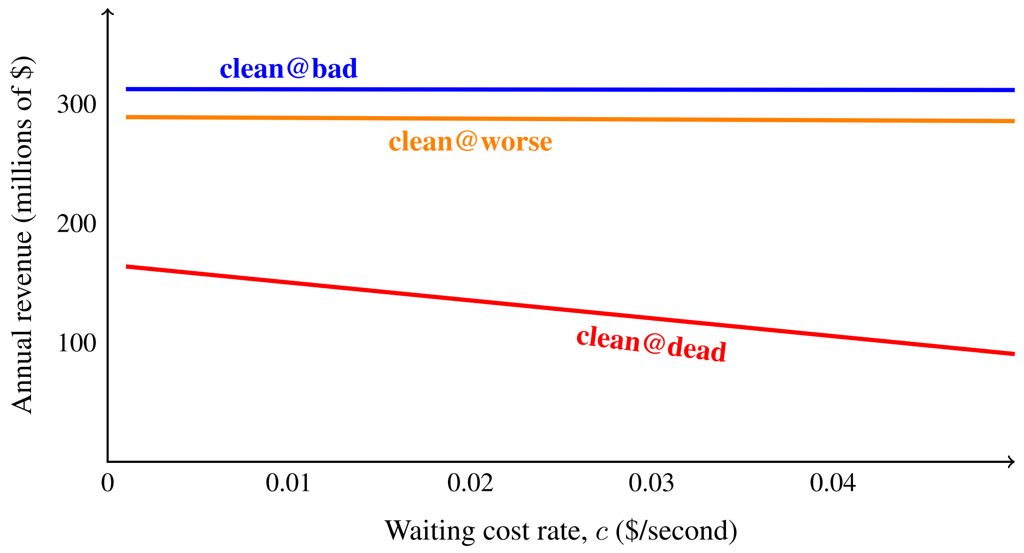


Figure 3.3: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P1**, as a function of the waiting cost parameter, c; **clean@bad** clearly outperforms the other policies, and the impact of c on **clean@bad** and **clean@worse** is negligible compared to the impact of c on **clean@dead**.

The impact of the waiting cost rate.

We first investigate the impact of the waiting cost rate, c, on the revenue rate $\mathcal{R}$ (see Figure 3.3). Here we see that for all policies there is a linear decline in $\mathcal{R}$ as a function of c, which is expected from the definition of $\mathcal{R}$. Right away, it is obvious that **clean@bad** is the optimal performing policy in terms of revenue maximization over all c. This is largely due to the fact that cleanup times become substantially higher as the system goes into the **worse** and **dead** malware states, which leads to greater downtime. Finally, we realize that the impact of c is much more pronounced on **clean@dead**, due the fact that in the **worse** state (which is avoided by the other two policies), the load is very high (about 99%), leading to substantially longer waiting times. The takeaway is that the more impact a malware state has on performance, the more important the waiting cost becomes.

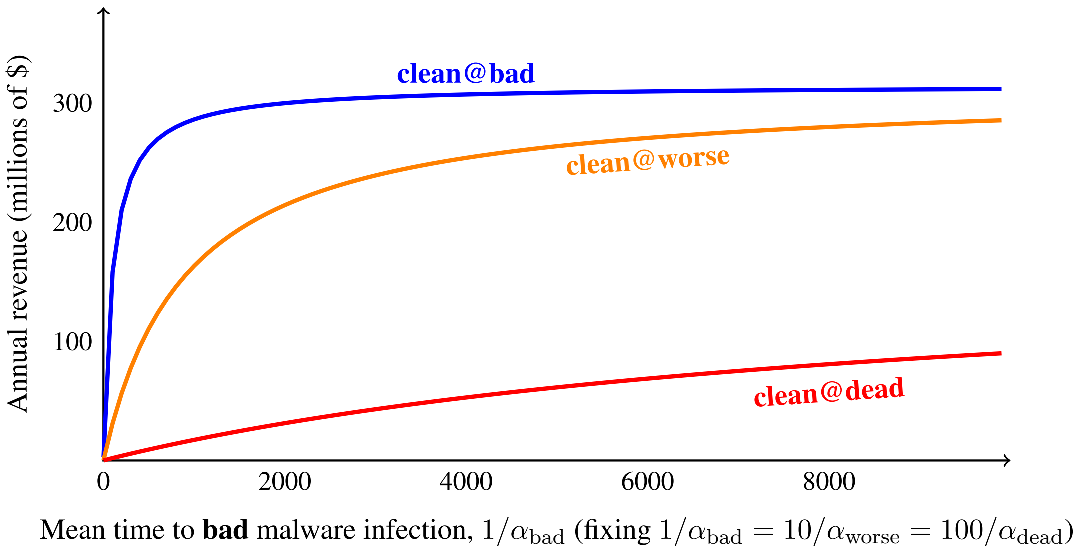


Figure 3.4: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P1**, as a function of the mean time to a **bad** malware infection, $1/\alpha_{\text{bad}}$, keeping $1/\alpha_{\text{bad}} = 10/\alpha_{\text{worse}} = 100/\alpha_{\text{dead}}$ fixed; **clean@bad** clearly outperforms the other policies, but the revenue under **clean@worse** approaches that of **clean@bad** as infections become less frequent.

The impact of infection rates.

Next, we investigate the impact of the malware infection rates on $\mathcal{R}$. We fix the relationship $1/\alpha_{\text{bad}} = 10/\alpha_{\text{worse}} = 100/\alpha_{\text{dead}}$ and observe how $\mathcal{R}$ changes as we vary these parameters (see Figure 3.4). Here we observe that **clean@bad** is again the optimal policy over the entire parameter range. However, as infections become rarer and rarer, the **clean@worse** policy does not lag too far behind **clean@bad**, with the difference in the revenue under these policies eventually vanishing. Improvement under the **clean@dead** policy, on the other hand, is much slower. We can draw two conclusions from Figure 3.4. First, cleaning up as frequently as possible can prevent escalation to more serious attacks, which means that detecting such attacks early (e.g., via intrusion detection) is valuable (and quantifiable in terms of annual revenue). In 3.4 we will revisit the importance of knowing whether the system is compromised or not. Second, if attacks are extremely infrequent, the effect of the choice of cleanup policy on revenue will not be substantial (so long as security loss rates are not too high, that is, so long as malware is primarily a nuisance rather than extremely dangerous).

The impact of cleanup rates.

We proceed to investigate the impact of the cleanup rates on $\mathcal{R}$. Here we leave β_{bad} fixed, as the first level of cleanup usually corresponds to something amounting to a system reboot, which typically takes on the order of 100 seconds. However, we fix $1/\beta_{\text{worse}} = 10/\beta_{\text{dead}}$, and vary these parameters to observe how different cleanup times for the **worse** and **dead** malware states might impact the revenue and the choice of optimal policy. For sufficiently low $1/\beta_{\text{worse}}$, the

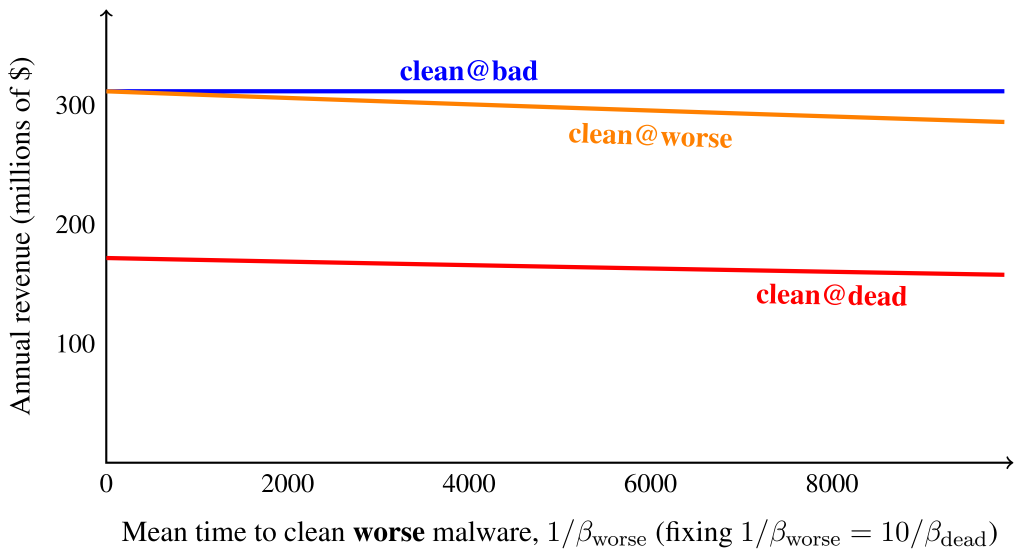


Figure 3.5: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P1**, as a function of the mean time to clean **worse** malware, $1/\beta_{\text{worse}}$, keeping $1/\beta_{\text{worse}} = 10/\beta_{\text{dead}}$ fixed; **clean@bad** is again the best policy except at very low values of $1/\beta_{\text{worse}}$, with the policies diverging near linearly from one another as $1/\beta_{\text{worse}}$ grows.

clean@worse policy can actually negligibly outperform the **clean@bad** policy, but once again, the **clean@bad** policy proves to be dominant over the parameter range, and there is a nearly linear decline in the revenue of all policies as the cleanup times grow larger, with **clean@bad** not being impacted at all since it never has to clean at the **worse** and **dead** malware states.

In all cases examined, **clean@bad** is clearly the optimal policy and **clean@dead** is always the worst. That said, we can crate artificial examples which reverse these trends, but these examples require parameters featuring unrealistically low levels of infection rates or fast cleanup times that are unlikely to correspond to real system configurations. Hence, for the visible malware model, it is indeed in the interest of revenue to clean the system at the first sign of trouble. We shall see that trends reverse for even reasonable systems under the hidden malware model.

Essentially, if you know you have been infected, you should clean up the system as soon as possible. This observation validates common practices among practitioners when an infection occurs. The best courses of action under hidden malware, however, will be much more subtle and complex.

3.4 The Case of Hidden Malware

In this section we consider hidden malware, which is not detectable in the non-**dead** states. This case is of interest because it more accurately models the types of malware that pose serious threats to a system's security. Attackers often want their attacks to go undetected, but even stealthy malware attacks can have observable effects on the system, including *performance degradation*. While the service provider cannot know whether they have been infected by malware or not, they can still monitor their system and observe performance degradation. However, malware is not the only reason that a system may be suffering from performance degradation; the degradation may be due to the system load, outdated software, or other external factors. For this reason, *we model performance degradation and malware separately*. At any given time, the system's *performance state* is *observable* but its *malware state* is *unobservable*.

As in the previous section, we first introduce the hidden malware model, then present an analysis, and conclude with select results, highlighting our observations.

3.4.1 Hidden Malware Model

In the hidden malware model, as in the visible malware model, the system can be in one of four malware states: **normal** (uninfected), **bad**, **worse**, and **dead**, with each successive malware state leading to higher security costs and slower performance. Additionally, a non-**dead** system can be in one of three *performance* states: **fast**, **slow**, or **slower**, serving jobs at rates μ_{fast}, μ_{slow}, and μ_{slower}, respectively. While the performance state can be observed by the service provider, the malware state is, as the name suggests, hidden, unless the system is **dead**. However, there is a correlation structure between the performance state and the malware state.[8] In particular, we assume that due to the resource-hungry nature of malware,[9] a **fast** system is necessarily **normal**, but a **slow** system can be either in the **normal** or **bad** state, and a **slower** system can be in any of the non-**dead** states.

The queueing, pricing, and revenue models are identical to those in the visible malware case; the objective remains to maximize the revenue rate, given by

$$\mathcal{R} = q\chi - c \cdot \mathbb{E}[N] - \mathcal{L} = q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}}. \tag{3.3}$$

While the service rate of the system depends on the system's *performance state*, it is the *malware state* of the system determines the security loss rate $\mathcal{L}$, as $\mathcal{L}$ directly depends on π_{dead} and π_{bad}, and does not directly depend on π_{slow} and π_{slower}.

Hidden Malware Evolution

The server can become infected by malware and/or degrade in performance in stages, as shown in Figure 3.6 (note that the chains of interest must still track the number of jobs), but only the performance state is observable. Initially, the system is **normal** and **fast**. Performance degradation occurring due to reasons other than malware causes a **fast** system to become **slow** with rate γ_{slow} and a **slow** system to become **slower** with rate γ_{slower}. A system also evolves form **normal** to **bad** to **worse** to **dead** with rates α_{bad}, α_{worse}, and α_{dead}, respectively. Moreover, if a

[8]In the visible malware model proposed in the previous section of this paper, there is a *perfect* correlation between the malware state and the performance state; in particular a **fast** system is always **normal**, a **slow** system is always **bad**, a **slower** system is always **worse**. By contrast, the hidden malware model features an *imperfect* correlation between malware and performance.

[9]Note that our framework can also model fully stealthy malware with no performance degradation. In this paper we specifically investigate malware that has a performance degradation effect.

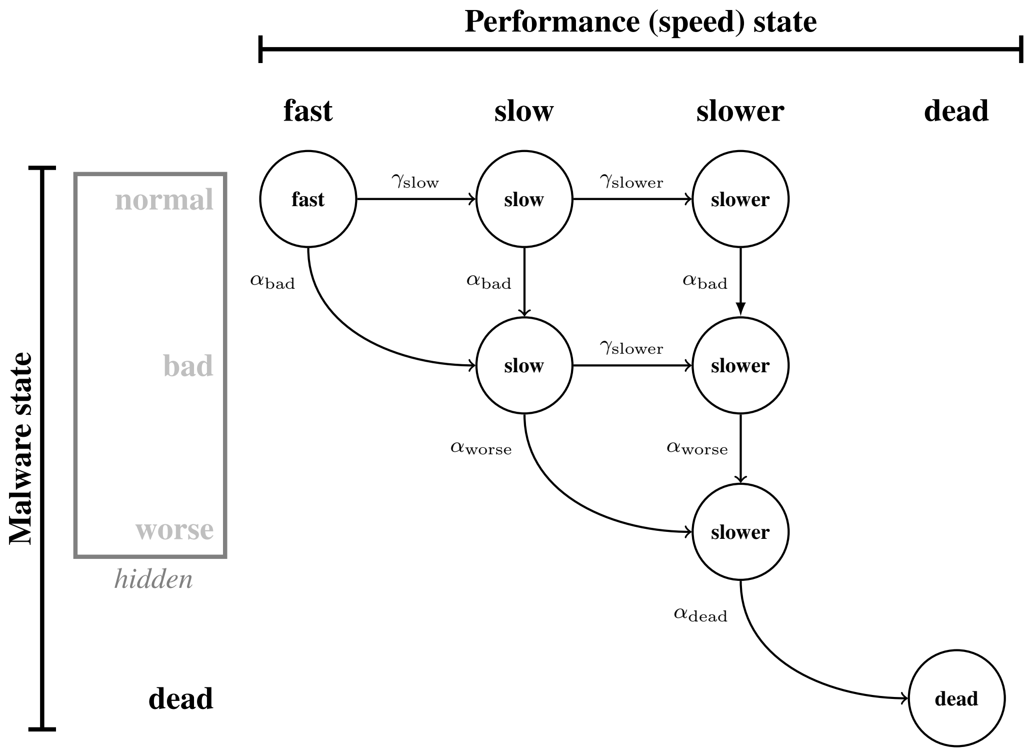


Figure 3.6: The CTMC for the evolution of performance degradation and malware infection on a system under the hidden malware model without system administration. The number of jobs have been omitted, but can be viewed as being tracked by states coming "out of the page" in the third-dimension. The system transitions from **fast** to **slow** with rate γ_{slow} and from **slow** to **slower** with rate γ_{slower}. The system transitions from **normal** to **bad** with rate α_{bad} (limiting the maximum speed to **slow**), from **bad** to **worse** with rate α_{worse} (limiting the maximum speed to **slower**), and from **worse** to **dead** with rate α_{dead}. Unless the system is **dead**, the *malware state* is hidden.

fast system just became **bad**, the resource-hungry nature of the malware will cause it to immediately become **slow** (a **slow** or **slower** system does not change speed when it becomes **bad**).[10] Similarly, if a **slow** system just became **worse**, it immediately becomes **slower**. Unlike the other malware states, the **dead** state is *not* hidden.

Hidden malware cleanup

A system potentially infected by hidden malware can be purged of malware (if any) and restored to full speed by a cleanup procedure. As before, such a cleanup procedure requires that all existing jobs are discarded from the system (customers are refused service and refunded q), and the system stops admitting customers for a duration of time until the system is restored to the

[10]If a system is already **slow** when it becomes **bad**, we assume that the system is slow enough to obfuscate the impact of malware on performance, and hence does not cause a drop in performance. Alternatively, we could model this complexity with additional states if desired.

normal and **fast** state. The length of time devoted to the cleanup procedure depends on the worst possible malware that *could* have infected the system—based on our model—and hence, this duration actually depends on the *performance* state. We call this the *pessimistic cleanup assumption* (e.g., a **normal slower** system takes the same amount of time to clean as a **worse slower** system). The cleanup procedure lasts an amount of time that is exponentially distributed with rate β_{bad}, β_{worse}, or β_{dead} when initiating the procedure in the **slow**, **slower**, or **dead** states, respectively;[11] that is, due to the pessimistic cleanup assumption, we clean up a **slow** system as if it was **bad** and a **slower** system as if it was **worse**. For simplicity, we again assume that there are no security losses during the cleanup procedure.

In the setting with hidden malware, we can again consider the clean-up policies that we examined in the case of visible malware, but modifying them to respond to a change in the *performance* state, rather than the now unobservable *malware* state. This results in the following policies:

- **clean@slow**: Clean once the system transitions to the **slow** malware state.
- **clean@slower**: Clean once the system transitions to the **slower** malware state.
- **clean@dead**: Clean once the system transitions to the **dead** malware state.

Hidden malware cleanup: delayed policies

Motivated by the idea that the service provider may be content with slower performance, but *not* with malware infections, we introduce policies that delay cleanup actions after transitioning to a particular cleanup state. For example, if $\alpha_{\text{bad}} \ll \gamma_{\text{slow}}$, a transition to the **slow** state is unlikely to suggest that the system has been infected by **bad** malware, and one can persist in a **slow** system for a considerable period of time with little risk of being unknowingly infected. The *delayed* policies we consider are as follows:

- **delayed-clean@slow**(ξ): After the system transitions to the **slow** state, wait an amount of time that is exponentially distributed with rate ξ, then clean; if a transition to **slower** occurs, clean immediately.
- **delayed-clean@slower**(ξ): After the system transitions to the **slower** state, wait an amount of time that is exponentially distributed with rate ξ, then clean; if a transition to **dead** occurs, clean immediately.
- **hybrid-delay**(ξ_1, ξ_2): After the system transitions to the **slow** state, wait an amount of time that is exponentially distributed with rate ξ_1, then clean, unless a transition to **slower** occurs, in which case, disregard the wait so far and instead wait an amount of time that is independently exponentially distributed with rate ξ_2, *then* clean up the system; if a transition to **dead** occurs, clean immediately.

Interpreting exponentially distributed random waiting times with rates 0 or ∞ to be the constant durations of infinite and zero time, respectively, we have the following observations:

- **delayed-clean@slow**(∞) corresponds to **clean@slow**,

[11] As in the case of visible malware, the analysis and results remain unchanged if the cleanup durations are drawn from non-exponential distributions with the same means.

- **delayed-clean@slow**(0) and **delayed-clean@slower**(∞) correspond to **clean@slower**,
- **hybrid-delay**(ξ, ∞) corresponds to **delayed-clean@slow**(ξ),
- **delayed-clean@slower**(0) corresponds to **clean@dead**, and
- **hybrid-delay**$(0, \xi)$ corresponds to **delayed-clean@slower**(ξ) for all rates ξ.

Consequently, the family of policies **hybrid-delay**(ξ_1, ξ_2) spans the other two delay-based policies, along with **clean@slow**, **clean@slower**, and **clean@dead**, and it is sufficient to determine $\mathcal{R}$ under this policy.

If the delays were *deterministic* rather than random, one could potentially obtain even greater revenue rates: a random delay is a lottery over a continuum of deterministic delays (with a potentially new outcome after every cleanup event), and so the best policy among the support of this lottery will do no worse than the lottery itself. However, we restrict attention to exponentially distributed delays for the purpose of tractability, in order to maintain a Markovian structure.[12]

Hidden malware cleanup: dynamic policies

Thus far we have only considered policies that make use of the *performance* state of the system, but we can also make cleanup decisions based on j, the instantaneous number of jobs in the system. For example, we may wish to perform cleanups only when the queue length is relatively short in order to avoid discarding too many jobs at the start of each cleanup procedure.

To this end, we extend the class of cleanup policies under consideration to include *dynamic policies* that take the current number of jobs, j, into account. There are a rich family of such dynamic policies in this space. For example, when there are j jobs in the system, we can clean up a system after waiting an amount of time that is exponentially distributed with rate $f(j)$ or $g(j)$ when we are in the **slow** or **slower** performance state, respectively. For such policies, whenever we transition to a new performance state or the number of jobs in the system changes, we disregard the wait so far and begin to wait for a new and independently drawn random duration of time before cleaning. Determining the optimal policy among this rich class would require solving an intractable dynamic program.

In this paper, we will instead consider two much simpler families of dynamic policies that are variants of the **hybrid-delay**(ξ_1, ξ_2) family of delay policies. We call the policies in these families *threshold policies*, as they depend on a threshold parameter, Θ. The threshold policies are defined as follows:

- **hybrid-delay**$_{\geq\Theta}(\xi_1, \xi_2)$: After the system transitions to the **slow** state, wait an amount of time that is exponentially distributed with rate ξ_1, then clean, unless a transition to **slower** occurs. Once the total amount of time in the **slower** state with $j \geq \Theta$ jobs exceeds an (independent) exponentially distributed random variable with rate ξ_2, clean the system; if a transition to **dead** occurs, clean immediately.
- **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$: This is the same as the preceding policy, except when in the **slower** state, we clean based on having spent sufficient time with $j \leq \Theta$.

[12]One could more closely approximate deterministic delays by implementing them as multi-phase Erlang distributions, which approach deterministic distributions as the number of (identical) phases tends to infinity, while keeping the mean fixed.

The **hybrid-delay**$_{\geq\Theta}(\xi_1, \xi_2)$ policy (respectively, the **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$ policy) is like the **hybrid-delay**(ξ_1, ξ_2) policy, except that the transition from the **slower** state to the corresponding cleanup state occurs with rate ξ_2 only when the number of jobs in the system is *at least* Θ (respectively, *at most* Θ); otherwise, this transition cannot occur. Consequently, the **hybrid-delay**$_{\geq 0}(\xi_1, \xi_2)$ and **hybrid-delay**$_{\leq\infty}(\xi_1, \xi_2)$ policies correspond to the **hybrid-delay**(ξ_1, ξ_2) policy.

Finally, we introduce one additional dynamic policy. The **drain@slower** policy is like the **clean@slow** policy, except that when the system transitions to the **slower** state, a cleanup procedure will only be initiated if there are $j = 0$ jobs in the system. Otherwise, we must wait until the system is empty, but while in the **slower** state no new jobs are admitted into the system (i.e., the arrival rate drops from λ to 0). Note that unlike the other policies, this policy exhibits an element of admission control.

Hidden malware cleanup: the omniscient policy

Finally, we also consider the **omniscient** cleanup policy, a *hypothetical* policy that *is* aware of the malware state of the system. The **omniscient** policy is of interest as it serves as a benchmark that we compare our other policies to and also allows us to quantify the benefits of perfect intrusion detection. This policy will clean-up the system as soon as it transitions to a particular subset of the joint malware-performance states, choosing the (for simplicity, non-delaying, non-dynamic) policy that maximizes $\mathcal{R}$ among those that are feasible in a "visible malware setting" under the "hidden malware correlation structure" presented in this setting. In particular, **omniscient** results in one of the following policies, depending on the parameters of interest:

- a variant of (i) **clean@slow** or (ii) **clean@slower**, where the cleanup duration depends on the *malware state* when the system transitions to the **slow** or **slower** performance state,[13] effectively ignoring the pessimistic cleaning assumption
- (iii) **clean@bad** or (iv) **clean@worse**, which require omniscience because malware is hidden
- (v) a hybrid of **clean@slower** and **clean@bad**, which initiates a cleanup procedure as soon as the system enters *either* the **slower** performance state *or* the **bad** malware state
- (vi) a more restrictive version of **clean@slower**, which initiates a cleanup procedure on a **slower** system only if it is *also* infected (i.e., in the **bad** or **worse** state)
- (vii) **clean@dead**, which *does not* require omniscience, but is possibly (albeit rarely) optimal.

We reiterate that the **omniscient** policy will mimic whichever of these policies maximizes the revenue rate for a particular parameter set.

With our cleanup policies defined, we turn our attention toward analyzing the revenue rate, $\mathcal{R}$, under these policies.

[13]For these variants, we assume that a cleanup of a **normal** system is assumed to take as long as the cleanup of a **bad** system (i.e., the duration is exponentially distributed with rate β_{bad}); we assume that β_{bad} is the fastest possible cleanup rate.

3.4.2 Hidden Malware Analysis

We restrict attention to the analysis of $\mathcal{R}$ under the **hybrid-delay**(ξ_1, ξ_2) class of policies. With the exception of the dynamic policies,[14] revenues under the other policies can be computed by taking the appropriate limits (and in the case of **omniscient**, by choosing the best among several candidate policies).[15]

As in the case of visible malware, we wish to compute,

$$\mathcal{R} = q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}},$$

and the difficulty again lies in computing χ and $\mathbb{E}[N]$, as they again require the analysis of a two-dimensional infinite state Markov chain (this time with six, rather than three, phases).

The first step in our approximation relies on determining the limiting probability associated with the six joint malware-performance states (see Figure 3.6) and each of three separate **clean** states: **clean-short**, **clean-med**, and **clean-long**, which return to the **normal fast** state with rates β_{bad}, β_{worse}, and β_{dead}, respectively. For notational convenience we label the phases and states of interest with the numbers 0–8, as follows:

- Phase 0 corresponds to the **normal** state of a **fast** system
- Phases 1 and 2 correspond to the **normal** and **bad** states of a **slow** system, respectively,
- Phases 3, 4, and 5 corresponds to the **normal**, **bad**, and **worse** states of a **slower** system, respectively, and
- *states* 6, 7, and 8 correspond to **clean-short**, **clean-med**, and **clean-long** states, respectively.

With this notation, we proceed to apply the Clearing Analysis on Phases (CAP) method to the hidden malware model under the **hybrid-delay**(ξ_1, ξ_2) policy. The CTMC we study looks like a more complicated version of the chain depicted in Figure 3.2 (from the analysis of the **clean@dead** policy for the visible malware model), except that it consist of six *phases*, and three *cleanup states*, with some transitions across phases "skipping over" intermediate phases, and multiple phases transitioning directly to cleanup states. Our CTMC consists of an infinite *repeating portion* and a finite *non-repeating portion*.

The repeating portion of our CTMC is made up of the six phases, one corresponding to each of the joint malware-performance states, 0–5. Each phase is an infinite collection of states making up a birth-death process tracking the number of jobs in the system. Each birth-death process has an arrival rate of λ, and departure rate of μ_{fast} (in Phase 0), μ_{slow} (in Phases 1 and 2), or μ_{slower} (in Phases 3, 4, and 5). Each state in the repeating portion of the Markov chain is denoted by (m, j), where m is the phase and j denotes the number of jobs in the system. We observe that the transitions across these phases are *unidirectional* in nature (i.e., when moving from one phase in $\{0, \dots, 5\}$ to another, the phase number always increases), which makes the

[14]Revenues for the threshold policies, **hybrid-delay**$_{\geq\Theta}(\xi_1, \xi_2)$ and **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$, are calculated by following a modification of the procedure described in this section that involves expanding the non-repeating portion of the Markov chain of interest to encompass all states where the number of jobs in the system is less than Θ.

[15]There can be computational advantages to determining $\mathcal{R}$ under the other policies directly (rather than taking limits of $\mathcal{R}$ under the **hybrid-delay**(ξ_1, ξ_2) policies). Such direct calculations are possible through simple modifications to the analyses presented in this section.

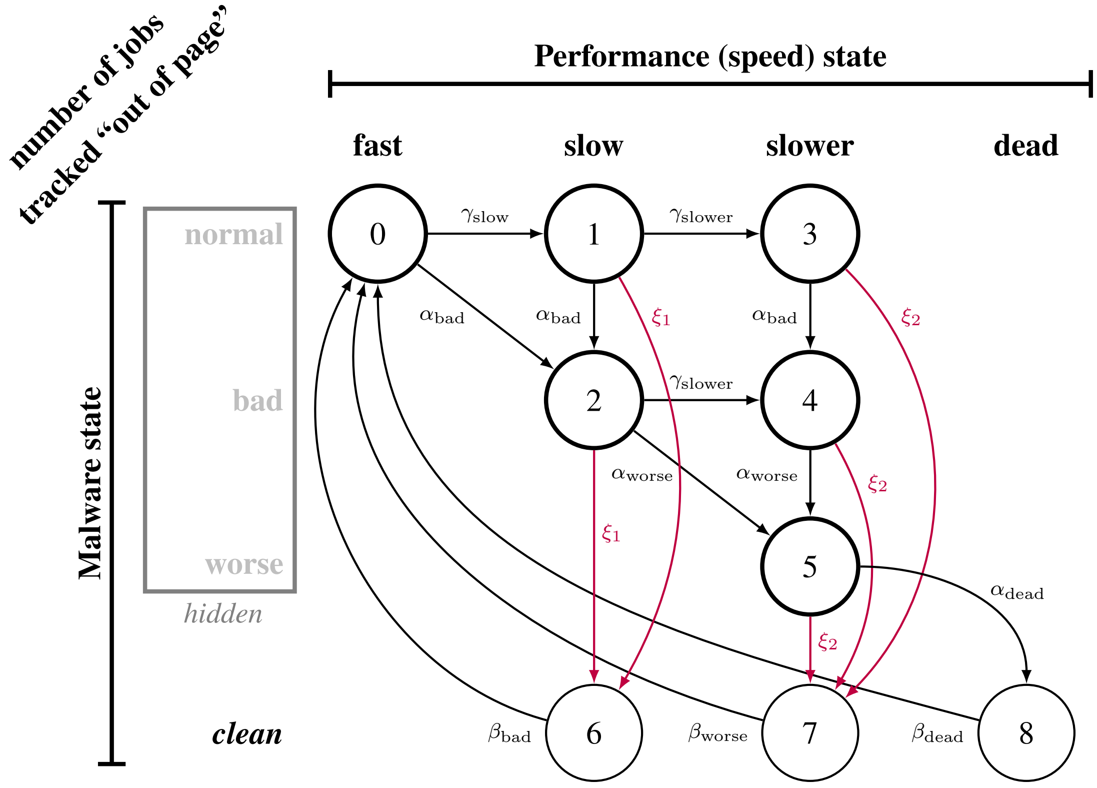


Figure 3.7: The CTMC governing the system under the **hybrid-delay**(ξ_1, ξ_2) cleanup policy. States and phases are labeled by the numbers 0–8, for notational convenience, with *Phases* 0–5 denoting the six joint malware-performance states, and *states* 6–8 denoting the three possible **clean** states (**clean-short**,**clean-med**, and **clean-long**). The transitions due to intentional delays imposed by the cleanup policy are shown in color for clarity. Note that each *phase* (denoted by a thicker border) is actually an infinite collection of states that evolves according to a birth-death process.

CTMC amenable to the CAP method. Moreover, transitions across these phases are independent of the number of jobs in the system, with associated transition rates depicted in Figure 3.12 (e.g., a transition from Phase 2 to Phase 5 occurs with rate α_{worse}, regardless of the number of jobs in the system). When any such transition occurs the number of jobs in the system remains unchanged.

The non-repeating portion of our CTMC is made up of the three cleanup states: **clean-short** (6), **clean-med** (7), and **clean-long** (8). Unlike phases, these are *single states*. As these states represent the system undergoing a cleanup procedure, they always transition directly to state (0,0) (i.e., Phase 0 with an empty queue). Transitions *to* one of cleanup states from one of the phases are independent of the queue length at the time of the transition, with associated transition rates depicted in Figure 3.12. Once such a transition is made, the queue length is reset to zero (as the jobs are discarded). As with transitions across phases, transitions from a phase to a cleanup state are independent of the number of jobs in the system; the associated transition rates are depicted in Figure 3.12 (e.g., a transition from Phase 6 to state 8 occurs with rate ξ_2).

We use the following notation:

- $\pi_{(m,j)}$ is the limiting probability of being in Phase $m \in \{0, \ldots, 5\}$ with $j \geq 0$ jobs
- π_6, π_7, π_8 are the limiting probabilities of being in cleanup states 6, 7, and 8, respectively,
- μ_m is the service rate in Phase m ($\mu_0 = \mu_{\text{fast}}$, $\mu_1 = \mu_2 = \mu_{\text{slow}}$, $\mu_3 = \mu_4 = \mu_5 = \mu_{\text{slower}}$), and
- α_m is the rate at which the system *leaves* Phase m:
  - ⋆ $\alpha_0 = \alpha_{\text{bad}} + \gamma_{\text{slow}}$
  - ⋆ $\alpha_1 = \alpha_{\text{bad}} + \gamma_{\text{slower}} + \xi_1$
  - ⋆ $\alpha_2 = \alpha_{\text{worse}} + \gamma_{\text{slower}} + \xi_1$
  - ⋆ $\alpha_3 = \alpha_{\text{bad}} + \xi_2$
  - ⋆ $\alpha_4 = \alpha_{\text{worse}} + \xi_2$
  - ⋆ $\alpha_5 = \alpha_{\text{dead}} + \xi_2$.

We again solve for limiting probabilities in the form

$$\pi_{(m,j)} = \sum_{k=0}^{m} a_{m,k} r_k^j,$$

with

$$r_k \equiv \lambda + \mu_k + \alpha_k - \frac{\sqrt{(\lambda + \mu_k + \alpha_k)^2}}{2\mu_k)}$$

(where the r_k values are assumed to be distinct). Determining the complete limiting distribution of the CTMC requires determining the $a_{m,k}$ coefficients (for $0 \leq k \leq m \leq 5$), together with π_6, π_7, π_8. These variables, together with the redundant $\pi_{(m,0)}$ variables, are the solutions to a linear system of equations, which are a combination of balance equations, the normalization equation, and relationships, which are derived via the CAP method. This system of equations is more complicated than the corresponding system for the case of visible malware, because in the present model, some phases can directly transition to more than one other phase and multiple phases can transition to cleanup states. Consequently, one visits phases in a non-deterministic order. The linear system is as follows:

$$
\begin{cases}
a_{0,0} = \pi_{(0,0)} \\
a_{m,m} = \pi_{(m,0)} - \sum_{k=0}^{m-1} a_{m,k} \qquad (1 \le m \le 5) \\
a_{1,0} = \dfrac{r_0 r_1 \gamma_{\text{slow}} a_{0,0}}{(r_0 - r_1)(\lambda - \mu_0 r_0 r_1)} \\
a_{2,0} = \dfrac{r_0 r_2 \alpha_{\text{bad}} (a_{0,0} + a_{1,0})}{(r_0 - r_2)(\lambda - \mu_0 r_0 r_2)} \\
a_{2,1} = \dfrac{r_1 r_2 \alpha_{\text{bad}} a_{1,1}}{(r_1 - r_2)(\lambda - \mu_1 r_1 r_2)} \\
a_{3,k} = \dfrac{r_k r_3 \gamma_{\text{slower}} a_{1,k}}{(r_k - r_3)(\lambda - \mu_k r_k r_3)} \qquad (0 \le k \le 1) \\
a_{3,2} = 0 \\
a_{4,k} = \dfrac{r_k r_4 (\gamma_{\text{slower}} a_{2,k} + \alpha_{\text{bad}} a_{3,k})}{(r_k - r_4)(\lambda - \mu_k r_k r_4)} \qquad (0 \le k \le 2) \\
a_{4,3} = \dfrac{r_3 r_4 \alpha_{\text{bad}} a_{3,3}}{(r_3 - r_4)(\lambda - \mu_3 r_3 r_4)} \\
a_{5,k} = \dfrac{r_k r_5 \alpha_{\text{worse}} (a_{2,k} + a_{4,k})}{(r_k - r_5)(\lambda - \mu_k r_k r_5)} \qquad (0 \le k \le 2) \\
a_{5,3} = \dfrac{r_3 r_5 \alpha_{\text{worse}} a_{4,3}}{(r_3 - r_5)(\lambda - \mu_3 r_3 r_5)} \\
a_{5,4} = \dfrac{r_4 r_5 \alpha_{\text{worse}} a_{4,4}}{(r_4 - r_5)(\lambda - \mu_4 r_4 r_5)} \\
\pi_{(0,0)} = \dfrac{1}{\lambda + \alpha_0} (\beta_{\text{bad}} \pi_6 + \beta_{\text{worse}} \pi_7 + \beta_{\text{dead}} \pi_8 + \mu_0 r_0 a_{0,0}) \\
\pi_{(1,0)} = \dfrac{1}{\lambda + \alpha_1} \left(\mu_1 \sum_{k=0}^{1} (r_k a_{1,k}) + \gamma_{\text{slow}} \pi_{(0,0)} \right) \\
\pi_{(2,0)} = \dfrac{1}{\lambda + \alpha_2} \left(\mu_2 \sum_{k=0}^{2} (r_k a_{2,k}) + \alpha_{\text{bad}} (\pi_{(0,0)} + \pi_{(1,0)}) \right) \\
\pi_{(3,0)} = \dfrac{1}{\lambda + \alpha_3} \left(\mu_3 \sum_{k=0}^{3} (r_k a_{3,k}) + \gamma_{\text{slower}} \pi_{(1,0)} \right) \\
\pi_{(4,0)} = \dfrac{1}{\lambda + \alpha_4} \left(\mu_4 \sum_{k=0}^{4} (r_k a_{4,k}) + \gamma_{\text{slower}} \pi_{(2,0)} + \alpha_{\text{bad}} \pi_{(3,0)} \right) \\
\pi_{(5,0)} = \dfrac{1}{\lambda + \alpha_5} \left(\mu_5 \sum_{k=0}^{5} (r_k a_{5,k}) + \alpha_{\text{worse}} (\pi_{(2,0)} + \pi_{(4,0)}) \right) \\
\pi_6 = \dfrac{\xi_1}{\beta_{\text{bad}}} \sum_{m=1}^{2} \sum_{k=0}^{m} \dfrac{a_{m,k}}{1 - r_k} \\
\pi_7 = \dfrac{\xi_2}{\beta_{\text{worse}}} \sum_{m=3}^{5} \sum_{k=0}^{m} \dfrac{a_{m,k}}{1 - r_k} \\
\pi_8 = \dfrac{\alpha_{\text{dead}}}{\beta_{\text{dead}}} \sum_{k=0}^{5} \dfrac{a_{5,k}}{1 - r_k} \\
1 = \left(\sum_{m=0}^{5} \sum_{k=0}^{m} \dfrac{a_{m,k}}{1 - r_k} \right) + \pi_6 + \pi_7 + \pi_8
\end{cases}
$$

In theory, one can obtain closed-form solutions for the limiting probability distribution of the CTMC by symbolically solving the system above, although this solution will likely be very unwieldy. This system can also be used to solve exact numeric solutions; various techniques can be used to circumvent badly conditioned matrices. With the limiting probabilities determined in a convenient form, we can compute $\mathbb{E}[N]$ and χ in terms of the π_6, π_7, π_8 and $a_{m,k}$ values. $\mathbb{E}[N]$ is given as

$$\mathbb{E}[N] = \sum_{j=0}^{\infty} j \cdot \mathbb{P}(N = j) = \sum_{m=0}^{5} \sum_{k=0}^{5} \frac{a_{m,k} r_m}{(1 - r_k)^2}. \tag{3.4}$$

Recalling that η is the rate at which customers are discarded, and letting $X(m)$ be the rate of initiating a cleanup event in Phase m, that is

$$X(m) = \begin{cases} 0, & m = 0 \\ \xi_1, & m \in \{1, 2\} \\ \xi_2, & m \in \{3, 4\} \\ \xi_2 + \alpha_{\text{dead}}, & m = 5, \end{cases} \tag{3.5}$$

we have the following expression for χ:

$$\chi = \lambda(1 - \pi_{\text{clean}}) - \eta = \lambda(1 - \pi_6 - \pi_7 - \pi_8) - \sum_{m=0}^{5} \sum_{k=0}^{m} \frac{X(m) \cdot a_{m,k}}{1 - r_k}. \tag{3.6}$$

Computing $\mathcal{R}$ also requires finding for π_{bad} and π_{worse}. We can either analyze a finite-state Markov chain or express these quantities (and π_{normal}) in terms of the $a_{m,k}$ coefficients as follows:

$$\begin{aligned} \pi_{\text{normal}} &= \sum_{j=0}^{\infty} \pi_{(0,j)} = \frac{a_{0,0}}{1 - r_0} + \sum_{k=0}^{1} \frac{a_{1,k}}{1 - r_k} + \sum_{k=0}^{3} \frac{a_{3,k}}{1 - r_k}, \\ \pi_{\text{bad}} &= \sum_{j=0}^{\infty} \pi_{(1,j)} = \sum_{k=0}^{2} \frac{a_{2,k}}{1 - r_k} + \sum_{k=0}^{4} \frac{a_{4,k}}{1 - r_k}, \\ \pi_{\text{worse}} &= \sum_{j=0}^{\infty} \pi_{(2,j)} = \sum_{k=0}^{5} \frac{a_{5,k}}{1 - r_k}. \end{aligned}$$

Finally, observing that states 2 and 4 correspond to the **bad** malware state, while state 5 corresponds to the **worse** state, we have the following exact expression for $\mathcal{R}$ under the **hybrid-delay**(ξ_1, ξ_2) policy:

$$\begin{aligned} \mathcal{R} &= q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}} \\ &= \lambda q(1 - \pi_6 - \pi_7 - \pi_8) - \sum_{m=0}^{5} \sum_{k=0}^{m} \frac{(qX(m) + cr_m)a_{m,k}}{(1 - r_k)^2} - \ell_{\text{bad}} \left(\sum_{k=0}^{2} \frac{a_{2,k}}{1 - r_k} + \sum_{k=0}^{4} \frac{a_{4,k}}{1 - r_k} \right) - \ell_{\text{worse}} \left(\sum_{k=0}^{5} \frac{a_{5,k}}{1 - r_k} \right) \end{aligned}$$

With this expression, we can evaluate the **hybrid-delay**(ξ_1, ξ_2) family of cleanup policies, including many simple policies such as **clean@slow**, **clean@slower**, and **clean@dead**.

Hidden Malware Default Parameter Set																
	1/second												$	$/second		
	λ	μ_{fast}	μ_{slow}	μ_{slower}	α_{bad}	α_{worse}	α_{dead}	γ_{slow}	γ_{slower}	β_{bad}	β_{worse}	β_{dead}	q	c	ℓ_{bad}	ℓ_{worse}
P2	10	30	14	10.1	10^{-3}	10^{-4}	10^{-7}	10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-4}	1	.05	.5	5

Table 3.2: Table of default parameter set, **P2**, used in this section. Recall the various families of parameters: λ (arrival rate), μ (service rates), α (malware infection rates), γ (non-malware degradation rates), β (cleanup rates), q (price of hypothetical delay-free service), c (waiting cost rate), and ℓ (security loss rates).

3.4.3 Hidden Malware Results

In this section, we use our ability to compute exact revenue rates under the hidden malware model in order to evaluate and compare various cleanup policies. Our results will be presented in the form of a case study for a system under the default parameter set, **P2**, a stylized parameter set developed after consulting with the security company ForAllSecure, Inc. This parameter set is presented in Table 3.4.3. Unlike **P1**, **P2** features non-malware related performance degradation and substantial security loss rates. Due to the high security costs, **clean@dead** performs poorly on all realistic cases, so for simplicity, we omit this policy from our results figures.

Our case study will focus on answering the following questions:

1. Can we gain more by improving cleanup speeds or improving intrusion detection?
2. Should we act immediately upon a performance degradation event or delay our cleanup actions?
3. What do we gain from incorporating queue length information into cleanup decisions?

In answering these questions for our particular case, we draw several insights regarding the malware cleanup problem.

Can we gain more by improving cleanup speeds or improving intrusion detection?

There are several ways in which a service provider can invest resources into delivering a more robust service with the hopes of generating greater revenue. In the setting of hidden malware, two such avenues of improvement are (i) *improving cleanup speeds* (e.g., by hiring additional staff when a potential problem is identified, or by automating more steps associated with the cleanup procedure) and *improving intrusion detection* (e.g., by developing or purchasing in intrusion detection software which can reliably inform the system administrator of an attack). We explore the benefits that arise from both of these approaches.[16]

In order to quantify the benefits of improving cleanup speeds under the **P2** parameter set, we leave β_{bad} fixed at 10^{-2} per second, and let $\beta_{\text{worse}} = \beta_{\text{bad}}/z$ and $\beta_{\text{dead}} = \beta_{\text{worse}}/z = \beta_{\text{bad}}/z^2$, and subsequently evaluate $\mathcal{R}$ (under both the **clean@slow** and **clean@slower** policies) as the free parameter z varies from 1 to 20. The lower z is, the faster a **slower** (or **dead**) system can be cleaned. In particular, $z = 10$ corresponds to the default cleanup rates under **P2** without modifications. Therefore, $z \ll 10$ corresponds to a significant investment in improving cleanup speeds.

[16]For simplicity, we do not make claims about how much such improvements cost and to what extent they are feasible.

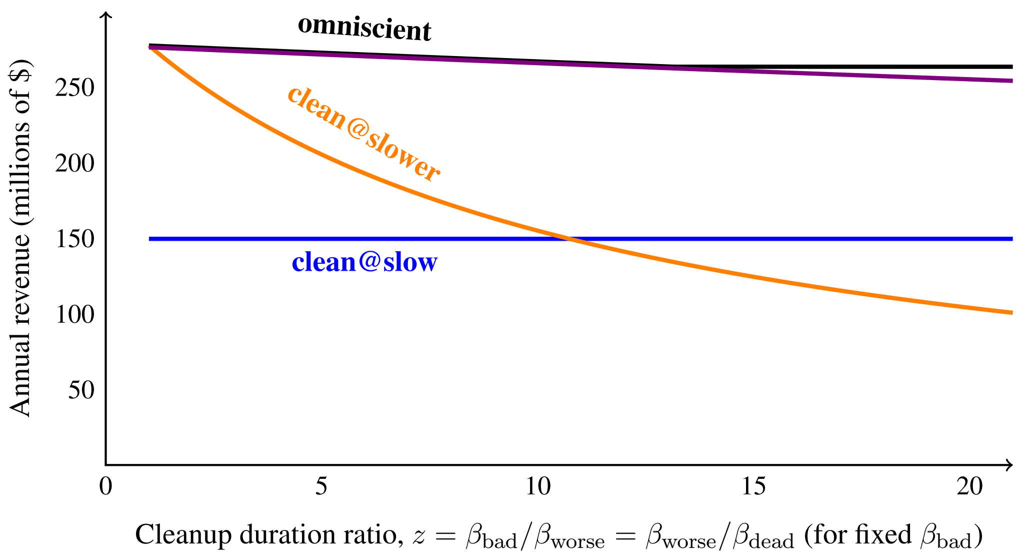


Figure 3.8: Revenue rate, $\mathcal{R}$, in millions of dollars per year under default parameters **P2**, as a function of z, where $z = \beta_{\text{bad}}/\beta_{\text{worse}} = \beta_{\text{worse}}/\beta_{\text{dead}}$, with β_{bad} kept fixed at the **P2** level of 10^{-2}. The **clean@slow** policy exhibits constant performance (as it only depends on the cleanup rate β_{bad}, which is fixed), while **clean@slower** exhibits a convex decline, outperforming **clean@slow** for lower values of z. The hypothetical **omniscient** policy outperforms all other policies. The unlabeled curve is a hypothetical policy that can gain most of the benefits of **omniscient** by using sequential cleanups.

Meanwhile, we quantify the *maximum possible benefits* from improving intrusion detection by also evaluating $\mathcal{R}$ under the hypothetical **omniscient** policy across the same range of values for z. Recall that the **omniscient** policy chooses an optimal subset of joint malware-performance states on which to initiate cleanups. The revenue rates are plotted in Figure 3.8.

We first observe that **clean@slow** exhibits a constant performance, because it depends on β_{bad}, and not on β_{worse} or β_{dead}; hence $\mathcal{R}$ is constant in z for the **clean@slow** policy. Meanwhile, **clean@slower** outperforms **clean@slow** for low values of z, exhibiting a convex decline. This shape is apparent across a wide range of realistic system parameters (and it can be proven to be a hyperbola). We also observe that at $z = 1$, $\mathcal{R}$ under **clean@slower** matches that under **omniscient**. To understand why, we note that **omniscient** behaves like **clean@slower** for low values of z, except that **omniscient** circumvents the pessimistic cleaning assumption (it cleans based on the actual *malware* state rather than the *performance* state). At $z = 1$, however, **omniscient** and **clean@slower** achieve the same revenue rate, because cleaning a **slower** system does not take less time if one is aware of the kind of malware present (if any) under the assumption that $\beta_{\text{bad}} = \beta_{\text{worse}}$, which is the case when $z = 1$. Eventually (visible in the Figure at $z \approx 13.2$), the **omniscient** policy will clean up a system as soon as it is **slower** or **bad**. We can conclude that intrusion detection can significantly improve the profitability of a system, but so can reducing the time required to cleanup more serious problems on a system. In this case, the benefits from *perfect* intrusion detection outweigh those from all but the most extreme improvements in cleanup speeds, suggesting that improving intrusion detection should be a higher priority.

In fact, it turns out that much of the benefit in the hypothetical **omniscient** policy is due to the fact that it is not bound by the pessimistic cleaning assumption (i.e., it does not need to implement a lengthy cleanup procedure if a **slower** system is not in the **worse** malware state). It can be tempting to mimic this advantage even when malware is not observable by cleaning a **slower** system by using a shorter cleanup (i.e., with cleaning rate β_{bad}). Then, if at the conclusion of that cleanup the system is still sluggish (observed to be in the **slower** state due to lingering malware that was not removed by the quick cleanup), one can implement a lengthier cleanup (i.e., with cleaning rate β_{worse}) that is guaranteed to remove the malware. The impressive performance of this "sequential cleanup policy" is shown by the unlabeled curve in Figure 3.8. Unfortunately, such a policy may not always be implementable in practice, as it might be a poor security practice to perform an insufficient cleanup procedure when there is risk of a serious infection. However, in contexts where such a policy can be considered sufficiently safe, it can be a great alternative to improving intrusion detection. For example, before formatting a system that appears to be potentially infected by malware, it may pay off to perform a quick reboot to see if the problem persists. In practitioners' terms, this is equivalent to "trying the easy solution first."[17]

We conclude that for our case study, intrusion detection is preferable to improving cleanup speeds (if one must choose only one and the two improvement costs are comparable) whenever near-perfect intrusion is possible, unless it is safe to use a sequence of progressively lengthier cleanups when dealing with a system in the **slower** state.

However, we must acknowledge that the **omniscient** policy is a *hypothetical* policy, and that

[17]Note that while this is an acceptable practice in several contexts and applications, it could be unimplementable for security reasons (e.g., performing a reboot in an infected host could permanently remove valuable forensics data).

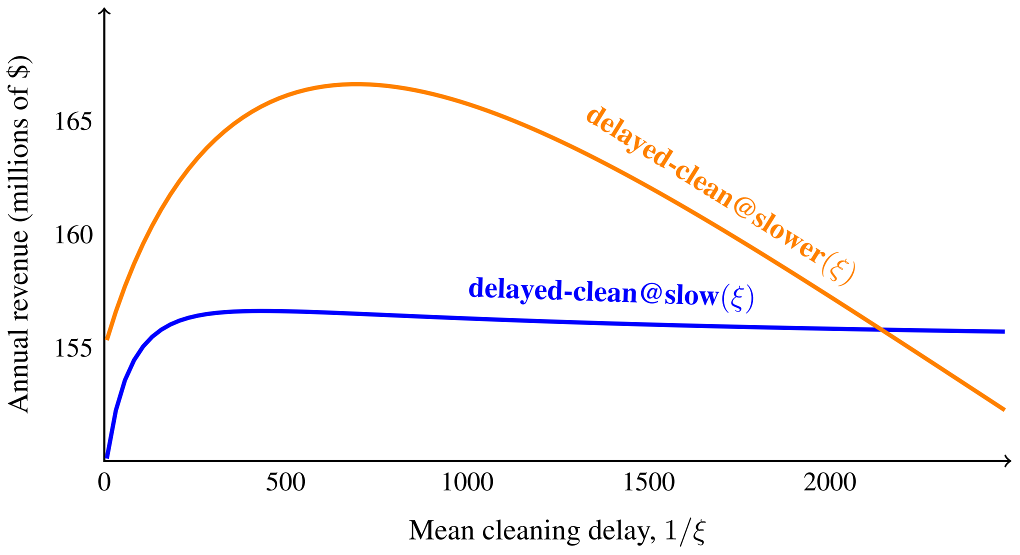


Figure 3.9: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P2**, as a function of the mean delay before initiating cleanup actions, $1/\xi$, both policies exhibit a benefit from introducing a small delay, but eventually attain a local maximum and subsequently decrease in profitability as $1/\xi$ increases.

perfect intrusion detection cannot exist in practice.

Should we act immediately upon a performance degradation event or delay our cleanup actions?

It is natural to ask whether a system should be cleaned as soon as one reaches a "target" performance degradation state, or if the cleanup procedure should be delayed once such a "target" state has been reached. The rationale for such a delay is that the service provider may be content with slower performance, but *not* with malware infections. In particular, since $\alpha_{\text{bad}} = 10\gamma_{\text{slow}}$ under **P2**, a transition to the **slow** state is unlikely to suggest that the system has been infected by **bad** malware, and one can persist in a **slow** system for a considerable period of time with little risk of being unknowingly infected. But how much of an improvement in $\mathcal{R}$ can we expect if we introduce such delays?

We explore the benefits of such delays by evaluating $\mathcal{R}$ under the **delayed-clean@slow**(ξ) and **delayed-clean@slower**(ξ) policies for the parameter set **P2**. Figure 3.9 shows a comparison of the revenue rates under these cleanup policies as a function of the mean cleanup delay, $1/\xi$.

We observe that for this parameter set, there is a significant benefit to implementing a delay under both policies. In fact, introducing such delays is beneficial in nearly all systems, except those where performance degradation and/or malware is so costly, that these costs dwarf the detrimental impact of frequent cleanups. Here, the best performing policy is **delayed-clean@slower**$(1/696)$, attaining a revenue rate of $\mathcal{R} \approx 166.64$ (in the units of millions of dollars per year). This revenue can be improved further by using the **hybrid-delay**$(1/678, 1/749)$ policy, which yields $\mathcal{R} \approx 167.84$. By using delays in both the

slow and **slower** states, we have a modest improvement of less than 1%, although such improvements can be more pronounced across a variety of parameter settings. Using deterministic delays can increase revenues even further. We will refer to the optimal hybrid delay policy a number of additional times, so for notational convenience, we let

$$\xi_1^* = 1/678 \qquad \text{and} \qquad \xi_2^* = 1/749.$$

One benefit of delaying cleanups, is of course decreasing the frequency of cleanup actions, while adding little additional risk of residing or entering a malware state. For example, if one has already transitioned to say the **slower** state, one has already effectively "paid" the sunk cost of a lengthier cleanup duration (which is unlikely to grow any longer if one imposes a reasonable delay, as transitions to the **dead** state are typically very low). In this case, one may as well decrease the frequency of cleanup procedures by spending additional time in the **slower** state. However, this is not the only benefit to implementing cleanup delays. When a change in *performance* level occurs, waiting times gradually increase over time, rather than increasing immediately. Therefore, if an average response time of less than t^* is "acceptable," (i.e. profitable, and worth operating, in the interest of engaging in less frequent cleanups) and one is transitioning from a performance state with a steady-state response time of $t_1 \ll t^*$ to one with $t_2 \gg t^*$, one can delay a cleanup event and still enjoy "acceptable" response times for some additional time while "spacing out" cleanup procedures. Hence, delaying cleanup procedures can even be beneficial in the case of visible malware.

We can conclude that in this case we should *not act immediately* upon a performance degradation event. Waiting for an appropriate amount of time (as given by our framework) can lead to significantly gains in revenue.

What do we gain from incorporating queue length information into cleanup decisions?

In the preceding discussion, the gradual transition from one steady-state response time to a higher steady-state response time is actually due to the gradual buildup of the queue. Consequently, the benefits of delaying cleanups motivates the consideration of *dynamic* cleanup policies that take the number of jobs into account in determining when to initiate a cleanup procedure? But how much can the service provider benefit from taking queue lengths into account?

We shed light on the potential benefits of dynamic policies by evaluating revenue rates for the special cases of dynamic policies that we call threshold policies. We use the best hybrid policy identified in the preceding section, **hybrid-delay**(ξ_1^*, ξ_2^*) as a baseline policy. We consider modifications of the baseline policy with various thresholds, Θ. Recall that the **hybrid-delay**$_{\geq\Theta}(\xi_1^*, \xi_2^*)$ family of policies are like the **hybrid-delay**(ξ_1^*, ξ_2^*) policy, except that when we are in the **slower** state, we only clean up the system after spending sufficient time *with the number of jobs in the system, j, meeting or exceeding the threshold* Θ. Figure 3.10 depicts the revenue rates under the **hybrid-delay**$_{\geq\Theta}(\xi_1^*, \xi_2^*)$ policies, for various choices of the threshold parameter Θ.

First, note that the baseline from the previous section corresponds to the policy with a threshold of $\Theta = 0$. Meanwhile, among the plotted policies,[18] an optimum revenue rate of $\mathcal{R} \approx 171.61$

[18]Naturally, we can expect to do better if we jointly optimize the delays ξ_1 and ξ_2 together with the threshold Θ, or consider dynamic policies beyond simple threshold policies. We do note, however, that allowing for two separate

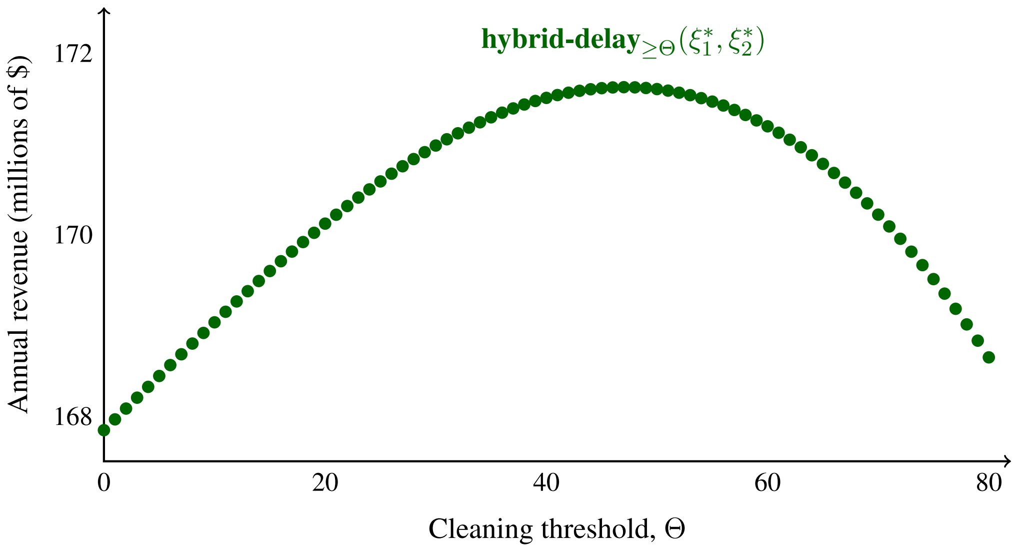


Figure 3.10: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P2**, as a function of the the threshold Θ, for the **hybrid-delay**$_{\geq\Theta}(\xi_1^*, \xi_2^*)$ family of policies, where cleanups in the **slower** state are only permitted if the number of jobs, j, exceeds the threshold Θ.

is achieved at the threshold $\Theta = 47$, representing an additional improvement of over 2% over the baseline policy. This non-negligible yet modest improvement highlights the power of dynamic policies and suggests that we *cannot* capture all of the benefits of dynamic policies by simply using cleaning delays. This realization also underscores the advantage of tracking the number of jobs in the system as opposed to treating the malware cleanup problem as a standard condition-based maintenance problem.

Recall that we also introduced the **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$ family of policies, which initiate a cleanup procedure in the **slower** state only when the number of jobs, j, falls at or *below* the threshold Θ. We would expect such policies (for small values of Θ) to perform poorly for the same reasons the preceding policies performed well.[19] These alternative threshold policies allow the system to persist in the **slower** state for far too long, as the system is rarely occupied by only a few jobs in the **slower** state.

Naturally, we ask if there exist scenarios where the **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$ policies outperform their **hybrid-delay**$_{\geq\Theta}(\xi_1, \xi_2)$ counterparts. As one can imagine, the **hybrid-delay**$_{\leq\Theta}(\xi_1, \xi_2)$ policies excel at minimizing the number of discarded jobs when a cleaning procedure is initiated. In fact, they limit this number to Θ per cleanup events triggered in the **slower** state. Unfortunately, it turns out that the costs associated with discarded jobs is often relatively insignificant. Even if hundreds of jobs are discarded at once, the number of jobs served between cleaning procedures may be orders of magnitude higher than this figure. However, one can imagine a modified setting where there is a much stronger desire to minimize discarded jobs.

Consider a variation of the model explored in this paper where each discarded job not only

thresholds (below which we cannot clean the system), one for the system in the **slow** state and the other for the system in the **slower** state did not appear to yield any benefits in this case.

[19]We have verified that this is the case under **P2** for the **hybrid-delay**$_{\leq\Theta}(\xi_1^*, \xi_2^*)$ family of policies.

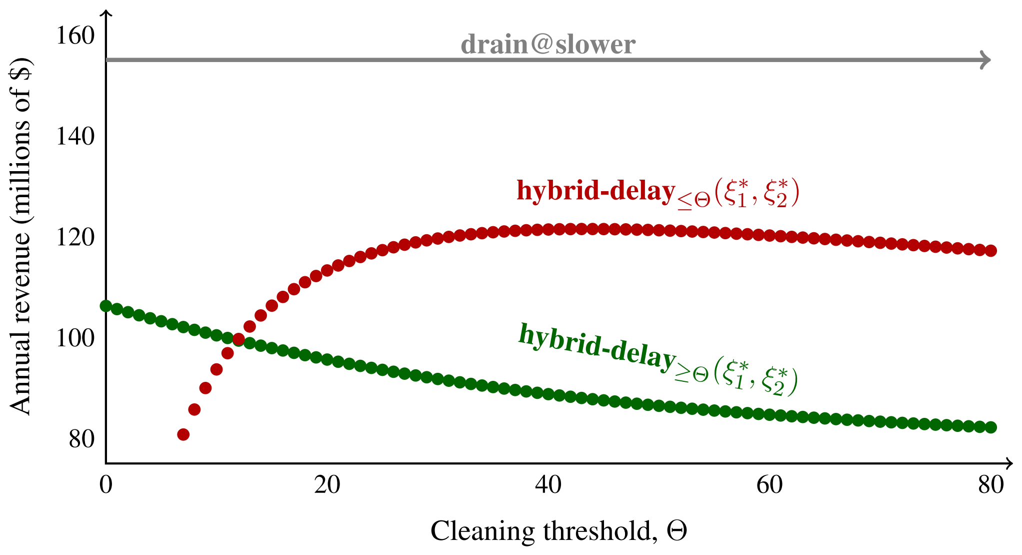


Figure 3.11: Revenue rate, $\mathcal{R}$, in millions of dollars per year, under the default parameter set **P2** with a job discarding penalty of $y = \$100$, as a function of the the threshold Θ, for the $\textbf{hybrid-delay}_{\geq\Theta}(\xi_1^*, \xi_2^*)$ and $\textbf{hybrid-delay}_{\leq\Theta}(\xi_1^*, \xi_2^*)$ families of policies. Also plotted is the revenue under the **drain@slower** policy (which is constant in Θ).

represents a missed opportunity for the service provider to collect q for serving an additional customer request, but also a penalty y (in dollars), associated with tarnished reputation or goodwill loss. That is, rather than interpreting the revenue rate as

$$\mathcal{R} = q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}},$$

we could interpret it as

$$\mathcal{R} = q\chi - c \cdot \mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}} - y\eta,$$

where we recall that η is the rate at which jobs are discarded.

Now let us consider the default parameter setting **P2**, except that we will let $y = \$100$, rather than $y = \$0$ (which is the case everywhere else in this paper). Note that this is an extreme value chosen for illustrative purposes; we are assuming that any given customer is only willing to pay \$1 for service, but that the service provider is somehow liable for an additional \$100 for every customer that is discarded from service (and rejecting a customer due to cleaning downtimes is without a cost, other than the missed opportunity to serve a customer). Such a setting may exist when extremely strict service level agreement contracts are in place.

We again use a baseline policy of $\textbf{hybrid-delay}(\xi_1^*, \xi_2^*)$, but this time we evaluate revenues for both the $\textbf{hybrid-delay}_{\geq\Theta}(\xi_1^*, \xi_2^*)$ and $\textbf{hybrid-delay}_{\leq\Theta}(\xi_1^*, \xi_2^*)$ families of policies. Due to the extreme value of y, we see in Figure 3.11 that $\mathcal{R}$ is decreasing in Θ for $\textbf{hybrid-delay}_{\geq\Theta}(\xi_1^*, \xi_2^*)$ and initially increasing (and subsequently decreasing) in Θ for $\textbf{hybrid-delay}_{\leq\Theta}(\xi_1^*, \xi_2^*)$, with the latter family of policies dominating for sufficiently high Θ. In fact, comparing the optimal policies (among the ones examined) from each family, $\textbf{hybrid-delay}_{\leq 44}(\xi_1^*, \xi_2^*)$ outperforms

hybrid-delay$_{\geq 0}(\xi_1^*, \xi_2^*)$ (in terms of $\mathcal{R}$) by over 14%. The strongest policy overall from these families, **hybrid-delay**$_{\leq 44}(\xi_1^*, \xi_2^*)$, allows for cleanups in the **slower** state only when there are no more than 44 jobs present in the system, while the strongest policy in the other family is actually the **hybrid-delay**(ξ_1^*, ξ_2^*) policy without any threshold. Meanwhile, the **drain@slow** policy outperforms all policies from these two families by a significant margin, indicating that admissions control is especially valuable in this context. The strong performance of the **hybrid-delay**$_{\leq \Theta}(\xi_1^*, \xi_2^*)$ family of policies together with the **drain@slow** policy suggests that cleanup procedures should only be undertaken when the queue length is relatively short in this particular setting. The common intuition that "the more highly utilized a system is, the more costly it is to take it offline" is justified in this setting.

The observation that both families of threshold policies can outperform the other depending on the setting suggests that even further gains are possible by considering more sophisticated dynamic policies. However, as previously stated, determining the optimal dynamic policy requires solving an intractable dynamic program. We are able to obtain results for these threshold policies by extending the non-repeating portion of the Markov chain when applying the CAP method. A similar approach can be used to evaluate other simple dynamic policies, but without a concrete way of identifying the best such policies in what is a highly multi-dimensional policy space.

In the following section, we investigate whether insights and recommendations, such as those presented in this section, can be obtained by circumventing exact analysis and instead relying on approximate analysis.

3.5 Approximate Analysis

In this section, we present a method for approximating $\mathcal{R}$ in the case of hidden malware[20] under the **hybrid-delay**(ξ_1, ξ_2) policy by viewing the system as a finite state CTMC, which does not explicitly track the number of jobs in the system. The approximation instead makes steady-state assumptions regarding the queueing dynamics at each joint malware-performance state to approximate $\mathbb{E}[N]$. We show how this approximation is derived, and proceed to show that this technique is often inadequate for evaluating cleaning policies and determining which policy is optimal. This observation highlights the advantages of the exact analysis presented in the previous section.

The first step in our approximation is determining the limiting probability distribution across the joint-malware performance states and cleanup states of the CTMC of interest (ignoring the number of jobs in the system). That is, we must determine the quantities $\pi_0, \pi_1, \ldots, \pi_8$, exactly (with states 0–8 defined in Section 3.4.2), but we may do so without using the CAP method. The CTMC governing the transitions between these states is shown in Figure 3.12, if we interpret the phases as merely being states, since we are no longer tracking the number of jobs in the system. We can determine these values exactly by solving the balance equations of the Markov chain:

[20]Similar techniques can be employed in the case of visible malware.

$$\begin{cases} (\alpha_{\text{bad}} + \gamma_{\text{slow}})\pi_0 = \beta_{\text{bad}}\pi_6 + \beta_{\text{worse}}\pi_7 + \beta_{\text{dead}}\pi_8 \\ (\alpha_{\text{bad}} + \gamma_{\text{slow}} + \xi_1)\pi_1 = \gamma_{\text{slow}}\pi_0 \\ (\alpha_{\text{worse}} + \gamma_{\text{slow}} + \xi_1)\pi_2 = \alpha_{\text{bad}}\pi_0 + \alpha_{\text{bad}}\pi_1 \\ (\alpha_{\text{bad}} + \xi_2)\pi_3 = \gamma_{\text{slow}}\pi_1 \\ (\alpha_{\text{worse}} + \xi_2)\pi_4 = \gamma_{\text{slow}}\pi_2 + \alpha_{\text{bad}}\pi_3 \\ (\alpha_{\text{dead}} + \xi_2)\pi_5 = \alpha_{\text{worse}}\pi_2 + \alpha_{\text{worse}}\pi_4 \\ (\beta_{\text{bad}})\pi_6 = \xi_1\pi_1 + \xi_1\pi_2 \\ (\beta_{\text{worse}})\pi_7 = \xi_2\pi_3 + \xi_2\pi_4 + \xi_2\pi_5 \\ (\beta_{\text{dead}})\pi_8 = \alpha_{\text{dead}}\pi_5 \\ \sum_{i=0}^{8}\pi_i = 1 \end{cases}$$

This finite linear system has a straightforward (if somewhat unwieldy) closed-form solution.

With these limiting probabilities determined exactly, we next turn our attention to *approximating* the steady-state expected number of jobs, $\mathbb{E}[N]$. We reformulate $\mathbb{E}[N]$ by conditioning on the current state (e.g., state 0, the **normal fast** state):

$$\mathbb{E}[N] = \sum_{i=0}^{8} \mathbb{E}[N|\text{state } i] \cdot \pi_i, \tag{3.7}$$

We then *approximate* the conditional expectation of N given a particular joint malware-performance (or cleanup) state by finding $\mathbb{E}[N]$ *as if the system would continue to be in this state forever*. That is, given a particular state, we approximate $\mathbb{E}[N]$ conditional on being in that malware state as being equal to the $\mathbb{E}[N]$ for an M/M/1 queue with arrival rate λ and the service rate μ corresponding to the service rate of the given state, which is given by the formula

$$\mathbb{E}[N]^{\text{M/M/1}} = \frac{\lambda}{\mu - \lambda}. \tag{3.8}$$

We observe that in state 0, the system is **fast**, in states 1–2 the system is **slow**, and in states 3–5, the system is **slower**, while in states 6–8 the system is undergoing a cleanup procedure where N is all 0. Hence, these facts, together with Equations (3.7) and (3.8) yield the approximation

$$\begin{aligned} \mathbb{E}[N] &= \sum_{i=0}^{8} \mathbb{E}[N|\text{state } i] \cdot \pi_i \\ &= \mathbb{E}[N|\textbf{fast}] \cdot \pi_0 + \mathbb{E}[N|\textbf{slow}] \cdot (\pi_1 + \pi_2) + \mathbb{E}[N|\textbf{slower}] \cdot (\pi_3 + \pi_4 + \pi_5) + 0 \\ &\approx \lambda \left(\frac{\pi_0}{\mu_{\text{fast}} - \lambda} + \frac{\pi_1 + \pi_2}{\mu_{\text{slow}} - \lambda} + \frac{\pi_3 + \pi_4 + \pi_5}{\mu_{\text{slower}} - \lambda} \right), \end{aligned} \tag{3.9}$$

in terms of $\pi_0, \ldots, \pi_5$.

Recall that $\chi = \lambda(1-\pi_{\text{clean}})-\eta = \lambda(1-\pi_6-\pi_7-\pi_8)$, where η is the rate at which customers are discarded. To approximate η and hence χ, we approximate the number of jobs in the system at the start of a cleanup procedure by using the steady-state number of jobs in the corresponding joint performance-malware state at which the cleanup procedure was initiated (i.e., we adapt the technique used to approximate $\mathbb{E}[N]$):

$$\begin{aligned}
\eta &= \mathbb{E}[N|\textbf{slow}] \cdot \xi_1(\pi_1+\pi_2) + \mathbb{E}[N|\textbf{slower}] \cdot (\xi_2(\pi_3+\pi_4+\pi_5) + \alpha_{\text{dead}}\pi_5) \\
&\approx \lambda\left(\frac{\xi_1(\pi_1+\pi_2)}{\mu_{\text{slow}}-\lambda} + \frac{\xi_2(\pi_3+\pi_4+\pi_5)+\alpha_{\text{dead}}\pi_5}{\mu_{\text{slower}}-\lambda}\right) \\
\chi &\approx \lambda\left(1-\pi_6-\pi_7-\pi_8-\frac{\xi_1(\pi_1+\pi_2)}{\mu_{\text{slow}}-\lambda} - \frac{\xi_2(\pi_3+\pi_4+\pi_5)+\alpha_{\text{dead}}\pi_5}{\mu_{\text{slower}}-\lambda}\right) \qquad (3.10)
\end{aligned}$$

Finally, we compute $\mathcal{R}$ by observing that states 2 and 4 correspond to the **bad** malware state, while state 5 corresponds to the **worse** state, which yields

$$\begin{aligned}
\mathcal{R} &= q\chi - c\cdot\mathbb{E}[N] - \ell_{\text{bad}}\pi_{\text{bad}} - \ell_{\text{worse}}\pi_{\text{worse}} \\
&\approx \lambda q\left(1-\pi_6-\pi_7-\pi_8-\frac{\xi_1(\pi_1+\pi_2)}{\mu_{\text{slow}}-\lambda} - \frac{\xi_2(\pi_3+\pi_4+\pi_5)+\alpha_{\text{dead}}\pi_5}{\mu_{\text{slower}}-\lambda}\right) \\
&\qquad - \lambda c\left(\frac{\pi_0}{\mu_{\text{fast}}-\lambda} + \frac{\pi_1+\pi_2}{\mu_{\text{slow}}-\lambda} + \frac{\pi_3+\pi_4+\pi_5}{\mu_{\text{slower}}-\lambda}\right) - \ell_{\text{bad}}(\pi_2+\pi_4) - \ell_{\text{worse}}(\pi_5)
\end{aligned}$$

The quality of this approximation depends on how closely the distribution of N, the number of jobs in the system, conditioned on being in a particular malware state, matches the *steady-state* distribution of N for that malware state given the assumption that one will always persist in that particular malware state. In particular, in approximating conditional distributions (e.g., N given one is in the **worse** state), our approximation does *not* take into account the distribution over N by which one first transitions to a particular malware state (e.g., is one more likely to transition to **worse** when one has 5 jobs in the system or when one has 20 jobs in the system?). Similarly, our approximation does not take into account how long one spends in that malware state; the more time one spends in a malware state, the more likely the distribution over N conditioned on being in that malware state begins to match the corresponding steady-state distribution. In particular, relatively high values of ξ_1 and/or ξ_2 can lead to a poor approximation, because such values will dictate short residence times in certain states. These shorter residence times may not provide sufficient time for N to "mix" to a distribution resembling its steady-state distribution for a particular joint malware-performance state, which will potentially result in significant errors.

We now turn toward investigating the predictive power of the approximation for $\mathcal{R}$. Figure 3.12 depicts the same scenario as Figure 3.9, but with additional curves showing the *approximated* revenue under these two policies. While the approximation for $\mathcal{R}$ under the **delayed-clean@slow**(ξ) family of policies is reasonable, the approximation for $\mathcal{R}$ under the **delayed-clean@slower**(ξ) family of policies is largely erroneous, especially as the approximation suggests that $\mathcal{R}$ is decreasing in $1/\xi$ across the entire parameter space, when in reality $\mathcal{R}$ is initially *increasing* in $1/\xi$. The reason for this is largely due to the reason as one of the benefits of delayed cleanups (and dynamic policies) that we discovered earlier: it takes time to "mix" from an acceptable steady-state to an unacceptable one. In this case, we have a steady-state response

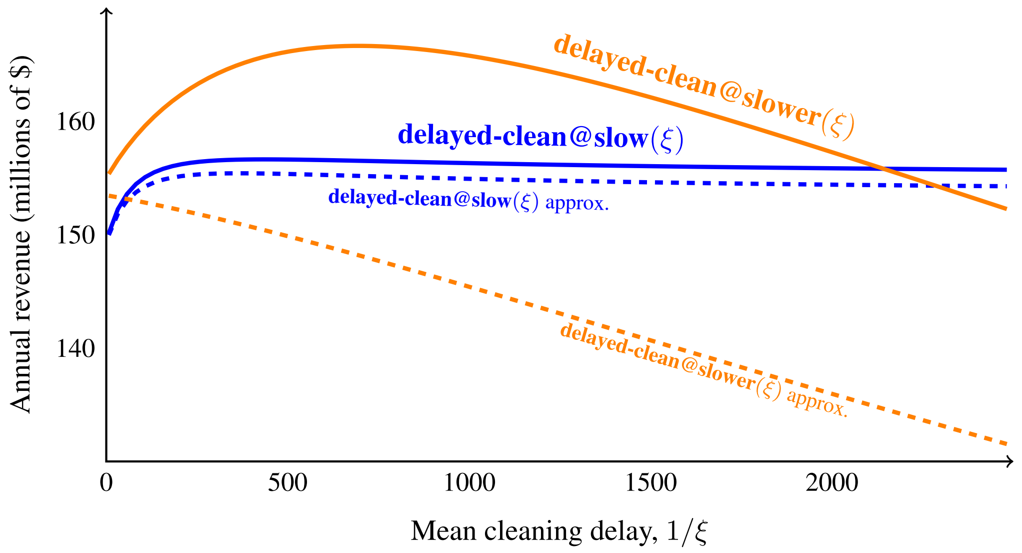


Figure 3.12: Revenue rate, $\mathcal{R}$, in millions of dollars per year under default parameter set **P2**, as a function of the mean delay before initiating cleanup actions, $1/\xi$. The solid lines show the actual $\mathcal{R}$ under the two policies, with the dotted lines showing the value of $\mathcal{R}$ approximated by the method presented in this section. The approximation for **clean@dead** especially is very poor.

time of 0.25 seconds in **slow**, and one of 10 seconds in **slower**, with the approximation "making this adjustment" immediately upon the transition, leading to an always decreasing approximation for $\mathcal{R}$ (as a function of $1/\xi$) under the **delayed-clean@slower**(ξ) policy.

The policy that is suggested as the strongest policy from the approximations (even considering the **hybrid-delay**(ξ_1, ξ_2) family of policies), is **delayed-clean@slow**$(1/367)$, which is approximated to attain a revenue rate of $\mathcal{R} \approx 155.39$, but in reality, does better, yielding $\mathcal{R} \approx 156.60$. This is still a loss of over 7% compared to the optimal hybrid delay policy of **hybrid-delay**$(1/678, 1/749)$. Across various test cases of systems experiencing frequent performance degradation and malware infections, revenue losses of 1–10% due to the use of the approximation are typical. Such differences can be very substantial, as the difference in *profits* will be even greater, once we account for the cost of running the service in the first place. The poor performance of the approximations based on finite state systems highlights one of the advantages of exact analysis. It is worth noting that the approximation errors are often much larger than 10%, but in cases where $\mathcal{R}$ is largely insensitive to the choice of policy (among "reasonable" policies), suboptimal decisions are not too costly. Moreover, on "timescales" where performance degradation, infection, and cleanup events happen very infrequently (e.g., if we consider **P2**, but scale all of the performance degradation, infection, and cleanup rates by a factor of 1/10 or less, while keeping other parameters fixed), the approximation error becomes negligible. Therefore, the appropriateness (or lack thereof) of this approximation depends strongly on the "timescale" a system is operating under.

3.6 Conclusion

This paper presents the first analytic study of the problem of determining when to clean up a customer-facing system that is susceptible to cyber attacks, such as malware. We consider both the case where the threat level of malware infecting the system is directly observable, and the case where malware is not detectable, and its presence can only be imperfectly inferred through monitoring the level of performance degradation experienced by the system.

Our contributions include proposing a Markovian model for the evolution of malware on a customer-facing system, and evaluating the performance of various cleanup policies using a combination of queueing-theoretic techniques. We find that in many realistic cases, one should *not* cleanup a system at the first indication of the presence of malware. In such cases, one should either wait for things to get worse, delay cleanup actions for some duration of time, or wait until the queue lengths exceeds a certain threshold. We also find that simpler approximations, which eschew tracking queue lengths in favor of steady-state queueing analysis with a finite state space, often lead to significantly suboptimal revenues (losing 1-10% of the revenue possible). The poor performance of these approximations, especially in the case of hidden malware highlights the importance of exact analysis for modeling this problem, and establishes that using techniques form the study of condition-based maintenance without incorporating queueing dynamics are insufficient.

One of our key discoveries is that the best policies are not necessarily those that act only when a new phenomenon is observed, and rather, there are significant benefits to delaying a response for some time after witnessing a performance degradation event. One reason that these delays are beneficial is that by delaying a cleanup, one reduces downtime, while enjoying acceptable waiting times before convergence to a new steady-state with unacceptable waiting times. Another way that one can harness the benefits of persisting in a system before reaching unacceptably high waiting times is to incorporate queue length information into the decision of when to clean a system, that is, to use a *dynamic* cleanup policy. Determining the optimal dynamic cleanup policy involves the analysis of an intractable Markov decision process. However, we are able to evaluate various *dynamic* policies, in particular threshold policies, and find that they provide a substantial improvement over their static (non-dynamic) counterparts. Therefore, we believe that the further analytic investigation of such dynamic policies is a natural direction for future work in this area.

Chapter 4

Routing when Servers are Strategic

4.1 Introduction

There is a broad and deep literature studying the scheduling and staffing of service systems that bridges operations research, applied probability, and computer science. This literature has had, and is continuing to have, a significant practical impact on the design of call centers (see, for example, the survey papers [55] and [7]), health care systems (see, for example, the recent book [76]), and large-scale computing systems (see, for example, the recent book [68]), among other areas. Traditionally, this literature on routing[1] and staffing has modeled the servers of the system as having fixed (possibly heterogeneous) service rates and then, given these rates, scheduling and staffing policies are proposed and analyzed. However, in reality, when the servers are *people*, the rate a server chooses to work can be, and often is, impacted by the routing and staffing policies used by the system.

For example, if requests are always scheduled to the "fastest" server whenever that server is available, then this server may have the incentive to slow her rate to avoid being overloaded with work. Similarly, if extra staff is always assigned to the division of a service system that is the busiest, then servers may have the incentive to reduce their service rates in order to ensure their division is assigned the extra staff. The previous two examples are simplistic; however, strategic behavior has been observed in practice in service systems. For example, empirical data from call centers shows many calls that last near 0 seconds [55]. This strategic behavior of the servers allowed them to obtain "rest breaks" by hanging up on customers – a rather dramatic means of avoiding being overloaded with work. For another example, academics are often guilty of strategic behavior when reviewing for journals. It is rare for reviews to be submitted before an assigned deadline since, if someone is known for reviewing papers very quickly, then they are likely to be assigned more reviews by the editor.

Clearly, the strategic behavior illustrated by the preceding examples can have a significant impact on the performance provided by a service system. One could implement a staffing or scheduling policy that is provably optimal under classical scheduling models, where servers are nonstrategic, and end up with far from optimal system performance as a result of undesirable strategic incentives created by the policy. Consequently, it is crucial for service systems to be

[1]Within this literature routing is often referred to as *scheduling*.

designed in a manner that provides the proper incentives for such “strategic servers”.

In practice, there are two approaches used for creating the proper incentives for strategic servers: one can either provide structured bonuses for employees depending on their job performance (performance-based payments) or one can provide incentives in how routing and staffing is performed that reward good job performance (incentive-aware scheduling). While there has been considerable research on how to design performance-based payments in the operations management and economics communities; the incentives created by scheduling and staffing policies are much less understood. In particular, *the goal of this chapter is to initiate the study of incentive-aware routing policies for strategic servers.*

The design of incentive-aware routing policies is important for a wide variety of service systems. In particular, in many systems performance-based payments such as bonuses are simply not possible, e.g., in service systems staffed by volunteers such as academic reviewing. Furthermore, many service systems do not use performance-based compensation schemes; for example, the 2005 benchmark survey on call center agent compensation in the U.S. shows that a large fraction of call centers pay a fixed hourly wage (and have no performance-based compensation) [10].

Even when performance-based payments are possible, the incentives created by routing policies impact the performance of the service system, and thus impact the success of performance-based payments. Further, since incentive-aware routing does not involve monetary payments (beyond a fixed employee salary), it may be less expensive to provide incentives through routing than through monetary bonuses. Additionally, providing incentives through scheduling and staffing eliminates many concerns about “unfairness” that stem from differential payments to employees.

Of course, the discussion above assumes that the incentives created by routing can be significant enough to impact the behavior. A priori it is not clear if they are, since simply changing the routing may not provide strong enough incentives to strategic servers to significantly change service rates, and thus system performance. It is exactly this uncertainty that motivates this chapter, which seeks to understand the impact of the incentives created by routing, and then to design incentive-aware routing policies that provide near-optimal system performance without the use of monetary incentives.

4.1.1 Contributions of This Chapter

This chapter makes two main contributions. We introduce a new model for the strategic behavior of servers in service systems and, additionally, we initiate the study routing in the context of strategic servers. Each of these contributions is described in the following.

Modeling Strategic Servers (Sections 4.2 and 4.3): The essential first step for an analysis of strategic servers is a model for server behavior that is simple enough to be analytically tractable and yet rich enough to capture the salient influences on how each server may choose her service rate. Our model is motivated by work in labor economics that identifies two main factors that impact the utility of agents: effort cost and idleness. More specifically, it is common in labor economics to model agents as having some “effort cost” function that models the decrease in utility which comes from an increase in effort [27]. Additionally, it is a frequent empirical observation that agents in service systems engage in strategic behavior to increase the amount of idle time they have ([55]). The key feature of the form of the utility we propose in Section 4.2 is that it captures the inherent trade-off between idleness and effort. In particular, a faster service rate would mean quicker completion of jobs and might result in a higher idle time, but it would also result in a higher effort cost. In Section 4.3 of this chapter, we apply our model in the context of an M/M/N system, where all servers behave strategically in a game setting.

Routing to Strategic Servers (Section 4.4): The second piece of this chapter studies the impact of strategic servers on the design of routing policies in multi-server service systems. When servers are not strategic, how to route (i.e., dispatch or schedule) jobs to servers in multi-server systems is well understood. In particular, the most commonly proposed policies for this setting include Fastest Server First (FSF), which dispatches arriving jobs to the idle server with the fastest service rate; Longest Idle Server First (LISF), which dispatches jobs to the server that has been idle for the longest period of time; and Random, which dispatches the job to each idle server with equal probability. When strategic servers are not considered, FSF is the natural choice for reducing the mean response time (though it is not optimal in general; see [39, 97]). However, in the context of strategic servers the story changes. In particular, we prove that FSF has no symmetric equilibria when strategic servers are considered, even when there are just two servers. Further, we prove that LISF, a commonly suggested policy for call centers due to its fairness properties, has the same, unique, symmetric equilibrium as random dispatching. In fact, we prove that there is a large *policy-space collapse* – all routing policies that are idle-time-order-based are equivalent in a very strong sense (Theorem 4.2).

With this in mind, one might suggest that Slowest Server First (SSF) would be a good dispatch policy, since it could incentivize servers to work fast; however, we prove that, like FSF, SSF has no symmetric equilibria (Theorem 4.4). However, by "softening" SSF's bias toward slow servers, we are able to identify policies in the two-server setting that are guaranteed to have a unique symmetric equilibrium and provide mean response times that are smaller than that under LISF and Random (Theorem 4.5).

A key message provided by the results described above is that routing policies must carefully balance two conflicting goals in the presence of strategic servers: making efficient use of the service capacity (e.g., by sending work to fast servers) while still incentivizing servers to work fast (e.g., by sending work to slow servers). While these two goals are inherently in conflict, our results show that it is possible to balance them in a way that provides improved performance over Random.

4.1.2 Related Work

While this chapter focuses on routing in the presence of strategic servers, this question is closely tied to how one must *staff* in the presence of strategic servers. As we have already described, the question of how to route and staff in many-server systems when servers have fixed, nonstrategic, service rates is well-studied. In general, this is a very difficult question, because the routing depends on the staffing and vice versa. However, when all the servers serve at the same rate, the routing question is moot. Then, [19] show that square-root staffing, first introduced in [47] and later formalized in [66], is economically optimal when both staffing and waiting costs are linear. Furthermore, square root staffing is remarkably robust: there is theoretical support for why it works so well for systems of moderate size ([85]), and it continues to be economically optimal both when abandonment is added to the M/M/N model ([57]) and when there is uncertainty in the arrival rate ([90]). Hence, to study the joint routing and staffing question for more complex systems, that include heterogeneous servers that serve at different rates and heterogeneous customers, many authors have assumed square root staffing and show how to optimize the routing for various objective functions (see, for example, [11, 14, 64, 122, 123]). In relation to this body of work, this chapter shows that scheduling and routing results for classical many-server systems that assume fixed service rates must be revisited when servers exhibit strategic behavior because classical routing policies may no longer be feasible in the strategic setting (see Section 4.4).

Importantly, the Fastest Server First routing policy mentioned earlier has already been recognized to be potentially problematic because it may be perceived as "unfair". The issue from an operational standpoint is that there is strong indication in the human resource management literature that the perception of fairness affects employee performance (see [36, 37]). This has motivated the analysis of "fair" routing policies that, for example, equalize the cumulative server idleness [15, 111], and the desire to find an optimal "fair" routing policy [12, 127]. Another approach is to formulate a model in which the servers choose their service rate in order to balance their desire for idle time (which is obtained by working faster) and the exertion required to serve faster. This leads to a non-cooperative game for an M/M/N queue in which the servers act as strategic players that selfishly maximize their utility.

Finally, the literature that is, perhaps, most closely related to this chapter is the literature on queueing games, which is surveyed in [74]. The bulk of this literature focuses on the impact of customers acting strategically (e.g., deciding whether to join and which queue to join) on queueing performance. Still, there is a body of work within this literature that considers settings where servers can choose their service rate, e.g., [25, 26, 60, 86]. However, in all of the aforementioned papers, there are two servers that derive utility from some monetary compensation per job or per unit of service that they provide. In contrast, our work considers servers that derive utility from idle time while paying a cost for exerting effort. The idea that servers value idle time is most similar to the setting in [59], but that paper restricts its analysis to a two server model. Perhaps the closest previous work to the current paper in analysis spirit is [8], which characterizes approximate equilibria in a market with many servers that compete on price and service level. However, this is similar in theme to [25, 86] in the sense that they consider servers as competing firms in a market. This contrasts with the current chapter, where our focus is on competition between servers *within the same firm*.

4.2 A Model for Strategic Servers

The objective of this chapter is to initiate an investigation into the effects of strategic servers on classical management decisions in service systems, e.g., staffing and routing. We start by, in this section, describing formally our model for the behavior of a strategic server.

The term "strategic server" could be interpreted in many ways depending on the server's goal. Thus, the key feature of the model is the utility function for a strategic server. Our motivation comes from a service system staffed by people who are paid a fixed wage, independent of performance. In such settings, one may expect two key factors to have a first-order impact on the experience of the servers: the amount of effort they put forth and the amount of idle time they have.

Thus, a first-order model for the utility of a strategic server is to linearly combine the cost of effort with the idle time of the server. This gives the following form for the utility of server i in a service system with N servers:

$$U_i(\boldsymbol{\mu}) = I_i(\boldsymbol{\mu}) - c(\mu_i),\ i \in \{1, \ldots, N\}, \tag{4.1}$$

where $\boldsymbol{\mu}$ is a vector of the rate of work chosen by each server (i.e., the service rate vector), $I_i(\boldsymbol{\mu})$ is the time-average idle time experienced by server i given the service rate vector $\boldsymbol{\mu}$, and $c(\mu_i)$ is the effort cost of server i. We take c to be an increasing, convex function which is the same for all servers. We assume that the strategic behavior of servers (choosing a utility-maximizing service rate) is independent of the state of the system and that the server has complete information about the steady state properties of the system when choosing a rate, i.e., they know the arrival rate, scheduling policy, staffing policy, etc., and thus can optimize $U_i(\boldsymbol{\mu})$.

The key feature of the form of the utility in (4.1) is that it captures the inherent trade-off between idleness and effort. The idleness, and hence the utility, is a steady state quantity. In particular, a faster service rate would mean quicker completion of jobs and might result in higher idle time in steady state, but it would also result in a higher effort cost. This trade-off then creates a difficult challenge for staffing and routing in a service system. To increase throughput and decrease response times, one would like to route requests to the fastest servers, but by doing so the utility of servers decreases, making it less desirable to maintain a fast service rate. Our model should be interpreted as providing insight into the *systemic* incentives created by scheduling and staffing policies rather than the *transitive* incentives created by the stochastic behavior of the system.

Our focus in this chapter will be to explore the consequences of strategic servers for staffing and routing in large service systems, specifically, in the $M/M/N$ setting. However, the model is generic and can be studied in non-queueing contexts as well.

To quickly illustrate the issues created by strategic servers, a useful example to consider is that of a $M/M/1$ queue with a strategic server.

Example 4.1 (The M/M/1 queue with a strategic server)**.** *In a classic $M/M/1$ system, jobs arrive at rate λ into a queue with an infinite buffer, where they wait to obtain service from a single server having fixed service rate μ. When the server is strategic, instead of serving at a fixed rate μ, the server chooses her service rate $\mu > \lambda$ in order to maximize the utility in* (4.1)*. To understand what service rate will emerge, recall that in a $M/M/1$ queue with $\mu > \lambda$ the steady state fraction of time that the server is idle is given by $I(\mu) = 1 - \frac{\lambda}{\mu}$. Substituting this expression into (4.1) means that the utility of the server is given by the following concave function:*

$$U(\mu) = 1 - \frac{\lambda}{\mu} - c(\mu).$$

We now have two possible scenarios. First, suppose that $c'(\lambda) < 1/\lambda$, so that the cost function does not increase too fast. Then, $U(\mu)$ attains a maximum in (λ, ∞) at a unique point $\mu^\star$, which is the optimal (utility maximizing) operating point for the strategic server. Thus, a stable operating point emerges, and the performance of this operating point can be derived explicitly when a specific form of a cost function is considered.

On the other hand, if $c'(\lambda) \geq 1/\lambda$, then $U(\mu)$ is strictly decreasing in (λ, ∞) and hence does not attain a maximum in this interval. We interpret this case to mean that the server's inherent skill level (as indicated by the cost function) is such that the server must work extremely hard just to stabilize the system, and therefore should not have been hired in the first place.

For example, consider the class of cost functions $c(\mu) = c_E \mu^p$. If $c(\lambda) < \frac{1}{p}$, then $\mu^\star$ solves $\mu^\star c(\mu^\star) = \frac{\lambda}{p}$, which gives $\mu^\star = \left(\frac{\lambda}{c_E p}\right)^{\frac{1}{p+1}} > \lambda$. On the other hand, if $c(\lambda) \geq \frac{1}{p}$, then $U(\mu)$ is strictly decreasing in (λ, ∞) and hence does not attain a maximum in this interval.

Before moving on to the analysis of the $M/M/N$ model with strategic servers, it is important to point out that the model we study focuses on a linear trade-off between idleness and effort. There are certainly many generalizations that are interesting to study in future work. One particularly interesting generalization would be to consider a concave (and increasing) function of idle time in the utility function, since it is natural that the gain from improving idle time from 10% to 20% would be larger than the gain from improving idle time from 80% to 90%. A preliminary analysis highlights that the results in this chapter would not qualitatively change in this context.[2]

4.3 The M/M/N Queue with Strategic Servers

Our focus in this chapter is on routing decisions in service systems, and so we adopt a classical model of this setting, the M/M/N, and adjust it by considering strategic servers, as described in Section 4.2. The analysis of routing policies is addressed in Section 4.4, but before moving on to the question of routing, we start by formally introducing the M/M/N model, and performing some preliminary analysis that is useful both in the context of staffing and routing.

In a M/M/N queue, customers arrive to a service system having N servers according to a Poisson process with rate λ. Delayed customers (those that arrive to find all servers busy) are served according to the First-Come-First-Served (FCFS) discipline. Each server is fully capable of handling any customer's service requirements. The time required to serve each customer is independent and exponential, and has a mean of one time unit when the server works at rate one. However, each server strategically chooses her service rate to maximize her own (steady state) utility, and so it is not a priori clear what the system service rates will be.

In this setting, the utility functions that the servers seek to maximize are given by

$$U_i(\boldsymbol{\mu}; \lambda, N, R) = I_i(\boldsymbol{\mu}; \lambda, N, R) - c(\mu_i), \qquad i \in \{1, \ldots, N\}, \tag{4.2}$$

where $\boldsymbol{\mu}$ is the vector of service rates, λ is the arrival rate, N is the number of servers (staffing level), and R is the routing policy. $I_i(\boldsymbol{\mu}; \lambda, N, R)$ is the steady state fraction of time that server i is idle. $c(\mu)$ is an increasing, convex function with $c'''(\mu) \geq 0$, that represents the server effort cost.

Note that, as compared with (4.1), we have emphasized the dependence on the arrival rate λ, staffing level N, and routing policy of the system, R. In the remainder of this article, we

[2]Specifically, if $g(I_i(\boldsymbol{\mu}))$ replaces $I_i(\boldsymbol{\mu})$ in (4.1), all the results in Section 4.3 characterizing equilibria service rates are maintained so long as $g''' < 0$, except for Theorem **??**, whose sufficient condition would have to be adjusted to accommodate g.

expose or suppress the dependence on these additional parameters as relevant to the discussion. In particular, note that the idle time fraction I_i (and hence, the utility function U_i) in (4.2) depends on how arriving customers are routed to the individual servers.

There are a variety of routing policies that are feasible for the system manager. In general, the system manager may use information about the order in which the servers became idle, the rates at which servers have been working, etc. This leads to the possibility of using simple policies such as Random, which chooses an idle server to route to uniformly at random, as well as more complex policies such as Longest/Shortest Idle Server First (LISF/SISF) and Fastest/Slowest Server First (FSF/SSF). We study the impact of this decision in detail in Section 4.4.

Given the routing policy chosen by the system manager and the form of the server utilities in (4.2), the situation that emerges is a competition among the servers for the system idle time. In particular, the routing policy yields a division of idle time among the servers, and both the division and the amount of idle time will depend on the service rates chosen by the servers.

As a result, the servers can be modeled as strategic players in a noncooperative game, and thus the operating point of the system is naturally modeled as an equilibrium of this game. In particular, a Nash equilibrium of this game is a set of service rates $\boldsymbol{\mu}^\star$, such that,

$$U_i(\mu_i^\star, \boldsymbol{\mu}_{-i}^\star; R) = \max_{\mu_i > \frac{\lambda}{N}} U_i(\mu_i, \boldsymbol{\mu}_{-i}^\star; R), \tag{4.3}$$

where $\boldsymbol{\mu}_{-i}^\star = (\mu_1^\star, \ldots, \mu_{i-1}^\star, \mu_{i+1}^\star, \ldots, \mu_N^\star)$ denotes the vector of service rates of all the servers except server i. Note that we exogenously impose the (symmetric) constraint that each server must work at a rate strictly greater than $\frac{\lambda}{N}$ in order to define a product action space that ensures the stability of the system.[3] Such a constraint is necessary to allow steady state analysis, and does not eliminate any feasible symmetric equilibria. We treat this bound as exogenously fixed, however in some situations a system manager may wish to impose quality standards on servers, which would correspond to imposing a larger lower bound (likely with correspondingly larger payments for servers). Investigating the impact of such quality standards is an interesting topic for future work.

Our focus in this chapter is on symmetric Nash equilibria. With a slight abuse of notation, we say that $\mu^\star$ is a symmetric Nash equilibrium if $\boldsymbol{\mu}^\star = (\mu^\star, \ldots, \mu^\star)$ is a Nash equilibrium (solves (4.3)). Throughout, the term "equilibrium service rate" means a symmetric Nash equilibrium service rate.

We focus on symmetric Nash equilibria for two reasons. First, because the agents we model intrinsically have the same skill level (as quantified by the effort cost functions), a symmetric equilibrium corresponds to a fair outcome. As we have already discussed, this sort of fairness is often crucial in service organizations ([12, 36, 37]). A second reason for focusing on symmetric

[3]One can imagine that servers, despite being strategic, would endogenously stabilize the system. To test this, one could study a related game where the action sets of the servers are $(0, \infty)$. Then, the definition of the idle time $I_i(\boldsymbol{\mu})$ must be extended into the range of $\boldsymbol{\mu}$ for which the system is overloaded; a natural way to do so is to define it to be zero in this range, which would ensure continuity at $\boldsymbol{\mu}$ for which the system is critically loaded. However, it is not differentiable there, which necessitates a careful piecewise analysis. A preliminary analysis indicates that in this scenario, no $\mu \in \left(0, \frac{\lambda}{N}\right]$ can ever be a symmetric equilibrium, and then, the necessary and sufficient condition of Theorem **??** would become $U(\mu^\star, \mu^\star) \geq \lim_{\mu_1 \to 0+} U(\mu_1, \mu^\star)$, which is more demanding than (**??**) (e.g., it imposes a finite upper bound on $\mu^\star$), but not so much so that it disrupts the staffing results that rely on this theorem (e.g., Lemma **??** still holds).

equilibria is that analyzing symmetric equilibria is already technically challenging, and it is not clear how to approach asymmetric equilibria in the contexts that we consider. Note that we do not rule out the existence of asymmetric equilibria; in fact, they likely exist, and it would be interesting to study whether they lead to better or worse system performance than their symmetric counterparts.

4.4 Routing to Strategic Servers

Thus far we have focused our discussion on staffing, assuming that jobs are routed randomly to servers when there is a choice. Of course, the decision of how to route jobs to servers is another crucial aspect of the design of service systems. As such, the analysis of routing policies has received considerable attention in the queueing literature, when servers are not strategic. In this section, we begin to investigate the impact of strategic servers on the design of routing policies.

In the classical literature studying routing when servers are nonstrategic, a wide variety of policies have been considered. These include "rate-based policies" such as Fastest Server First (FSF) and Slowest Server First (SSF); as well as "idle-time-order-based policies" such as Longest Idle Server First (LISF) and Shortest Idle Server First (SISF). Among these routing policies, FSF is a natural choice to minimize the mean response time (although, as noted in the Introduction, it is not optimal in general). This leads to the question: how does FSF perform when servers are strategic? In particular, does it perform better than the Random routing that we have so far studied?

Before studying optimal routing to improve performance, we must first answer the following even more fundamental question: what routing policies admit symmetric equilibria? This is a very challenging goal, as can be seen by the complexity of the analysis for the $M/M/N$ under Random routing. This section provides a first step towards that goal.

The results in this section focus on two broad classes of routing policies *idle-time-order-based policies* and *rate-based policies*, which are introduced in turn in the following.

4.4.1 Idle-Time-Order-Based Policies

Informally, idle-time-order-based policies are those routing policies that use only the rank ordering of when servers last became idle in order to determine how to route incoming jobs. To describe the class of idle-time-order-based policies precisely, let $\mathcal{I}(t)$ be the set of servers idle at time $t > 0$, and, when $\mathcal{I}(t) \neq \emptyset$, let $\boldsymbol{s}(t) = (s_1, \ldots, s_{|\mathcal{I}(t)|})$ denote the ordered vector of idle servers at time t, where server s_j became idle before server s_k whenever $j < k$. For $n \geq 1$, let $\mathcal{P}_n = \Delta(\{1, \ldots, n\})$ denote the set of all probability distributions over the set $\{1, \ldots, n\}$. An idle-time-order-based routing policy is defined by a collection of probability distributions $\boldsymbol{p} = \{p^S\}_{S \in 2^{\{1,2,\ldots,N\}} \backslash \emptyset}$, such that $p^S \in \mathcal{P}_{|S|}$, for all $S \in 2^{\{1,2,\ldots,N\}} \backslash \emptyset$. Under this policy, at time t, the next job in queue is assigned to idle server s_j with probability $p^{\mathcal{I}(t)}(j)$. Examples of idle-time-order-based routing policies are as follows.

1. *Random.* An arriving customer that finds more than one server idle is equally likely to be routed to any of those servers. Then, $p^S = (1/|S|, \ldots, 1/|S|)$ for all $S \in 2^{\{1,2,\ldots,N\}} \backslash \emptyset$.

2. *Weighted Random.* Each such arriving customer is routed to one of the idle servers with probabilities that may depend on the order in which the servers became idle. For example, if

$$p^S(j) = \frac{|S| + 1 - j}{\sum_{n=1}^{|S|} n}, \; j \in S, \; \text{for } s_j \in S, \; \text{for all } S \in 2^{\{1,2,\dots,N\}} \backslash \emptyset,$$

then the probabilities are decreasing according to the order in which the servers became idle. Note that $\sum_j p^S(j) = \frac{|S|(|S|+1) - \frac{1}{2}|S|(|S|+1)}{\frac{1}{2}|S|(|S|+1)} = 1$.

3. *Longest Idle Server First (Shortest Idle Server First).* Each such arriving customer is routed to the server that has idled the longest (idled the shortest). Then, $p^S = (1, 0, \dots, 0)$ ($p^S = (0, \dots, 0, 1)$) for all $S \subseteq \{1, 2, \dots, N\}$.

Policy-Space Collapse.

Surprisingly, it turns out that all idle-time-order-based policies are "equivalent" in a very strong sense — they all lead to the same steady state probabilities, resulting in a remarkable *policy-space collapse* result, which we discuss in the following.

Fix R to be some idle-time-order-based routing policy, defined through the collection of probability distributions $\boldsymbol{p} = \{p^S\}_{\emptyset \neq S \subseteq \{1,2,\dots,N\}}$. The states of the associated continuous time Markov chain are defined as follows:

- State B is the state where all servers are busy, but there are no jobs waiting in the queue.
- State $\boldsymbol{s} = (s_1, s_2, \dots, s_{|\mathcal{I}|})$ is the ordered vector of idle servers $\mathcal{I}$. When $\mathcal{I} = \emptyset$, we identify the empty vector $\boldsymbol{s}$ with state B.
- State m ($m \geq 0$) is the state where all servers are busy and there are m jobs waiting in the queue (i.e., there are $N + m$ jobs in the system). We identify state 0 with state B.

When all servers are busy, there is no routing, and so the system behaves exactly as an $M/M/1$ queue with arrival rate λ and service rate $\mu_1 + \cdots + \mu_N$. Then, from the local balance equations, the associated steady state probabilities π_B and π_m for $m = 0, 1, 2, \dots$, must satisfy

$$\pi_m = (\lambda/\mu)^m \pi_B \text{ where } \mu = \sum_{j=1}^{N} \mu_j. \tag{4.4}$$

One can anticipate that the remaining steady state probabilities satisfy

$$\pi_{\boldsymbol{s}} = \pi_B \prod_{s \in \mathcal{I}} \frac{\mu_s}{\lambda} \quad \text{for all } \boldsymbol{s} = (s_1, s_2, \dots, s_{|\mathcal{I}|}) \text{ with } |\mathcal{I}| > 0, \tag{4.5}$$

and the following theorem verifies this by establishing that the detailed balance equations are satisfied.

Theorem 4.2. *All idle-time-order-based policies have the steady state probabilities that are uniquely determined by (4.4)-(4.5), together with the normalization constraint that their sum is one.*

Proof. It is sufficient to verify the detailed balance equations. For reference, it is helpful to refer to Figure 4.1, which depicts the relevant portion of the Markov chain. We require the following additional notation. For all $\mathcal{I} \subseteq \{1, 2, \dots, N\}$, all states $\boldsymbol{s} = (s_1, s_2, \dots, s_{|\mathcal{I}|})$, all servers $s' \in \{1, 2, \dots, N\} \backslash \mathcal{I}$, and integers $j \in \{1, 2, \dots, |\mathcal{I}| + 1\}$, we define the state $\boldsymbol{s}[s', j]$ by

$$\boldsymbol{s}[s', j] \equiv (s_1, s_2, \dots, s_{j-1}, s', s_j, \dots, s_{|\mathcal{I}|}).$$

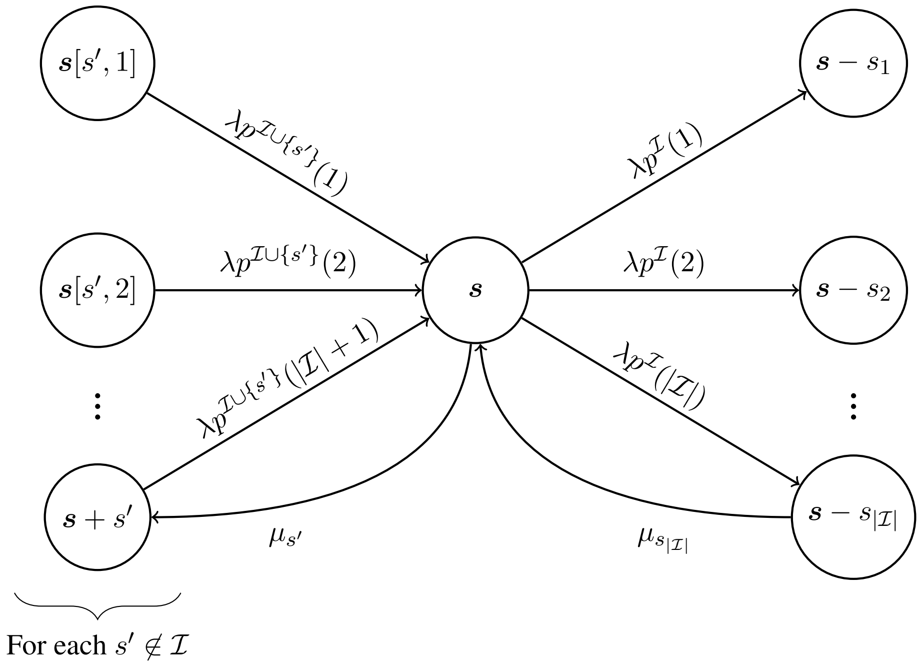

$\boldsymbol{s}[s',1]$
$\boldsymbol{s}[s',2]$
$\boldsymbol{s}+s'$
$\boldsymbol{s}$
$\boldsymbol{s}-s_1$
$\boldsymbol{s}-s_2$
$\boldsymbol{s}-s_{|\mathcal{I}|}$
$\lambda p^{\mathcal{I}\cup\{s'\}}(1)$
$\lambda p^{\mathcal{I}\cup\{s'\}}(2)$
$\lambda p^{\mathcal{I}\cup\{s'\}}(|\mathcal{I}|+1)$
$\lambda p^{\mathcal{I}}(1)$
$\lambda p^{\mathcal{I}}(2)$
$\lambda p^{\mathcal{I}}(|\mathcal{I}|)$
$\mu_{s'}$
$\mu_{s_{|\mathcal{I}|}}$
For each $s' \notin \mathcal{I}$

Figure 4.1: Snippet of the Markov chain showing the rates into and out of state $\boldsymbol{s} = (s_1, \dots, s_{|\mathcal{I}|})$. For convenience, we use $\boldsymbol{s} - s_j$ to denote the state $(s_1, s_2, \dots, s_{j-1}, s_{j+1}, \dots, s_{|\mathcal{I}|})$ and $\boldsymbol{s} + s'$ to denote the state $\boldsymbol{s}[s', |\mathcal{I}| + 1] = (s_1, s_2, \dots, s_{|\mathcal{I}|}, s')$.

We first observe that:

$$\begin{aligned}
\text{Rate into state } \boldsymbol{s} \text{ due to an arrival} &= \lambda \sum_{s' \notin \mathcal{I}} \sum_{j=1}^{|\mathcal{I}|+1} \pi_{\boldsymbol{s}[s',j]} p^{\mathcal{I}\cup\{s'\}}(j) \\
&= \lambda \sum_{s' \notin \mathcal{I}} \sum_{j=0}^{|\mathcal{I}|} \frac{\mu_{s'} \pi_B}{\lambda} \prod_{s \in \mathcal{I}} \left(\frac{\mu_s}{\lambda} \right) p^{\mathcal{I}\cup\{s'\}}(j) \\
&= \sum_{s' \notin \mathcal{I}} \mu_{s'} \pi_B \prod_{s \in I} \frac{\mu_s}{\lambda} = \sum_{s' \notin \mathcal{I}} \mu_{s'} \pi_{\boldsymbol{s}} \\
&= \text{Rate out of state } \boldsymbol{s} \text{ due to a departure.}
\end{aligned}$$

Then, to complete the proof, we next observe that for each $s' \notin \mathcal{I}$:

$$\begin{aligned}
\text{Rate into state } \boldsymbol{s} \text{ due to a departure} &= \mu_{s_{|\mathcal{I}|}} \pi_{(s_1, s_2, \ldots, s_{|\mathcal{I}|-1})} \\
&= \mu_{s_{|\mathcal{I}|}} \pi_B \prod_{s \in \mathcal{I} \setminus \{s_{|\mathcal{I}|}\}} \frac{\mu_s}{\lambda} \\
&= \lambda \pi_B \prod_{s \in \mathcal{I}} \frac{\mu_s}{\lambda} = \lambda \pi_{\boldsymbol{s}} \\
&= \text{Rate out of state } \boldsymbol{s} \text{ due to an arrival.}
\end{aligned}$$

□

Theorem 4.2 is remarkable because there is no dependence on the collection of probability distributions $\boldsymbol{p}$ that define R. Therefore, it follows that all idle-time-order-based routing policies result in the same steady state probabilities. Note that, concurrently, a similar result has been discovered independently in the context of loss systems (see [65]).

In relation to our server game, it follows from Theorem 4.2 that all idle-time-order-based policies have the same equilibrium behavior as Random. This is because an equilibrium service rate depends on the routing policy through the server idle time vector $(I_1(\boldsymbol{\mu}; R), \ldots, I_N(\boldsymbol{\mu}; R))$, which can be found from the steady state probabilities in (4.4)-(4.5). As a consequence, if there exists (does not exist) an equilibrium service rate under Random, then there exists (does not exist) an equilibrium service rate under any idle-time-order-based policy. In summary, it is not possible to achieve better performance than under Random by employing any idle-time-order-based policy.

4.4.2 Rate-Based Policies

Informally, a rate-based policy is one that makes routing decisions using only information about the rates of the servers. As before, let $\mathcal{I}(t)$ denote the set of idle servers at time t. In a rate-based routing policy, jobs are assigned to idle servers only based on their service rates. We consider a parameterized class of rate-based routing policies that we term *r-routing policies* ($r \in \mathbb{R}$). Under an r-routing policy, at time t, the next job in queue is assigned to idle server $i \in \mathcal{I}(t)$ with probability

$$p_i(\boldsymbol{\mu}, t; r) = \frac{\mu_i^r}{\sum_{j \in \mathcal{I}(t)} \mu_j^r}$$

Notice that for special values of the parameter r, we recover well-known policies. For example, setting $r = 0$ results in Random; as $r \to \infty$, it approaches FSF; and as $r \to -\infty$, it approaches SSF.

In order to understand the performance of rate-based policies, the first step is to perform an equilibrium analysis, i.e., we need to understand what the steady state idle times look like under any r-routing policy. The following proposition provides us with the required expressions.

Proposition 4.3. *Consider a heterogeneous $M/M/2$ system under an r-routing policy, with arrival rate $\lambda > 0$ and servers 1 and 2 operating at rates μ_1 and μ_2 respectively. The steady state probability that server 1 is idle is given by:*

$$I_1^r(\mu_1,\mu_2) = \frac{\mu_1(\mu_1+\mu_2-\lambda)\left[(\lambda+\mu_2)^2+\mu_1\mu_2+\frac{\mu_2^r}{\mu_1^r+\mu_2^r}(\lambda\mu_1+\lambda\mu_2)\right]}{\mu_1\mu_2(\mu_1+\mu_2)^2+(\lambda\mu_1+\lambda\mu_2)\left[\mu_1^2+2\mu_1\mu_2-\frac{\mu_1^r}{\mu_1^r+\mu_2^r}(\mu_1^2-\mu_2^2)\right]+(\lambda\mu_1)^2+(\lambda\mu_2)^2},$$

and the steady state probability that server 2 is idle is given by $I_2^r(\mu_1,\mu_2) = I_1^r(\mu_2,\mu_1)$.

Proof. In order to derive the steady state probability that a server is idle, we first solve for the steady state probabilities of the $M/M/2$ system (with arrival rate λ and service rates μ_1 and μ_2 respectively) under an arbitrary probabilistic routing policy where a job that arrives to find an empty system is routed to server 1 with probability p and server 2 with probability $1-p$. Then, for an r-routing policy, we simply substitute $p = \frac{\mu_1^r}{\mu_1^r+\mu_2^r}$.

It should be noted that this analysis (and more) for 2 servers has been carried out by [91]. Prior to that, [115] carried out a partial analysis (by analyzing an r-routing policy with $r = 1$). However, we rederive the expressions using our notation for clarity.

The dynamics of this system can be represented by a continuous time Markov chain shown in Figure 4.2 whose state space is simply given by the number of jobs in the system, except when there is just a single job in the system, in which case the state variable also includes information about which of the two servers is serving that job. This system is stable when $\mu_1 + \mu_2 > \lambda$ and we denote the steady state probabilities as follows:

- π_0 is the steady state probability that the system is empty.
- $\pi_1^{(j)}$ is the steady state probability that there is one job in the system, served by server j.
- For all $k \geq 2$, π_k is the steady state probability that there are k jobs in the system.

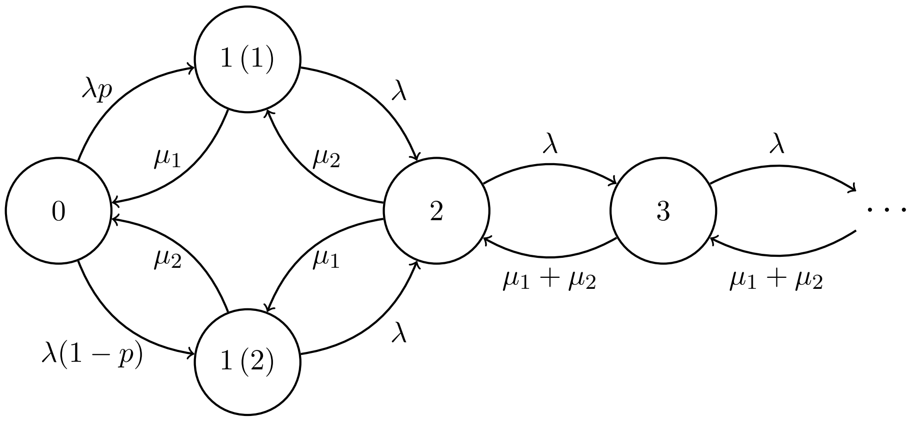


Figure 4.2: The $M/M/2$ Markov chain with probabilistic routing.

We can write down the balance equations of the Markov chain as follows:

$$\begin{aligned}
\lambda \pi_0 &= \mu_1 \pi_1^{(1)} + \mu_2 \pi_1^{(2)} \\
(\lambda + \mu_1)\pi_1^{(1)} &= \lambda p \pi_0 + \mu_2 \pi_2 \\
(\lambda + \mu_2)\pi_1^{(2)} &= \lambda(1-p)\pi_0 + \mu_1 \pi_2 \\
(\lambda + \mu_1 + \mu_2)\pi_2 &= \lambda \pi_1^{(1)} + \lambda \pi_1^{(2)} + (\mu_1 + \mu_2)\pi_3 \\
\forall k \geq 3\colon \quad (\lambda + \mu_1 + \mu_2)\pi_k &= \lambda \pi_{k-1} + (\mu_1 + \mu_2)\pi_{k+1},
\end{aligned}$$

yielding the following solution to the steady state probabilities:

$$\begin{aligned}
\pi_0 &= \frac{\mu_1\mu_2(\mu_1+\mu_2-\lambda)(\mu_1+\mu_2+2\lambda)}{\mu_1\mu_2(\mu_1+\mu_2)^2 + \lambda(\mu_1+\mu_2)(\mu_2^2 + 2\mu_1\mu_2 + (1-p)(\mu_1^2-\mu_2^2)) + \lambda^2(\mu_1^2+\mu_2^2)} \qquad (4.6)\\
\pi_1^{(1)} &= \frac{\lambda(\lambda + p(\mu_1+\mu_2))\pi_0}{\mu_1(\mu_1+\mu_2+2\lambda)} \\
\pi_1^{(2)} &= \frac{\lambda(\lambda + (1-p)(\mu_1+\mu_2))\pi_0}{\mu_2(\mu_1+\mu_2+2\lambda)}.
\end{aligned}$$

Consequently, the steady state probability that server 1 is idle is given by

$$I_1(\mu_1,\mu_2;p) = \pi_0 + \pi_1^{(2)} = \left(1 + \frac{\lambda(\lambda + (1-p)(\mu_1+\mu_2))}{\mu_2(\mu_1+\mu_2+2\lambda)}\right)\pi_0.$$

Substituting for π_0, we obtain

$$I_1(\mu_1,\mu_2;p) = \frac{\mu_1(\mu_1+\mu_2-\lambda)\left[(\lambda+\mu_2)^2 + \mu_1\mu_2 + (1-p)\lambda(\mu_1+\mu_2)\right]}{\mu_1\mu_2(\mu_1+\mu_2)^2 + \lambda(\mu_1+\mu_2)\left[\mu_2^2 + 2\mu_1\mu_2 + (1-p)(\mu_1^2-\mu_2^2)\right] + \lambda^2(\mu_1^2+\mu_2^2)}. \qquad (4.7)$$

Finally, for an r-routing policy, we let $p = \frac{\mu_1^r}{\mu_1^r+\mu_2^r}$ to obtain:

$$\begin{aligned}
I_1^r(\mu_1,\mu_2) &= I_1\left(\mu_1,\mu_2; p = \frac{\mu_1^r}{\mu_1^r+\mu_2^r}\right) \\
&= \frac{\mu_1(\mu_1+\mu_2-\lambda)\left[(\lambda+\mu_2)^2 + \mu_1\mu_2 + \frac{\mu_2^r}{\mu_1^r+\mu_2^r}\lambda(\mu_1+\mu_2)\right]}{\mu_1\mu_2(\mu_1+\mu_2)^2 + \lambda(\mu_1+\mu_2)\left[\mu_2^2 + 2\mu_1\mu_2 + \frac{\mu_2^r}{\mu_1^r+\mu_2^r}(\mu_1^2-\mu_2^2)\right] + \lambda^2(\mu_1^2+\mu_2^2)}.
\end{aligned}$$

By symmetry of the r-routing policy, it can be verified that $I_2^r(\mu_1,\mu_2) = I_1^r(\mu_2,\mu_1)$, completing the proof. □

Note that we restrict ourselves to a 2-server system for this analysis. This is due to the fact that there are no closed form expressions known for the resulting Markov chains for systems with more than 3 servers. It may be possible to extend these results to 3 servers using results from [103]; but, the expressions are intimidating, to say the least. However, the analysis for two servers is already enough to highlight important structure about the impact of strategic servers on policy design.

In particular, our first result concerns the FSF and SSF routing policies, which can be obtained in the limit when $r \to \infty$ and $r \to -\infty$ respectively. Recall that FSF is asymptotically optimal in the nonstrategic setting. Intuitively, however, it penalizes the servers that work the fastest by sending them more and more jobs. In a strategic setting, this might incentivize servers to decrease their service rate, which is not good for the performance of the system. One may wonder if by doing the opposite, that is, using the SSF policy, servers can be incentivized to increase their service rate. However, our next theorem (Theorem 4.4) shows that neither of these policies is useful if we are interested in symmetric equilibria.

Recall that our model for strategic servers already assumes an increasing, convex effort cost function with $c'''(\mu) \geq 0$. For the rest of this section, in addition, we assume that $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$.[4]

Theorem 4.4. *Consider an M/M/2 queue with strategic servers. Then, FSF and SSF do not admit a symmetric equilibrium.*

Proof. We first highlight that when all servers operate at the same rate $\mu \in \left(\frac{\lambda}{N}, \infty\right)$, both FSF and SSF are equivalent to Random routing. Henceforth, we refer to such a configuration as a symmetric operating point μ. In order to prove that there does not exist a symmetric equilibrium under either FSF or SSF, we show that at any symmetric operating point μ, any one server can attain a strictly higher utility by unilaterally setting her service rate to be slightly lower (in the case of FSF) or slightly higher (in the case of SSF) than μ.

We borrow some notation from the proof of Proposition 4.3 where we derived the expressions for the steady state probability that a server is idle when there are only 2 servers under any probabilistic policy, parameterized by a number $p \in [0, 1]$ which denotes the probability that a job arriving to an empty system is routed to server 1. Recall that $I_1(\mu_1, \mu_2; p)$ denotes the steady state probability that server 1 is idle under such a probabilistic policy, and the corresponding utility function for server 1 is $U_1(\mu_1, \mu_2; p) = I_1(\mu_1, \mu_2; p) - c(\mu_1)$. Then, by definition, the utility function for server 1 under FSF is given by:

$$U_1^{FSF}(\mu_1, \mu_2) = \begin{cases} U_1(\mu_1, \mu_2; p = 0) & , \ \mu_1 < \mu_2 \\ U_1\left(\mu_1, \mu_2; p = \frac{1}{2}\right) & , \ \mu_1 = \mu_2 \\ U_1(\mu_1, \mu_2; p = 1) & , \ \mu_1 > \mu_2. \end{cases}$$

Similarly, under SSF, we have:

$$U_1^{SSF}(\mu_1, \mu_2) = \begin{cases} U_1(\mu_1, \mu_2; p = 1) & , \ \mu_1 < \mu_2 \\ U_1\left(\mu_1, \mu_2; p = \frac{1}{2}\right) & , \ \mu_1 = \mu_2 \\ U_1(\mu_1, \mu_2; p = 0) & , \ \mu_1 > \mu_2. \end{cases}$$

Note that while the utility function under any probabilistic routing policy is continuous everywhere, the utility function under FSF or SSF is discontinuous at symmetric operating points. This discontinuity turns out to be the crucial tool in the proof. Let the two servers be operating

[4]The sufficient condition $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$ might seem rather strong, but it can be shown that it is necessary for the symmetric first order condition to have a unique solution. This is because, if $c'(\frac{\lambda}{2}) > \frac{1}{\lambda}$, then the function $\varphi(\mu)$, defined in (4.10), ceases to be monotonic, and as a result, for any given r, the first order condition $\varphi(\mu) = r$ could have more than one solution.

at a symmetric operating point μ. Then, it is sufficient to show that there exists $0 < \delta < \mu - \frac{\lambda}{2}$ such that

$$U_1^{FSF}(\mu - \delta, \mu) - U_1^{FSF}(\mu, \mu) > 0, \tag{4.8}$$

and

$$U_1^{SSF}(\mu + \delta, \mu) - U_1^{FSF}(\mu, \mu) > 0. \tag{4.9}$$

We show (4.8), and (4.9) follows from a similar argument. Note that

$$\begin{aligned} U_1^{FSF}(\mu - \delta, \mu) - U_1^{FSF}(\mu, \mu) &= U_1(\mu - \delta, \mu; p = 0) - U_1\left(\mu, \mu; p = \frac{1}{2}\right) \\ &= \big(U_1(\mu - \delta, \mu; p = 0) - U_1(\mu, \mu; p = 0)\big) \\ &\quad + \left(U_1(\mu, \mu; p = 0) - U_1\left(\mu, \mu; p = \frac{1}{2}\right)\right) \end{aligned}$$

Since the first difference, $U_1(\mu - \delta, \mu; p = 0) - U_1(\mu, \mu; p = 0)$, is zero when $\delta = 0$, and is continuous in δ, it is sufficient to show that the second difference, $U_1(\mu, \mu; p = 0) - U_1(\mu, \mu; p = \frac{1}{2})$, is strictly positive:

$$\begin{aligned} U_1(\mu, \mu; p = 0) - U_1\left(\mu, \mu; p = \frac{1}{2}\right) &= I_1(\mu, \mu; p = 0) - I_1\left(\mu, \mu; p = \frac{1}{2}\right) \\ &= \frac{\lambda(2\mu - \lambda)}{(\mu + \lambda)(2\mu + \lambda)} > 0 \qquad \big(\text{using } (4.7)\big). \end{aligned}$$

□

Moving beyond FSF and SSF, we continue our equilibrium analysis (for a finite r) by using the first order conditions to show that whenever an r-routing policy admits a symmetric equilibrium, it is unique. Furthermore, we provide an expression for the corresponding symmetric equilibrium service rate in terms of r, which brings out a useful monotonicity property.

Theorem 4.5. *Consider an M/M/2 queue with strategic servers. Then, any r-routing policy that admits a symmetric equilibrium, admits a unique symmetric equilibrium, given by $\mu^\star = \varphi^{-1}(r)$, where $\varphi\colon (\frac{\lambda}{2}, \infty) \to \mathbb{R}$ is the function defined by*

$$\varphi(\mu) = \frac{4(\lambda + \mu)}{\lambda(\lambda - 2\mu)} \left(\mu(\lambda + 2\mu)c'(\mu) - \lambda\right). \tag{4.10}$$

Furthermore, among all such policies, $\mu^\star$ is decreasing in r, and therefore, $\mathbb{E}[T]$, the mean response time (a.k.a. sojourn time) at symmetric equilibrium is increasing in r.

Proof. The proof of this theorem consists of two parts. First, we show that under any r-routing policy, any symmetric equilibrium $\mu^\star \in (\frac{\lambda}{2}, \infty)$ must satisfy the equation $\varphi(\mu^\star) = r$. This is a direct consequence of the necessary first order condition for the utility function of server 1 to attain an interior maximum at $\mu^\star$. The second part of the proof involves using the condition $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$ to show that φ is a *strictly decreasing bijection* onto $\mathbb{R}$, which would lead to the following implications:

- φ is invertible; therefore, if an r-routing policy admits a symmetric equilibrium, it is unique, and is given by $\mu^\star = \varphi^{-1}(r)$.
- $\varphi^{-1}(r)$ is strictly decreasing in r; therefore, so is the unique symmetric equilibrium (if it exists). Since the mean response time $\mathbb{E}[T]$ is inversely related to the service rate, this establishes that $\mathbb{E}[T]$ at symmetric equilibrium (across r-routing policies that admit one) is increasing in r.

We begin with the first order condition for an interior maximum. The utility function of server 1 under an r-routing policy, from (4.2), is given by

$$U_1^r(\mu_1, \mu_2) = I_1^r(\mu_1, \mu_2) - c(\mu_1)$$

For $\mu^\star \in (\lambda/2, \infty)$ to be a symmetric equilibrium, the function $U_1^r(\mu_1, \mu^\star)$ must attain a global maximum at $\mu_1 = \mu^\star$. The corresponding first order condition is then given by:

$$\left.\frac{\partial I_1^r}{\partial \mu_1}(\mu_1, \mu^\star)\right|_{\mu_1=\mu^\star} = c'(\mu^\star), \tag{4.11}$$

where I_1^r is given by Proposition 4.3. The partial derivative of the idle time can be computed and the left hand side of the above equation evaluates to

$$\left.\frac{\partial I_1^r}{\partial \mu_1}(\mu_1, \mu^\star)\right|_{\mu_1=\mu^\star} = \frac{\lambda(4\lambda + 4\mu^\star + \lambda r - 2\mu^\star r)}{4\mu^\star(\lambda+\mu^\star)(\lambda+2\mu^\star)}. \tag{4.12}$$

Substituting in (4.11) and rearranging the terms, we obtain:

$$\frac{4(\lambda+\mu^\star)}{\lambda(\lambda-2\mu^\star)}\left(\mu^\star(\lambda+2\mu^\star)c'(\mu^\star) - \lambda\right) = r.$$

The left hand side is equal to $\varphi(\mu^\star)$, thus yielding the necessary condition $\varphi(\mu^\star) = r$.

Next, we proceed to show that if $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$, then φ is a strictly decreasing bijection onto $\mathbb{R}$. Note that the function

$$\varphi(\mu) = \frac{4(\lambda+\mu)}{\lambda(\lambda-2\mu)}\left(\mu(\lambda+2\mu)c'(\mu) - \lambda\right)$$

is clearly a continuous function in $(\frac{\lambda}{2}, \infty)$. In addition, it is a surjection onto $\mathbb{R}$, as evidenced by the facts that $\varphi(\mu) \to -\infty$ as $\mu \to \infty$ and $\varphi(\mu) \to \infty$ as $\mu \to \frac{\lambda}{2}+$ (using $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$).

To complete the proof, it is sufficient to show that $\varphi'(\mu) < 0$ for all $\mu \in (\frac{\lambda}{2}, \infty)$. First, observe that

$$\varphi'(\mu) = \frac{4\psi(\mu)}{\lambda(\lambda-2\mu)^2},$$

where

$$\psi(\mu) = \mu(\lambda+\mu)(\lambda^2 - 4\mu^2)c''(\mu) + (\lambda^3 + 6\lambda^2\mu - 8\mu^3)c'(\mu) - 3\lambda^2.$$

Since $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$, as $\mu \to \frac{\lambda}{2}+$, $\psi(\mu) < 0$. Moreover, since $c'''(\mu) > 0$, for all $\mu > \frac{\lambda}{2}$, we have

$$\psi'(\mu) = -4\mu(\lambda+\mu)\left(\mu^2 - \left(\frac{\lambda}{2}\right)^2\right)c'''(\mu) - 4\left(\mu - \frac{\lambda}{2}\right)(\lambda^2 + 6\lambda\mu + 6\mu^2)c''(\mu) - 24\left(\mu^2 - \left(\frac{\lambda}{2}\right)^2\right)c'(\mu) < 0.$$

It follows that $\psi(\mu) < 0$ for all $\mu > \frac{\lambda}{2}$. Since $\varphi'(\mu)$ has the same sign as $\psi(\mu)$, we conclude that $\varphi'(\mu) < 0$, as desired. $\square$

In light of the inverse relationship between r and $\mu^\star$ that is established by Theorem 4.5, the system manager would ideally choose the smallest r such that the corresponding r-routing policy admits a symmetric equilibrium, which is in line with the intuition that a bias towards SSF (the limiting r-routing policy as $r \to -\infty$) incentivizes servers to work harder. However, there is a hard limit on how small an r can be chosen (concurrently, how large an equilibrium service rate $\mu^\star$ can be achieved) so that there exists a symmetric equilibrium, as evidenced by our next theorem.

Theorem 4.6. *Consider an M/M/2 queue with strategic servers. Then, there exists $\overline{\mu}, \underline{r} \in \mathbb{R}$, with $\underline{r} = \varphi(\overline{\mu})$, such that no service rate $\mu > \overline{\mu}$ can be a symmetric equilibrium under any r-routing policy, and no r-routing policy with $r < \underline{r}$ admits a symmetric equilibrium.*

Proof. From Theorem 4.5, we know that if a symmetric equilibrium exists, then it is unique, and is given by $\mu^\star = \varphi^{-1}(r)$, where φ establishes a one-to-one correspondence between r and $\mu^\star$ ($\mu^\star$ is strictly decreasing in r and vice versa). Therefore, it is enough to show that there exists a finite upper bound $\overline{\mu} > \frac{\lambda}{2}$ such that no service rate $\mu > \overline{\mu}$ can be a symmetric equilibrium under *any* r-routing policy. It would then automatically follow that for $\underline{r} = \varphi(\overline{\mu})$, no r-routing policy with $r \leq \underline{r}$ admits a symmetric equilibrium. We prove this by exhibiting a $\overline{\mu}$ and showing that if $\mu \geq \overline{\mu}$, then the utility function of server 1, $U_1^r(\mu_1, \mu)$, cannot attain a global maximum at $\mu_1 = \mu$ for any $r \in \mathbb{R}$.

We begin by establishing a lower bound for the maximum utility $U_1^r(\mu_1, \mu)$ that server 1 can obtain under any r-routing policy:

$$\max_{\mu_1 > \frac{\lambda}{2}} U_1^r(\mu_1, \mu) \geq U_1^r\left(\frac{\lambda}{2}, \mu\right) = I_1^r\left(\frac{\lambda}{2}, \mu\right) - c\left(\frac{\lambda}{2}\right) \geq -c\left(\frac{\lambda}{2}\right) = U_1^r\left(\frac{\lambda}{2}, \frac{\lambda}{2}\right). \tag{4.13}$$

By definition, if $\mu^\star$ is a symmetric equilibrium under any r-routing policy, then the utility function of server 1, $U_1^r(\mu_1, \mu^\star)$, is maximized at $\mu_1 = \mu^\star$, and hence, using (4.13), we have

$$U_1^r(\mu^\star, \mu^\star) \geq U_1^r(\frac{\lambda}{2}, \frac{\lambda}{2}). \tag{4.14}$$

Next, we establish some properties on $U_1^r(\mu, \mu)$ that help us translate this necessary condition for a symmetric equilibrium into an upper bound on any symmetric equilibrium service rate. We have,

$$U_1^r(\mu, \mu) = 1 - \frac{\lambda}{2\mu} - c(\mu),$$

which has the following properties:

- Since $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$, $U_1^r(\mu, \mu)$, as a function of μ, is strictly increasing at $\mu = \frac{\lambda}{2}$.
- $U_1^r(\mu, \mu)$ is a concave function of μ.

This means that $U_1^r(\mu, \mu)$ is strictly increasing at $\mu = \frac{\lambda}{2}$, attains a maximum at the unique $\mu_\dagger > \frac{\lambda}{2}$ that solves the first order condition $\mu_\dagger^2 c'(\mu_\dagger) = \frac{\lambda}{2}$, and then decreases forever. This shape of the curve $U_1^r(\mu, \mu)$ implies that there must exist a unique $\overline{\mu} > \mu_\dagger$, such that $U_1^r(\overline{\mu}, \overline{\mu}) = U_1^r(\frac{\lambda}{2}, \frac{\lambda}{2})$.

Since $U_1^r(\mu, \mu)$ is a strictly decreasing function for $\mu > \mu_\dagger$, it follows that if $\mu^\star > \overline{\mu}$, then, $U_1^r(\mu^\star, \mu^\star) < U_1^r(\overline{\mu}, \overline{\mu}) = U_1^r(\frac{\lambda}{2}, \frac{\lambda}{2})$, contradicting the necessary condition (4.14). This establishes the required upper bound $\overline{\mu}$ on any symmetric equilibrium service rate, completing the proof. □

This proof is constructive, as we exhibit an $\underline{r}$, however, it is not clear whether this is tight, that is, whether there exists a symmetric equilibrium for all r-routing policies with $r \geq \underline{r}$. We provide a partial answer to this question of what r-routing policies do admit symmetric equilibria in the following theorem.

Theorem 4.7. *Consider an M/M/2 queue with strategic servers. Then, there exists a unique symmetric equilibrium under any r-routing policy with $r \in \{-2, -1, 0, 1\}$.*

Proof. A useful tool for proving this theorem is Theorem 3 from [32], whose statement we have adapted to our model:

Theorem 4.8. *A symmetric game with a nonempty, convex, and compact strategy space, and utility functions that are continuous and quasiconcave has a symmetric (pure-strategy) equilibrium.*

We begin by verifying that our 2-server game meets the qualifying conditions of Theorem 4.8:

- *Symmetry:* First, all servers have the same strategy space of service rates, namely, $(\frac{\lambda}{2}, \infty)$. Moreover, since an r-routing policy is symmetric and all servers have the same cost function, their utility functions are symmetric as well. Hence, our 2-server game is indeed symmetric.
- *Strategy space:* The strategy space $(\frac{\lambda}{2}, \infty)$ is nonempty and convex, but not compact, as required by Theorem 4.8. Hence, for the time being, we modify the strategy space to be $[\frac{\lambda}{2}, \overline{\mu} + 1]$ so that it is compact, where $\overline{\mu}$ is the upper bound on any symmetric equilibrium, established in Theorem 4.6, and deal with the implications of this modification later.
- *Utility function:* $U_1^r(\mu_1, \mu_2)$ is clearly continuous. From Mathematica, it can be verified that the idle time function $I_1^r(\mu_1, \mu_2)$ is concave in μ_1 for $r \in \{-2, -1, 0, 1\}$, and since the cost function is convex, this means the utility functions are also concave. (Unfortunately, we could not get Mathematica to verify concavity for non-integral values of r, though we strongly suspect that it is so for the entire interval $[-2, 1]$.)

Therefore, we can apply Theorem 4.8 to infer that an r-routing policy with $r \in \{-2, -1, 0, 1\}$ admits a symmetric equilibrium in $[\frac{\lambda}{2}, \overline{\mu} + 1]$. We now show that the boundaries cannot be symmetric equilibria. We already know from Theorem 4.6 that $\overline{\mu} + 1$ cannot be a symmetric equilibrium. (We could have chosen to close the interval at any $\mu > \overline{\mu}$. The choice $\overline{\mu} + 1$ was arbitrary.) To see that $\frac{\lambda}{2}$ cannot be a symmetric equilibrium, observe that $c'(\frac{\lambda}{2}) < \frac{1}{\lambda}$ implies that $U_1^r(\mu_1, \frac{\lambda}{2})$ is increasing at $\mu_1 = \frac{\lambda}{2}$ (using the derivative of the idle time computed in (4.12)), and hence server 1 would have an incentive to deviate. Therefore, any symmetric equilibrium must

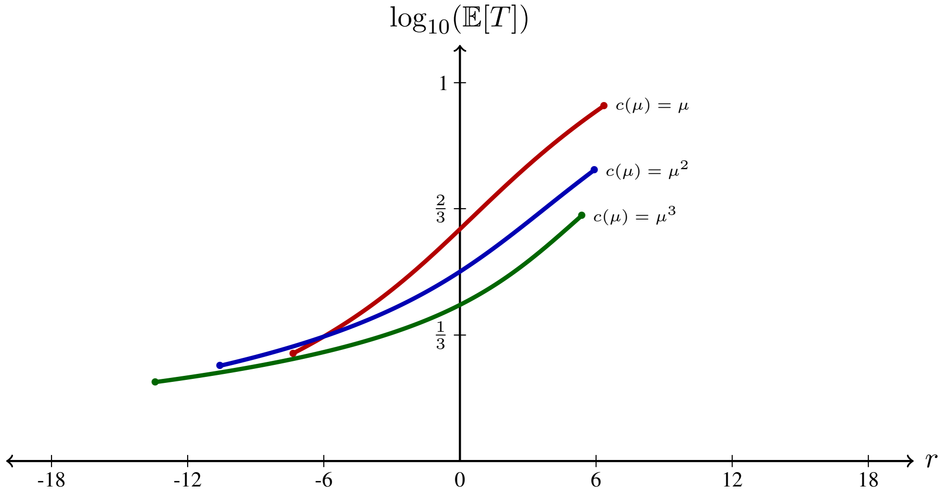


Figure 4.3: Equilibrium mean response time (a.k.a. sojourn time) as a function of the policy parameter, r, when the arrival rate is $\lambda = \frac{1}{4}$, for three different effort cost functions: linear, quadratic, and cubic.

be an interior point, and from Theorem 4.5, such an equilibrium must be unique. This completes the proof. $\square$

Notice that we show equilibrium existence for four integral values of r. It is challenging to show that all r-routing policies in the interval $[-2, 1]$ admit a symmetric equilibrium. Theorem 4.7 provides an upper bound on the $\underline{r}$ of Theorem 4.6, that is, $\underline{r} \leq -2$. Therefore, if the specific cost function c is unknown, then the system manager can guarantee better performance than Random ($r = 0$), by setting $r = -2$. If the specific cost function is known, the system manager may be able to employ a lower r to obtain even better performance. For example, consider a 2-server system with $\lambda = 1/4$ and one of three different effort cost functions: $c(\mu) = \mu$, $c(\mu) = \mu^2$, and $c(\mu) = \mu^3$. Figure 4.3 shows the corresponding equilibrium mean response times (in red, blue, and green, respectively). It is worth noting that the more convex the effort cost function, larger the range of r (and smaller the minimum value of r) for which a symmetric equilibrium exists.

4.5 Conclusion

The rate at which each server works in a service system has important consequences for service system design. However, traditional models of service systems do not capture the fact that human servers respond to incentives created by scheduling and staffing policies, because traditional models assume each server works at a given fixed service rate. In this chapter, we initiate the study of a class of strategic servers that seek to optimize a utility function which values idle time and includes an effort cost.

Our focus is on the analysis of routing policies for an M/M/N queue with strategic servers, and our results highlight that strategic servers have a dramatic impact on the optimal routing policies. In particular, policies that are optimal in the classical, nonstrategic setting can perform quite poorly when servers act strategically. We find that by considering a new family of policies, namely rate-based policies, we can achieve much better system performance.

Finally, it is important to note that we have focused on symmetric equilibrium service rates. We have not proven that asymmetric equilibria do not exist. Thus, it is natural to wonder if there are routing policies that result in an asymmetric equilibrium. Potentially, there could be one group of servers that have low effort costs but negligible idle time and another group of servers that enjoy plentiful idle time but have high effort costs. The question of asymmetric equilibria becomes even more interesting when the servers have different utility functions. For example, more experienced servers likely have lower effort costs than new hires. Also, different servers can value their idle time differently. How do we design routing and staffing policies that are respectful of such considerations?

Chapter 5

Routing with Heterogeneous Job Values

5.1 Introduction

Server farms are commonplace today in web servers, data centers, and in compute clusters. Such architectures are inexpensive (compared to a single fast server) and afford flexibility and scalability in computational power. However, their efficiency relies on having a good algorithm for routing incoming jobs to servers.

A typical server farm consists of a front-end router, which receives all the incoming jobs and dispatches each job to one of a collection of servers which do the actual processing, as depicted in Figure 5.1. The servers themselves are "off-the-shelf" commodity servers which typically schedule all jobs in their queue via Processor-Sharing (PS); this cannot easily be changed to some other scheduling policy. All the decision-making is done at the central dispatcher. The dispatcher (also called a load balancer) employs a *dispatching* policy (often called a *load balancing* policy or a *task assignment policy*), which specifies to which server an incoming request should be routed. Each incoming job is *immediately* dispatched by the dispatcher to one of the servers (this immediate dispatching is important because it allows the server to quickly set up a connection with the client, before the connection request is dropped). Typical dispatchers used include Cisco's Local Director [1], IBM's Network Dispatcher [107], F5's Big IP [48], Microsoft Sharepoint [2], etc. Since scheduling at the servers is not under our control, it is extremely important that the right dispatching policy is used.

Prior work has studied dispatching policies with the goal of minimizing mean *response time*, $\mathbb{E}[T]$; a job's response time is the time from when the job arrives until it completes. Several papers have specifically studied the case where the servers schedule their jobs via PS (see [9, 18, 20, 49, 63, 83, 100, 104]). Here, it has been show that the Join-the-Shortest-Queue (JSQ) policy performs very well, for general job size distributions. Even picking the shortest of a small subset of the queues, or simply trying to pick an idle queue if it exists, works very well. Interestingly, such simple policies like JSQ are superior even to policies like Least-Work-Left, which route a job to the server with the least remaining total work (sum of remaining sizes of all jobs at the queue), rather than simply looking at the number of jobs [70]. In addition, there have been many more papers studying dispatching policies where the servers schedule jobs in First-Come-First-Served (FCFS) order (see e.g., [6, 16, 28, 34, 45, 46, 49, 69, 71, 89, 101, 129, 130]). Here

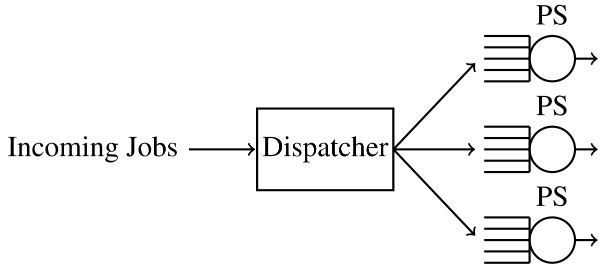


Figure 5.1: Dispatching in server farms with Processor-Sharing (PS) servers.

high job size variability can play a large role, and policies like Size-Interval-Task-Assignment (SITA) [71], which segregates jobs based on job size, or Least-Work-Left [73], which routes job to the queue with the least total remaining work (rather than the smallest number of jobs), are far superior to JSQ.

However, all of this prior work has assumed that jobs have equal importance (value), in that they are equally sensitive to delay. This is not at all the case. Some jobs might be background jobs, which are largely insensitive to delay, while others have a live user waiting for the result of the computation. There may be other jobs that are even more important in that *many* users depend on their results, or other jobs depend on their completion. We assume that every job has a value, V, independent of its size (service requirement). Given jobs with heterogeneous values, the right metric to minimize is not the mean response time, $\mathbb{E}[T]$, but rather the mean *value-weighed response time*, $\mathbb{E}[VT]$, where jobs of higher value (importance) are given lower response times.

The problem of minimizing $\mathbb{E}[VT]$, where V and T are independent, is also not new, although it has almost exclusively been considered in the case of server scheduling, *not in the case of dispatching* (see Prior Work section). Specifically, there is a large body of work in the operations research community where jobs have a *holding cost*, c, independent of the job size, and the goal is to minimizing $\mathbb{E}[c \cdot T]$ over all jobs. Here it is well-known that the $c\mu$ rule is optimal [50]. In the $c\mu$ rule, c refers to a job's holding cost and μ is the reciprocal of a job's size. The $c\mu$ rule always runs the job with the highest product c times μ; thus, jobs with high holding cost and/or small size are favored. However, there has been no $c\mu$-like dispatching policy proposed for server farms.

In this paper, we assume a server farm with a dispatcher and PS servers. Jobs arrive according to a Poisson process and are immediately dispatched to a server. The value, V, of an arrival is known, but its size, S, is not known. Furthermore, we assume that value and size are independent, so that knowing the job's value does not give us information about the job's size. We assume that we know the distribution job values. Furthermore, job sizes are exponentially-distributed with unit mean. By requiring that jobs are exponentially distributed, we are consistent with the assumption that there is no way to estimate a job's size; otherwise, we could use "age" information to update predictions on the remaining size of each job, and some of the policies of interest would become much more complex.[1] Nothing else is known about future arrivals.

[1]We do in fact carry out a set of simulations assuming an alternative job size distribution, with policies that ignore "age" information. The qualitative results remain the same as those under exponentially distributed job sizes;

In making dispatching decisions, we assume that we know the *queue length* at each server (this is the number of jobs at the PS server) as well as the values of the jobs at each server. In this context, we ask:

"What is a good dispatching policy to minimize $\mathbb{E}[VT]$?"

Even in this simple setting, it is not at all obvious what makes a good dispatching policy. We consider several policies (see Section 5.4 for more detail):

- The **Random (RND)** dispatching policy ignores job values and queue lengths. Arrivals are dispatched randomly.
- The **Join-Shortest-Queue (JSQ)** dispatching policy ignores values and routes each job to the server with the fewest number of jobs. This policy is known to be optimal in the case where all values are equal [63].
- The **Value-Interval-Task-Assignment (VITA)** dispatching policy is reminiscent of the SITA policy, where this time jobs are segregated by *value*, with low-value jobs going to one server, medium value jobs going to the next server, higher-value jobs going to the next server, and so on. The goal of this policy is to isolate high value jobs from other jobs, so that the high value jobs can experience low delay. The distribution of V and system load ρ are used to determine the optimal threshold(s) for minimizing $\mathbb{E}[VT]$.
- The **C-MU** dispatching policy is motivated by the $c\mu$ rule for scheduling in servers. Each arrival is dispatched so as to maximize the average instantaneous value of the jobs completing, assuming no future arrivals, where the average is taken over the servers. This policy makes use of the value of the arrival and the values of all the jobs at each server.
- The **Length-And-Value-Aware (LAVA)** dispatching policy is very similar to the C-MU policy. Both policies incorporate queue length and job values in their decision. However, whereas C-MU places jobs so as to maximize the expected instantaneous value of jobs completed, LAVA places jobs so as to explicitly minimize $\mathbb{E}[VT]$ over jobs. Both policies make their decisions solely based on jobs already in the system.

This paper is the first to introduce the VITA, C-MU, and LAVA policies.

Via a combination of asymptotic analysis, exact analysis, and simulation we show the following in Sections 5.5 and 5.6. We find that generally RND is worse than VITA, which is worse than JSQ, which is worse than LAVA. In fact, under an asymptotic regime we prove that as system load $\rho \to 1$, the ratio $\mathbb{E}[VT]^{\text{RND}} : \mathbb{E}[VT]^{\text{VITA}} : \mathbb{E}[VT]^{\text{JSQ}} : \mathbb{E}[VT]^{\text{LAVA}}$ approaches $4 : 2 : 2 : 1$. The C-MU policy, on the other hand, avoids neat classification. There are value distributions and loads for which C-MU is the best policy of those we study, and others for which C-MU is the worst. In fact, C-MU can become unstable even when system load $\rho < 1$. Finally, while VITA is generally not a great policy, we find that there are certain regimes under which VITA approaches optimality under light load ($\rho < 1/2$), performing far better than the other policies we study.

But is it possible to do even better than the above dispatching policies? We find that under a particularly skewed value distribution, there is a policy, "Gated VITA," which can outperform all of the aforementioned policies by an *arbitrary* factor. The idea behind this policy is to split high and low value jobs, while using a "gate" to place a limit on the number of low-value jobs that can

see Section 5.5.

interfere with high-value jobs (see Section 5.7 for details). If one is willing to forego simplicity in the dispatching policies, one can further use first policy iteration to significantly improve upon simple policies (see Section 5.8 for details).

5.2 Prior work on value-driven dispatching

The problem of finding dispatching policies with the aim of minimizing value-weighted response time has received very little attention in the literature. Below we discuss the few papers in this setting, which are (only tangentially) related to our own.

One paper concerned with the minimization of an $\mathbb{E}[VT]$-like metric is [9], where a constant value parameter is associated with each *server*. In this setting, job values are not treated as exogenous random variables determined at the time of arrival; instead, the value of a job is set to the value associated with the server serving the job, and hence, a job's value is determined by where the dispatcher sends it.

Another research stream that considers heterogeneity in the delay sensitivity of jobs is the dispatching literature concerned with minimizing *slowdown*, $\mathbb{E}[\frac{1}{X} \cdot T]$, where X is a job's service requirement (size) [72, 84, 131]. This body of literature differs from our work in two key ways. First, unlike our work, the "value" of each job is deterministically related to (in particular, is the multiplicative inverse of), rather than independent of, the job's size. Second, the slowdown metric necessitates the examination of dispatching policies that can observe job sizes.

Finally, settings similar to ours are considered in [81, 82]. Unlike this chapter, however, these papers do not provide a comprehensive comparison of dispatching policies: [81] is concerned with deriving one specific policy (the lookahead policy), while [82] only considers the simple random and round robin policies, together with FPI improvements on these policies, which make use of job sizes.

5.3 Model for PS server system

The basic system, illustrated in Fig. 5.1, is as follows:

- We have m servers with Processor-Sharing (PS) scheduling discipline and service rate μ_i. Throughout the simulation and analytic portion of this chapter, we give particular attention to the case where $m = 2$ and $\mu_1 = \mu_2$.
- Jobs arrive according to the Poisson process with rate λ and are immediately dispatched to one of the m servers.
- Job j is defined by a pair $(X^{(j)}, V^{(j)})$, where $X^{(j)}$ denotes the size of the job and $V^{(j)}$ is its value.
- Job sizes obey exponential distribution with unit mean, $\mathbb{E}[X] = 1$, preventing us from using the "age" of a job to learn about its remaining size.
- The system load is given by $\rho \equiv \lambda/(\sum_{i=1}^{m} \mu_i)$. When $m = 2$ and $\mu_1 = \mu_2$, we have $\rho = \lambda/(2\mu)$.

- We can observe the number of jobs in each server (queue length), but not their service times.
- The values $\{V^{(j)}\}$ are drawn from a known distribution with finite mean and nonzero variance. A job's value becomes known upon arrival. We can also observe the values of jobs at each server.
- Jobs are i.i.d., i.e., $(X^{(j)}, V^{(j)}) \sim (X, V)$, where $X^{(j)}$ and $V^{(j)}$ are independent. In particular, it is not possible to deduce anything about a job's size based on its value.
- The objective is to minimize the mean (or time-average) value-weighted response time, given by $\mathbb{E}[VT] \equiv \lim_{n\to\infty} \left(\frac{1}{n} \sum_{j=1}^{n} V^{(j)} T^{(j)} \right)$, where $T^{(j)}$ is the response time experienced by job j.

Notation: Throughout, it will be convenient to use n_i to denote the number of jobs that an arrival sees at server i; $v_{i,j}$ to be the value of the jth job at server i; $v_i^{\text{sum}} \equiv \sum_{j=1}^{n_i} v_{i,j}$ to denote the total values of jobs that an arrival sees at server i; and $\bar{v}_i \equiv v_i^{\text{sum}}/n_i$ to denote the average value of jobs at server i.

5.4 Description of simple dispatching policies

In describing our dispatching policies, it will be convenient to use the following terms.

Definition 5.1. *The* **state** *of a queue consists of its queue length and the specific values of jobs at the queue.*

Definition 5.2. *A dispatching policy is called* **static** *if its decision is independent of the queue states and independent of all past placement of jobs.*[2]

Definition 5.3. *A dispatching policy is called* **value-aware** *if the policy requires knowing the value of a new arrival.*

5.4.1 Random dispatching (RND)

The RND policy dispatches each incoming job to server i with probability $1/m$, where m is the number of servers.

5.4.2 Join-the-Shortest-Queue dispatching (JSQ)

The JSQ policy dispatches each incoming job to the server with the shortest queue length. If multiple queues have the same shortest length, JSQ picks among them at random. Like RND, JSQ does not make use of the value a job. JSQ is typically superior to RND in that it balances the *instantaneous* queue lengths. It is known to be either optimal or very good for minimizing $\mathbb{E}[T]$ in a variety of settings [63]. Observe that $\mathbb{E}[T] = \mathbb{E}[VT]$ in the case where all values are equal.

[2]Note that a policy such as Round-Robin is not considered **static** in this chapter. The placement of a job in the Round-Robin policy is determined by the placement of the previous job: if the Round-Robin policy sends an arrival to server j, then it sends the next arrival to server $j + 1 \mod m$. In particular, a static policy ensures that the arrival process to each server is a Poisson process.

5.4.3 Value-Interval-Task-Assignment (VITA)

The VITA policy is our first value-aware policy. The idea is that each server is assigned a "value interval" (e.g., "small," "medium," or "large" values), and an incoming job is dispatched to that server that is appropriate for its value. Specifically, assume that the value distribution has a continuous support without atomic probabilities, ranging from 0 to ∞. In this case, we can imagine specifying value "thresholds," $\xi_0, \xi_1, \ldots, \xi_m$, where $0 = \xi_0 < \xi_1 < \ldots < \xi_{m-1} < \xi_m = \infty$. Then VITA assigns jobs with value $V \in (\xi_{i-1}, \xi_i)$ to server i.[3] In the case where there is a nonzero probability mass associated with a particular value, v, it may be the case that jobs with value v are routed to a subset $n > 1$ of the m servers. In this case, we also must specify additional "thresholds" in the form of probabilities, $p_1, p_2, \cdots, p_n$, where $\sum_i p_i = 1$, and jobs of value v are routed to the ith server of the n with probability p_i.

Thus we can see that VITA may depend on various threshold parameters. *Throughout we define VITA to use those thresholds which minimize* $\mathbb{E}[VT]$. VITA is a static policy, and thus practical for distributed operation with any number of parallel dispatchers.

The intuition behind VITA is that it allows high-value jobs to have isolation from low-value jobs. Given that our goal is to provide high-value jobs with low response times, it makes sense to have some servers which serve exclusively higher-value jobs, so that these jobs are not slowed down by other jobs. Of course the optimal choice of thresholds depends on the value distribution and the load.

It turns out that VITA is the optimal static policy for minimizing $\mathbb{E}[VT]$. For clarity, we will prove that VITA is the optimal static value-aware policy in the case of $m = 2$ servers with identical service rates; however this result easily extends to $m > 2$ servers.

Proposition 5.4. *VITA is the optimal (i.e., $\mathbb{E}[VT]$-minimizing) static policy for any two-server system with identical service rates. Furthermore, VITA unbalances the load, whereas all load balancing static policies achieve the same performance as RND.*

Proof. Deferred to Appendix. □

5.4.4 C-MU

The classic $c\mu$ rule for scheduling in servers prioritizes jobs with the highest product of value (c) and inverse expected remaining service requirement (μ). This policy for server scheduling is known to be optimal in many scheduling contexts [50].

Our C-MU dispatching policy is inspired by the $c\mu$ scheduling rule, in that it aims to maximize the value-weighted departure rate.

As always, we use n_i to denote the number of jobs that an arrival sees at server i; $v_i^{\mathrm{sum}} = \sum_{j=1}^{n_i} v_{i,j}$ for the sum of job values at server i; and $\bar{v}_i = v_i^{\mathrm{sum}}/n_i$ to denote the average value of jobs at server i. Since PS scheduling provides all jobs equal service rate, $\bar{v}_i\mu_i$ denotes the current "rate of value departing" from server i. The *total rate of value departing* is of course $\sum_i \bar{v}_i\mu_i$.

The C-MU dispatching rule greedily routes each incoming job so as to maximize the instantaneous total rate of value departing. This policy is myopic in that it makes its routing decision

[3]Since it is preferable to send high-value jobs to wherever they can be served fastest, we assume without loss of generality that $\mu_1 \leq \cdots \leq \mu_m$.

solely based on jobs already in the system, not taking into account future arrivals or departures. Specifically, an incoming job of value v is routed to that server, i, whose rate of value departing will increase the most (or decrease least) by having the job. That is, i satisfying

$$\operatorname*{argmax}_i \frac{\mu_i}{n_i+1}\left(v_i^{\text{sum}}+v\right)-\frac{\mu_i}{n_i}\cdot v_i^{\text{sum}}=\operatorname*{argmax}_i \frac{v-\bar{v}_i}{n_i+1}\mu_i=\operatorname*{argmin}_i \frac{\bar{v}_i-v}{n_i+1}\mu_i.$$

Let $\alpha_{\text{C-MU}}(v)$ denote the server to which a job of value v is routed under C-MU dispatching. Then

$$\boxed{\alpha_{\text{C-MU}}(v)=\operatorname*{argmin}_i \frac{\bar{v}_i-v}{n_i+1}\mu_i.} \tag{5.1}$$

Note that if some value jobs are common, then $\bar{v}_i=v$ may occur frequently and the numerator in (5.1) becomes zero. In such cases C-MU is “clueless”. For example, it cannot decide between a server with a billion jobs with value v and a server with a single job with value v. We use *random splitting* to resolve ties, but note that in some specific (asymptotic) cases significant improvements can be achieved if ties are resolved in some other manner, e.g., via JSQ.

As an example of C-MU, suppose that there are 2 jobs of value 10 at server 1 and 4 jobs of value 1 at server 2. Suppose also that $\mu_1=\mu_2=1$. The current value-weighted departure rate at server 1 is 10, and that at server 2 is 1. Consider an incoming arrival of value v. If the arrival is routed to server 1, then it will increase the mean value-weighted departure rate at server 1 to $(20+v)/3$. The total rate of value departing from the system will increase from 11 to $(23+v)/3$. If, on the other hand, the arrival is routed to server 2, then it will increase the mean value-weighted departure rate at server 2 to $(4+v)/5$. The total rate of value departing from the system will then increase from 11 to $(54+v)/5$. Thus the new arrival should be routed to server 1 if $(23+v)/3>(54+v)/5$; to server 2 if $(23+v)/3<(54+v)/5$; and to each server with probability $1/2$ if $(23+v)/3=(54+v)/5$. These cases correspond to v greater than, less than, and equal to $47/2$, respectively. Note that the value of the incoming job has to be very high (i.e., $47/2$ or greater) before C-MU is willing to send it to server 1.

5.4.5 Length-and-Value-Aware (LAVA)

Like the C-MU policy, LAVA is state-aware, using both the queue lengths and the values of all jobs at each server. Also like C-MU, LAVA is myopic with respect to further arrivals. The key difference is that LAVA attempts to directly minimize the metric of interest, $\mathbb{E}[VT]$, whereas C-MU aims to maximize the total rate of value departing, which is related to minimizing $\mathbb{E}[VT]$ but not the same.

LAVA routes an arriving job so as to minimize $\mathbb{E}[VT]$, averaged over all jobs currently in the system, including the current arrival, assuming that no future jobs arrive. Similar myopic policies have been considered in [18, 63] where values are homogeneous, and the goal is minimizing $\mathbb{E}[VT]$.

Before formally defining LAVA, it is useful to compute $\mathbb{E}[VT]$ for an arbitrary system state under the assumption that there will be no further arrivals. The jobs at a server are equally likely to leave in any order. Hence the sum of the response times of jobs currently at server i is $\frac{n_i(n_i+1)}{2}\cdot\frac{1}{\mu_i}$, and the expected response time for each job at server i is $\frac{n_i+1}{2\mu_i}$. Thus the mean value-weighted response time of the jth job at server i is $\frac{n_i+1}{2\mu_i} \cdot v_{i,j}$. Recall that $\mathbb{E}[VT]$ is a per-job average. Thus, if n is the total number of jobs in the system, (i.e., $n = \sum_{i=1}^{m} n_i$), then

$$\mathbb{E}[VT] = \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{n_i} \left\{ \frac{n_i+1}{2\mu_i} \cdot v_{i,j} \right\} = \frac{1}{n} \sum_{i=1}^{m} \frac{n_i+1}{2\mu_i} \cdot v_i^{\text{sum}} = \sum_{i=1}^{m} S_i \qquad \text{where} \qquad S_i = \frac{n_i+1}{2\mu_i} \cdot v_i^{\text{sum}}.$$

We proceed to formally define LAVA. Given an arriving job with value v, we want to dispatch this job so as to minimize the resulting $\mathbb{E}[VT]$, as expressed above. That is, we want to send the arriving job to whichever server i will experience the least increase in S_i. If we send a job of value v to server i, then the expected response time of jobs at server i will increase from $(n_i+1)/(2\mu_i)$ to $(n_i+2)/(2\mu_i)$, resulting in a new value of S_i: $S_i^{new} = \frac{n_i+2}{2\mu_i}\left(v + v_i^{\text{sum}}\right)$. This means that

$$S_i^{new} - S_i = \frac{1}{2\mu_i}\left((n_i+2)v + v_i^{\text{sum}}\right).$$

Let $\alpha_{\text{LAVA}}(v)$ denote the server to which a job of value v is routed under LAVA dispatching.

$$\boxed{\alpha_{\text{LAVA}}(v) = \underset{i}{\operatorname{argmin}} \frac{1}{2\mu_i}\left((n_i+2)v + v_i^{\text{sum}}\right).} \tag{5.2}$$

With identical servers $\mu_i = \mu_j$, we can extract common constants and simplify the description of the policy to

$$\boxed{\alpha_{\text{LAVA}}(v) = \underset{i}{\operatorname{argmin}}\, n_i v + v_i^{\text{sum}}.} \tag{5.3}$$

As usual, ties are broken via random assignment.

Using this final formula (5.3), we return again to the example that we considered for C-MU, where there are 2 jobs of value 10 at server 1 and 4 jobs of value 1 at server 2. Under LAVA, a new arrival of value v prefers to go to server 1 if $2v + 2 \cdot 10 < 4v + 4 \cdot 1$, i.e., if $v > 8$. So a new arrival of value greater than 8 will join the value 10 jobs, while a new arrival of value less than 8 will join the value 1 jobs (a new arrival of value exactly 8 will pick randomly). Contrast this with C-MU, where the cutoff was $47/2$ rather than 8 for the same example.

5.5 Simulation results and intuitions

In this section, we report our findings from simulation in a two-server system (with identical service rates) for the value distributions given in Table 5.1. We also provide intuition for the results. Formal proofs will be given in the next section. In all cases, $\mathbb{E}[V] = 1$, and the continuous distributions (a)–(c) are presented in increasing order of variability, as are the discrete distributions (d)–(f). The variance is particularly high for distributions (e) and (f). Note that while the fraction of high-value jobs is the same in distributions (e) and (f), the *low-value* jobs contribute about 100 times as much to $\mathbb{E}[V]$ under (e) than under (f). Distribution (f) has a "sharply bimodal" form (to be defined in Section 6.3), whereby the high value jobs are extremely rare, yet comprise almost all of the value.

(a)	Uniform,	$V \sim \text{Uniform}(0, 2)$
(b)	Exponential,	$V \sim \text{Exp}(1)$
(c)	Bounded Pareto,	$V \sim \text{Pareto}(\min = 0.188, \max = 10\,000, \alpha = 1.2)$
(d)	Bimodal,	$V \sim \text{Bimodal}(99\%, 0.1; 1\%, 9.01)$
(e)	Bimodal,	$V \sim \text{Bimodal}(99.9\%, 0.1; 0.1\%, 900.1)$
(f)	Bimodal,	$V \sim \text{Bimodal}(99.9\%, 1/999; 0.1\%, 999)$

Table 5.1: Value distributions considered in the examples. $\mathbb{E}[V] = 1$ in all cases. $\text{Bimodal}(x\%, v_1; y\%, v_2)$ indicates that $x\%$ of jobs have value v_1 and $y\%$ of jobs have value v_2.

The simulation results are presented in Fig. 5.2, where for each of the distributions (a)–(f), we have plotted the performance of each policy, P, normalized by the performance of JSQ (i.e., $\mathbb{E}[VT]^{\text{P}}/\mathbb{E}[VT]^{\text{JSQ}}$), as a function of ρ. In all of these experiments, job sizes are exponentially distributed with mean 1. We ran additional simulations where (i) job sizes follow a Weibull distribution and (ii) job sizes follow a deterministic distribution. For these additional simulations, we continued to use the same implementation of LAVA and C-MU which do not attempt to learn a job's remaining service time. The results under these additional simulations were qualitatively similar to those under with exponentially distributed job sizes, suggesting that our findings are largely insensitive to the job-size distribution, as might be expected under Processor-Sharing service.

It is immediately apparent that RND is far worse than JSQ in all figures, and that $\mathbb{E}[VT]$ under RND converges to twice that under JSQ as load approaches 1. This factor 2 result will be proven in Proposition 5.5, however it is understandable since under high load the two servers under JSQ function similarly to a single server with twice the speed.

Given our result in Proposition 5.4 showing that VITA is the optimal static policy, it may seem surprising that VITA offers only a modest improvement over RND in Fig. 5.2 (a)–(d) and is so inferior to JSQ, despite JSQ not even using value information. To see what is going on, notice that under low load, there are only a few jobs in the system. Here VITA can mess up by putting these jobs onto the same server (if they have the same value), whereas JSQ never will. In fact dynamic policies like JSQ, CMU and LAVA are all "idle-eager," in that they will always route a job to an idle server if one exists. This gives them an advantage over VITA for low load. Under high load, VITA does not have much flexibility over protecting high-value jobs, since it is forced to balance load. Here, the performance under VITA is close to that under RND, even though VITA is the optimal static load balancing policy (cf. Proposition 5.4).

Fig. 5.2(f) is the exception. Here, under moderate load VITA outperforms all the other policies examined. The reason is that when nearly all of the value in the system is made up by a very small fraction of the jobs, VITA can maintain a much shorter queue for the most valuable jobs, without paying a big penalty for the resulting additional delays faced by the other jobs. This effect is particularly potent when $\rho < 1/2$, because the valuable jobs do not need to share the server "reserved" for them, since all low value jobs can be directed to the other server without violating stability constraints. Even under high load, we see that VITA performs similarly to JSQ, rather than RND. Although VITA has very limited flexibility for protecting high-value jobs

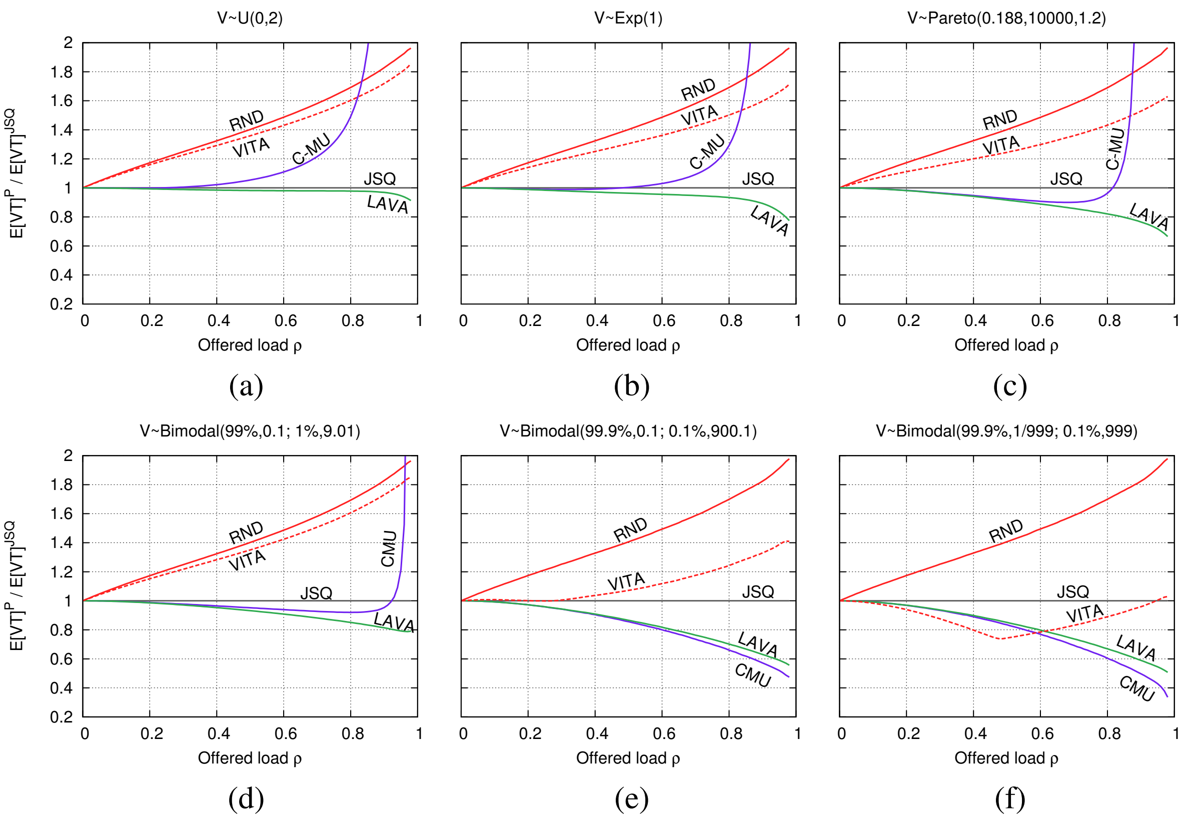


Figure 5.2: Performance relative to JSQ in a two-server system, with value distributions taken from Table 5.1 (a)–(f). Each point corresponds to the mean performance averaged over 100 million jobs.

under high load, the high value jobs under distribution (f) are *so extremely valuable* that even giving them a slight reduction in load (say 0.97 for the high value jobs and 0.99 for the low value jobs), will buy a factor of 2 improvement over RND. We formalize these notions in Theorems 5.8 and 5.9 of Section 5.6.

LAVA often outperforms all of the other policies. It is also the only policy that outperforms JSQ in all cases examined. Its consistent improvement over JSQ can be attributed to the fact that LAVA behaves like JSQ when all values are the same (or very close), but LAVA also uses value information to dispatch in favor of the most valuable jobs. Specifically, when an extremely valuable job enters the system, LAVA places the job somewhere and then essentially ceases sending jobs to that server (because it is crucial to the $\mathbb{E}[VT]$ metric that this job not be slowed down). Thus, this particularly valuable job ends up timesharing with an average of $n/2$ jobs over its lifetime under LAVA, where n denotes the number of jobs the high value job saw when it arrived. This "stopper" effect under LAVA is particularly important for sharply bimodal distributions like (f), where, as $\rho \to 1$, LAVA approximately obtains a 50% reduction in the $\mathbb{E}[VT]$ over JSQ. We formalize this result in Corollary 5.12.

Finally, turning to the C-MU policy, we notice that $\mathbb{E}[VT]$ diverges, suggesting that the system is unstable, for some values of $\rho < 1$ under distributions (a)–(d), while C-MU is the best performing policy under distributions (e) and (f). C-MU's good and bad performance can be attributed to the same protectiveness of high value jobs that we saw in LAVA. Specifically, the presence of a single high-value job at a server can prohibit the entry of any low-value jobs to that server. On the one hand, this is beneficial because the high-value jobs are protected; on the other hand, the server with the high-value job may be underutilized, giving rise to instability at the other server. While LAVA also exhibits such "stoppering" behavior, under C-MU this prohibition is more *severe* as it occurs *regardless of the length of the queue at the other server*. For value distributions (a)–(d), C-MU's extreme protectiveness of high-value jobs leads to instability at the other server. For distributions (e) and (f), the high-value jobs are much rarer; so much so that the "stoppering" periods are a lot less frequent and instability does not occur. Observation 5.7 and Theorem 5.11 of Section 5.6 shed more light on C-MU's behavior.

5.6 Analytic results

Motivated by the observations in the previous section, we proceed to present and prove analytic results.

5.6.1 RND and JSQ under high load

We start by noting that V and T are independent under both RND and JSQ, yielding $\mathbb{E}[VT] = \mathbb{E}[V]\mathbb{E}[T]$. Using a result from [130] that $\mathbb{E}[T]^{\text{RND}} \geq \mathbb{E}[T]^{\text{JSQ}}$ for all ρ, it follows that $\mathbb{E}[VT]^{\text{RND}} \geq \mathbb{E}[VT]^{\text{JSQ}}$ for all ρ. The proposition below provides an asymptotic comparison of RND and JSQ as $\rho \to 1$.

Proposition 5.5. *As* $\rho \to 1$, $\mathbb{E}[VT]^{\text{RND}}/\mathbb{E}[VT]^{\text{JSQ}} \to 2$.

Proof. We start by recalling a result by Foschini and Salz [51], stating that as $\rho \to 1$, the response time for a two-server system under JSQ with servers operating at rate μ approaches that of a single-server operating at rate 2μ, i.e.,

$$\lim_{\rho\to 1} \frac{\mathbb{E}[T]^{\text{JSQ}}}{1/(2\mu - \lambda)} = 1.$$

Next, we use the fact that V and T are independent under both RND and JSQ, yielding

$$\mathbb{E}[VT]^{\text{RND}} = \mathbb{E}[V]\mathbb{E}[T]^{\text{RND}} = \frac{\mathbb{E}[V]}{\mu - \lambda/2} \quad \text{and} \quad \mathbb{E}[VT]^{\text{JSQ}} = \mathbb{E}[V]\mathbb{E}[T]^{\text{JSQ}},$$

and complete the proof by applying the result from [51]:

$$\lim_{\rho\to 1} \frac{\mathbb{E}[VT]^{\text{RND}}}{\mathbb{E}[VT]^{\text{JSQ}}} = \lim_{\rho\to 1} \left(\left(\frac{\mathbb{E}[V]}{\mu - \lambda/2} \right) \bigg/ \left(\frac{\mathbb{E}[V]}{2\mu - \lambda} \right) \right) = \lim_{\rho\to 1} \left(\frac{2\mu - \lambda}{\mu - \lambda/2} \right) = 2. \qquad \square$$

5.6.2 Stability and instability

A dispatching policy is called **stable** if the resulting system has a finite response time (i.e., $\mathbb{E}[T] < \infty$), otherwise we say that it is unstable. A lack of stability can lead to infinitely poor performance with respect to $\mathbb{E}[VT]$. In fact, if the value distribution has positive lower and upper bounds, then $\mathbb{E}[VT]$ is finite if and only if $\mathbb{E}[T]$ is finite (i.e., if and only if the dispatching policy is stable). Hence, it is important to avoid implementing unstable policies. The following result shows that if the value distribution has positive lower and upper bounds, then all of the policies we have studied, with the exception of C-MU, are stable if and only if $\rho < 1$:

Theorem 5.6. *Let the value distribution have lower bound $a > 0$ and upper bound $b < \infty$. Then for a two-server system with identical service rates,* RND, JSQ, VITA, *and* LAVA *are stable if and only if $\rho < 1$.*

Proof. First we note that all dispatching policies are unstable when $\rho \geq 1$.

Clearly, $\mathbb{E}[T]^{\text{RND}} = 1/(\mu - \lambda/2)$, so this policy is stable whenever $\mu > \lambda/2$, or equivalently, whenever $\rho < 1$, as required. Next, recall that $\mathbb{E}[T]^{\text{JSQ}} \leq \mathbb{E}[T]^{\text{RND}} < \infty$ by [130], establishing that JSQ is also stable. Since VITA is the optimal static policy by Proposition 5.4, $\mathbb{E}[VT]^{\text{VITA}} \leq \mathbb{E}[VT]^{\text{RND}} = \mathbb{E}[V]\mathbb{E}[T]^{\text{RND}} < \infty$ for all $\rho < 1$. Moreover, since the value distribution has positive upper and lower bounds, $\mathbb{E}[VT]^{\text{VITA}} < \infty$ implies $\mathbb{E}[T]^{\text{VITA}} < \infty$, establishing that VITA is stable for all $\rho < 1$.

Next, we consider stability of LAVA: let N_{short} (N_{long}) be the time-varying random variable giving the number of jobs at whichever queue happens to be shorter (longer) at a given point in time. Clearly, $N_{\text{short}} \leq N_{\text{long}}$ at all times. Moreover, LAVA is stable if and only if $\mathbb{E}[N_{\text{long}}] < \infty$. Assume by way of contradiction that LAVA is actually unstable and $\mathbb{E}[N_{\text{long}}] = \infty$. In this case, $\mathbb{E}[N_{\text{short}}] < \infty$, because at least one of the servers must have an incoming time-average arrival rate of no more than $\lambda/2 < \mu$. Now observe that whenever $bN_{\text{short}} < aN_{\text{long}}$, LAVA sends all jobs to the shorter queue, setting the arrival rate to the longer queue to 0. However, since $\mathbb{E}[N_{\text{short}}] < \infty$, while $\mathbb{E}[N_{\text{long}}] = \infty$, this condition will hold almost always, contradicting the fact that $\mathbb{E}[N_{\text{long}}] = \infty$. □

Unlike the other policies, C-MU can give rise to unstable systems even when $\rho < 1$.

Observation 5.7. *There exists a value distribution with positive upper and lower bounds such that* C-MU *is unstable for some load $\rho < 1$.*

We now provide some theoretical justification for Observation 5.7. Our theoretical justifications are motivated by what we have witnessed in simulation. Consider a value distribution which is bimodal with values (for example) 1 and 10, where the value 1 jobs are extremely rare. Imagine a two-server system under C-MU dispatching with $\rho \in (3/4, 1)$. For a long time, all arrivals have value 10, and these are randomly split between the two queues by the C-MU policy. Eventually, a value 1 job arrives and is dispatched to the server with the longer queue. Without loss of generality, we assume that this is server 2. At this point the C-MU policy enters a particular *overload regime*, in which all subsequent arrivals are sent to server 2.[4] The reason for this is that the average value of jobs at server 1 is 10, while the average value of jobs at server 2 is

[4]We are ignoring the rare case where server 2 has many completions of value 10 jobs, making its queue length sufficiently less than that of server 1, causing value 1 jobs to be sent to server 1, while the queue at server 2 is shorter.

less than 10; hence, the average departing value across servers will be maximized by routing to server 2. During the overload regime, server 2's queue grows rapidly, as it receives all jobs.

Our simulations suggest that for sufficiently high ρ, we enter this overload regime frequently. Furthermore, although there are times when we are not in this overload regime, during which server 2's queue will shrink, we enter the overload regime so frequently that eventually server 2's queue grows beyond the point of no return, and server 2 heads off to instability.

To understand why this happens, consider what causes the overload regime to end. The *most common cause* is that server 1, which is receiving no jobs, becomes idle, dropping its average value to 0. In this case, the very next arrival will be dispatched to server 1. With very high probability, that arrival will be a value 10 job, which will again send us back to the overload regime, where all arrivals are sent to server 2. In fact, server 1 will likely repeatedly alternate between having no jobs and a single value 10 job, setting the effective arrival rate at server 2 to $2\lambda/3$, which is greater than the departure rate μ, as $\rho > 3/4$. The only catch is that with extremely low probability, when server 1 is idle, a value 1 job might be the next arrival, rather than a value 10 job. This creates a new regime where all value 10 jobs start going to server 1; however, with high probability this regime is (relatively) short-lived because it only lasts until that value 1 job leaves the system, which will happen quickly since the value 1 job was the first to join the queue at server 1. Once server 2 has built up enough jobs, this rare event barely affects its queue length.

A *less common cause* for the overload regime to end is that server 2 loses its value 1 job. This turns out to be unlikely for two reasons: (i) The value 1 job at server 2 arrives into an already busy queue, so if ρ is sufficiently high, it will take a while until it departs (given PS scheduling). (ii) While server 2 has a value 1 job and server 1 does not, new value 1 arrivals are twice as likely to come to server 2 (remember that server 2's effective arrival rate is $2\lambda/3$), thus server 2 is more likely to get any new value 1 job that arrives, hence "replenishing" its supply of value 1 jobs.

Again, the point is that once the queue length at server 2 builds sufficiently, any departure from the overload regime is quickly reversed, returning us to the overload regime.

5.6.3 Results under sharply bimodal distributions

In this section we explore the efficacy of policies under a class of value distributions exhibiting a high degree of variability. These distributions are of interest for two reasons. First, some of the policies are amenable to asymptotic analysis under these distributions, allowing us to formally establish trends seen in Section 5.5, albeit in a more limited setting. Second, the VITA and C-MU behave quite differently in this regime, as typified by Fig. 5.2 (f) and it is insightful to formally explore these observations.

We write $V \sim \mathrm{SBD}(p)$, indicating that V obeys a *sharply bimodal distribution* with a "sharpness" parameter $p > 1/2$, such that

$$V \sim \begin{cases} \dfrac{1-p}{p} = \text{“low-value”} & \text{w.p. } p \\[2ex] \dfrac{p}{1-p} = \text{“high-value”} & \text{w.p. } 1-p. \end{cases}$$

This distribution satisfies the convention $\mathbb{E}[V] = 1$, where the low-value jobs (i.e., those with value $(1-p)/p$) constitute a $1-p$ fraction of the total value, and the high-value jobs (i.e., those with value $p/(1-p)$), constitute a p fraction of the total value. We are typically interested in this distribution in the asymptotic regime where $p \to 1$; here high-value jobs are extremely rare yet comprise essentially all of the value. Note that as $p \to 1/2$, $\mathrm{SBD}(p)$ converges to a constant; we name this family of distributions for their "sharp" behavior that emerges only when $p \approx 1$.

Our first result for sharply bimodal distributions concerns the asymptotic optimality of VITA when $V \sim \mathrm{SBD}(p)$ as $p \to 1$ and $\rho < 1/2$.

Theorem 5.8. *Let* $0 < \rho < 1/2$ *and* $V \sim \mathrm{SBD}(p)$ *in a system with two identical servers. As* $p \to 1$*,* VITA *is asymptotically optimal in the sense that for any policy* P*, we have* $\lim_{p\to 1} \mathbb{E}[VT]^{\mathrm{VITA}}/\mathbb{E}[VT]^{\mathrm{P}} \le 1$.

Proof. Consider a static dispatching policy, P', that reserves server 1 for low-value jobs and server 2 for the high-value jobs. The system will be stable since, when $\rho < 1/2$, either server can process all arrivals even if operating alone. Since VITA is the optimal static policy, the performance of P' gives an upper bound on the performance of VITA. Taking the limit as $p \to 1$, we have the bound

$$\lim_{p\to 1} \mathbb{E}[VT]^{\mathrm{VITA}} \le \lim_{p\to 1} \mathbb{E}[VT]^{\mathrm{P}'} = \lim_{p\to 1} \left(\frac{1-p}{\mu - p\lambda} + \frac{p}{\mu - (1-p)\lambda} \right) = \frac{1}{\mu}.$$

Finally, under *any* policy P, we have $\mathbb{E}[VT]^{\mathrm{P}} \ge \mathbb{E}[V]/\mu = 1/\mu$, so $\lim_{p\to 1} \mathbb{E}[VT]^{\mathrm{VITA}}/\mathbb{E}[VT]^{\mathrm{P}} \le 1$. □

Next, we prove that although VITA may not be an asymptotically optimal dispatching policy under higher loads, it continues to dominate the performance of JSQ for all $\rho < 1$, as long as the value distribution is sufficiently "sharp."

Theorem 5.9. *Let* $V \sim \mathrm{SBD}(p)$ *in a system with two identical servers. As* $p \to 1$*,* VITA *asymptotically performs as well as* JSQ*, if not better. That is,* $\lim_{p\to 1} \mathbb{E}[VT]^{\mathrm{VITA}}/\mathbb{E}[VT]^{\mathrm{JSQ}} \le 1$*. Moreover, this inequality is tight as* $\rho \to 1$.

Proof. The case where $\rho < 1/2$ is a direct consequence of Theorem 5.8.

Now consider the case where $\rho > 1/2$. We will upper bound the performance of VITA by a family of static policies, $\mathrm{P}(r)$, parametrized by $r \in (0,1)$. The VITA-like policy $\mathrm{P}(r)$ sends as many high-value jobs to server 2 as possible, while setting the arrival rate of jobs to server 1 at $\lambda_1(r) \equiv \lambda/2 + (\mu - \lambda/2)r$ and the arrival rate of jobs to server 2 at $\lambda_2(r) \equiv \lambda/2 - (\mu - \lambda/2)r$. Note that $\mathrm{P}(r)$ is a stable policy. Since $\rho > 1/2$, for p sufficiently close to 1, we must have $p > \mu/\lambda = 1/(2\rho)$, which ensures that the arrival rate of high-value jobs, $\lambda(1-p)$, is less than the total arrival rate of jobs to server 2, $\lambda_2(r) = \lambda/2 - (\mu - \lambda/2)r$, for all r. Consequently, whenever p is sufficiently large (i.e., $p > 1/(2\rho)$), $\mathrm{P}(r)$ sends *all* high-value jobs to server 2 for all $r \in (0,1)$.

Consider an arbitrary arrival that enters a system and is dispatched via $\mathrm{P}(r)$. There are three mutually exclusive possibilities when $p > 1/(2\rho)$:

- with probability $\lambda_1(r)/\lambda$ we have a low-value arrival that is sent to server 1,
- with probability $\lambda_2(r)/\lambda - (1-p)$, we have a low-value arrival that is sent to server 2,

- and with probability $1-p$, we have a high-value arrival that is sent to 2.

Using this information, we find that

$$\mathbb{E}[VT]^{\mathrm{P}(r)} = \left(\frac{\lambda_1(r)}{\lambda}\right)\left(\frac{1-p}{p}\right)\left(\frac{1}{\mu-\lambda_1(r)}\right) + \left[\left(\frac{\lambda_2(r)}{\lambda} - (1-p)\right)\left(\frac{1-p}{p}\right) + (1-p)\left(\frac{p}{1-p}\right)\right]\left(\frac{1}{\mu-\lambda_2(r)}\right).$$

Now recall that $\mathbb{E}[VT]^{\mathrm{JSQ}} \geq 1/(2\mu-\lambda)$, while $\mathrm{P}(r)$ upper bounds the performance of VITA, yielding

$$\lim_{p\to 1}\left(\frac{\mathbb{E}[VT]^{\mathrm{VITA}}}{\mathbb{E}[VT]^{\mathrm{JSQ}}}\right) \leq \lim_{p\to 1}\left(\frac{\mathbb{E}[VT]^{\mathrm{P}(r)}}{1/(2\mu-\lambda)}\right) = \frac{2}{1+r}.$$

Taking an infimum over $r \in (0,1)$, we have the desired result.

To see that the inequality is tight as $\rho \to 1$, first observe that the family of static policies $\mathrm{P}(r)$ subsumes VITA, as this family of policies includes all static policies that isolate the high-value jobs. Next, observe that $\lim_{\rho\to 1}\left(\mathbb{E}[VT]^{\mathrm{JSQ}} \cdot (2\mu-\lambda)\right) = 1$. Consequently, both the upper bound on the performance of VITA (after optimizing for r) and the lower bound on the performance of JSQ are tight as $\rho \to 1$, completing the proof. □

Remark: Note that although JSQ outperforms VITA in Figure 5.2 (f) as $\rho \to 1$ (i.e., for the highest values of ρ), this does not contradict Theorem 5.9. Under distribution (f), we only have $p \approx 1$, while Theorem 5.9 holds in the limit as $p \to 1$.

We proceed to state a result comparing the asymptotic performance of LAVA with that of JSQ under the $\mathbb{E}[VT]$ metric, but we first state a Lemma that will be helpful in proving this result.

Lemma 5.10. *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers. As $p \to 1$:*

- *The limiting distribution of the number of low value jobs, N_ℓ, under LAVA converges weakly to the limiting distribution of the total number of jobs, N, under JSQ, and $\mathbb{E}[N_\ell]^{\mathrm{LAVA}} \to \mathbb{E}[N]^{\mathrm{JSQ}}$.*
- *The limiting distribution of the number of high-value jobs, N_h, under LAVA converges weakly to the zero distribution, and $\mathbb{E}[N_h]^{\mathrm{LAVA}} \to 0$.*
- *The limiting distribution of the length of the shorter queue (i.e., the instantaneous minimum length of the two queues), N_{short}, under LAVA converges to the limiting distribution of N_{short} under JSQ, and $\mathbb{E}[N_{\mathrm{short}}]^{\mathrm{LAVA}} \to \mathbb{E}[N_{\mathrm{short}}]^{\mathrm{JSQ}}$.*

Proof. Deferred to appendix. □

Theorem 5.11. *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers. As $p \to 1$,* LAVA *asymptotically performs at least as well as* JSQ. *That is,* $\lim_{p\to 1}\mathbb{E}[VT]^{\mathrm{LAVA}}/\mathbb{E}[VT]^{\mathrm{JSQ}} \leq 1$. *Moreover, there exists an asymptotic regime where $\rho \to 1$ and $p \to 1$, such that* $\mathbb{E}[VT]^{\mathrm{LAVA}}/\mathbb{E}[VT]^{\mathrm{JSQ}} \to 1/2$.

Proof. First, fix $\rho \in (0,1)$, and consider a two-server system with values $V \sim \mathrm{SBD}(p)$. Let T_ℓ and T_h be the response time of a low-value and high-value job respectively. As $p \to 1$, from Lemma 5.10, we have that $\mathbb{E}[N_\ell]^{\mathrm{LAVA}} \to \mathbb{E}[N]^{\mathrm{JSQ}}$, from which Little's Law yields

$$\lim_{p\to 1}\left(\mathbb{E}[T_\ell]^{\mathrm{LAVA}}\right) = \lim_{p\to 1}\left(\frac{\mathbb{E}[N_\ell]^{\mathrm{LAVA}}}{\lambda p}\right) = \frac{\mathbb{E}[N]^{\mathrm{JSQ}}}{\lambda} = \mathbb{E}[T]^{\mathrm{JSQ}} = \mathbb{E}[T_\ell]^{\mathrm{JSQ}}.$$

That is, a low-value job under LAVA experiences a mean response time *asymptotically* equal to that experienced under JSQ as $p \to 1$. We proceed to show that high-value jobs experience even lower response times, which is sufficient to show that LAVA asymptotically does no worse than JSQ with respect to the $\mathbb{E}[VT]$ metric as $p \to 1$.

Observe that under *all* value distributions, $\mathbb{E}[T|V=v]^{\text{LAVA}}$ (i.e., the mean response time of a job with value v under LAVA) is a nonincreasing function of v: when a job arrives to a system, if its value, v, were any higher, the job would either be routed to the same queue or a queue of equal or shorter length. Once a job is in the system, if its value, v, were any higher, it would reduce (or keep fixed) the arrival rate of new jobs coming into its queue. Hence, jobs with higher values are no worse off than jobs with lower values with respect to mean response times, so $\mathbb{E}[T_h]^{\text{LAVA}} \le \mathbb{E}[T_\ell]^{\text{LAVA}} \to \mathbb{E}[T]^{\text{JSQ}} = \mathbb{E}[T_h]^{\text{JSQ}}$ as $p \to 1$. Since both types of jobs are no worse off under LAVA than under JSQ, it follows that $\lim_{p\to 1} \mathbb{E}[VT]^{\text{LAVA}}/\mathbb{E}[VT]^{\text{JSQ}} \le 1$ as claimed.

In order to prove the remaining claim that LAVA can outperform JSQ by a factor of two under heavy traffic, we must more accurately quantify $\mathbb{E}[T_\ell]^{\text{LAVA}}$ as $p \to 1$, rather than simply providing a bound as we have done above. We proceed by considering the state of the LAVA system seen by a high-value arrival. By PASTA, this high-value arrival sees a system in steady state, and as $p \to 1$, N_ℓ will be distributed like the number of jobs, N, in a JSQ system, and $\mathbb{E}[N_h]^{\text{LAVA}} \to 0$. Since this job sees only low-value jobs, it will be routed to the shorter queue, say server 2's queue, which will cease to accept further low-value jobs, in accordance with the LAVA policy. Server 2 will only accept high-value jobs, which arrive at a rate of $(1-p)\lambda \to 0$.[5] Hence, the high-value job must share server 2 only with those low-value jobs *already present* in the system when it arrives. The shorter queue contains N_{short} jobs (of independent exponentially distributed sizes S_i, each with mean $1/\mu$). By Lemma 5.10, $\mathbb{E}[N_{\text{short}}]^{\text{LAVA}} \to \mathbb{E}[N_{\text{short}}]^{\text{JSQ}}$, yielding

$$\lim_{p\to 1}\left(\mathbb{E}[T_\ell]^{\text{LAVA}}\right) = \mathbb{E}\left[\frac{\sum_{i=1}^{N_{\text{short}}+1} S_i}{N_{\text{short}}+1}\right]^{\text{JSQ}} = \frac{(\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2)\mathbb{E}[S_i]}{2} = \frac{\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2}{2\mu}.$$

Consequently, we have

$$\lim_{p\to 1}\mathbb{E}[VT]^{\text{LAVA}} = \lim_{p\to 1}\left(p\left(\frac{1-p}{p}\right)\mathbb{E}[T_\ell]^{\text{LAVA}} + (1-p)\left(\frac{p}{1-p}\right)\left(\frac{\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2}{2\mu}\right)\right) = \frac{\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2}{2\mu},$$

where we use the fact that $\lim_{p\to 1}\left((1-p)\mathbb{E}[T_\ell]^{\text{LAVA}}\right) = 0 \cdot \mathbb{E}[T]^{\text{JSQ}} = 0$.

Now consider the case where ρ is no longer fixed, and we in fact have $\rho \to 1$. Then we can evaluate the following iterated limit:

$$\lim_{\rho\to 1}\left(\lim_{p\to 1}\left(\frac{\mathbb{E}[VT]^{\text{LAVA}}}{\mathbb{E}[VT]^{\text{JSQ}}}\right)\right) = \lim_{\rho\to 1}\left(\frac{\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2}{2\mu\cdot\mathbb{E}[T]^{\text{JSQ}}}\right) = \lim_{\rho\to 1}\left(\rho\cdot\frac{\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}+2}{\mathbb{E}[N]^{\text{JSQ}}}\right) = \frac{1}{2},$$

where we have used the facts that $\mathbb{E}[N]^{\text{JSQ}} \to \infty$ and $\mathbb{E}[N_{\text{short}}]^{\text{JSQ}}/\mathbb{E}[N]^{\text{JSQ}} \to 1/2$ as $\rho \to 1$. The first fact is clear, and the latter fact follows form Foschini and Salz [51]: as $\rho \to 1$ the

[5] Since $p \to 1$, the high-value jobs are arbitrarily more valuable than the low-value jobs by a factor of $\left(\frac{p}{1-p}\right)^2 \to \infty$, so server 1's queue will never grow so long as to resume the arrival process of low-value jobs into server 2's queue during the high-value job's residence.

JSQ instantaneous queue lengths are asymptotically balanced, so the length of the shorter queue is on the order of half the total number of jobs in the system. The convergence of the iterated limit implies the existence of a sequence of pairs $\{(p_n, \rho_n)\}_{n=1}^{\infty} \to (1,1)$ (i.e., an "asymptotic regime") under which $\mathbb{E}[VT]^{\mathrm{LAVA}}/\mathbb{E}[VT]^{\mathrm{JSQ}} \to 1/2$, as claimed.

□

Corollary 5.12. *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers. There exists an asymptotic regime where $\rho \to 1$ and $p \to 1$, such that we have the following ratio between the performance of various policies:*

$$\mathbb{E}[VT]^{\mathrm{RND}} : \mathbb{E}[VT]^{\mathrm{VITA}} : \mathbb{E}[VT]^{\mathrm{JSQ}} : \mathbb{E}[VT]^{\mathrm{LAVA}} \to 4:2:2:1.$$

Proof. The result follows immediately from Proposition 5.5 and Theorems 5.9 and 5.11. □

We explain why C-MU also performs well in this regime. Just as LAVA essentially employs JSQ until the arrival of a rare high-value job, C-MU essentially employs a variant of RND (where a job is sent to an idle server whenever possible, but dispatching is otherwise random) until the arrival of such a job. Under both LAVA and C-MU, a high-value job is subsequently sent to the server with the shorter queue. The server receiving the high value job will essentially cease to receive arrivals until the completion of that job. It may appear that despite all this LAVA should outperform C-MU because queue lengths under JSQ are shorter than those under RND. We note, however, that under RND, the time-average length of the *shorter* queue is half that of an *arbitrary* queue.

Finally, we ask what would happen if the two-server system was replaced by a single server with twice the service rate, resulting in an M/M/1/PS system. At first it might seem that the single server is superior, but we need to remember that our metric is $\mathbb{E}[VT]$. Corollary 5.13 shows that there are regimes for which a two-server system is superior.

Corollary 5.13. *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers operating at rate μ. Then there exist $\rho^* < 1$ and $p^* < 1$, such that $\mathbb{E}[VT]^{\mathrm{LAVA}} < \mathbb{E}[VT]^{\mathrm{SINGLE}}$, where* SINGLE *refers to a single* PS *server operating at rate 2μ.*

Proof. From Theorem 5.11, there exists a regime with $\rho \to 1$, where the performance of LAVA is *strictly* better than JSQ. Moreover, we know from [51] that when $\rho \to 1$, the mean response times under JSQ and SINGLE are arbitrarily close, and since V and T are independent under both JSQ and SINGLE, their performance under $\mathbb{E}[VT]$ is also arbitrarily close. Hence, by continuity there exist $\rho^* < 1$ and $p^* < 1$ such that $\mathbb{E}[VT]^{\mathrm{LAVA}} < \mathbb{E}[VT]^{\mathrm{SINGLE}}$. □

5.7 A (sometimes) far better policy: Gated VITA (G-VITA)

In Sections 5.5 and 5.6, we saw that the VITA policy often performs poorly relative to most of the other policies, except under sharp bimodal value distributions such as distribution (f) (cf. the definition of $\mathrm{SBD}(p)$ in Section 5.6.3). For such distributions, VITA is asymptotically optimal (as the value distribution grows increasingly sharp) for $\rho < 1/2$. Although VITA continues to perform modestly well for these distributions when $\rho > 1/2$, it does not perform nearly as well as LAVA when $\rho \to 1$.

To understand why VITA does not perform as well under high loads, observe that the VITA policy's strength lies in isolating the highest value jobs. However, when load is particularly high, the relative efficacy of this isolating effect is limited because high-value jobs must share their "dedicated" server with too many low-value jobs, in order to ensure system stability. In this section, we explore how adding a "non-static component" to VITA can greatly reduce the number of low-value jobs utilizing this "dedicated" server, without sacrificing system stability at higher loads. We present a policy, Gated VITA (G-VITA), that outperforms LAVA by *an arbitrary factor* under a particular high load regime.

5.7.1 G-VITA

We define the G-VITA policy with parameter g for two-server systems with bimodal value distributions.

- G-VITA sends low-value jobs to server 2 if and only if the number of low-value jobs present at server 2 is at most g.
- G-VITA always sends high-value jobs to server 2.

We can interpret server 2 as having a limited number of slots reserved for use by low-value jobs. When all of these slots are occupied, a "gate" will close and bar the entry of any further low-value arrivals (sending them to server 1). The gate remains closed until a low-value jobs departs. Note that the gate never bars the entry of high-value jobs. Moreover, while the queue at server 2 can hold up to g low-value jobs and any number of high-value jobs, the queue at server 1 only holds low-value jobs.

While we have defined G-VITA only for bimodal value distributions, the G-VITA policy can be extended to general value distributions by classifying jobs as "low-value" and "high-value" jobs in an appropriate manner.

There exist values of $\rho < 1$ for which the G-VITA policy with fixed parameter g is unstable; however, we will show that for any $\rho < 1$, stability can be guaranteed by requiring g to be sufficiently high.

Lemma 5.14. *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers. As $p \to 1$, the* G-VITA *policy with parameter g under load ρ is stable whenever $\rho < 1$ and either*

$$g > -\frac{\log(2-2\rho)}{\log(2\rho)} - 1, \qquad \textit{or alternatively,} \qquad g > \log_2\left(\frac{\rho}{1-\rho}\right).$$

Proof. Deferred to appendix. □

Recall that when $V \sim \mathrm{SBD}(p)$ in a regime where $\rho \to 1$ and $p \to 1$, high-value jobs under RND, JSQ, VITA, or LAVA share their server with an average number of low-value jobs on the order of $\frac{\rho}{1-\rho}$. Meanwhile, we have just shown that there exist stable G-VITA policies in the same regime such that high-value jobs need only share their server with about $\log_2\left(\frac{\rho}{1-\rho}\right)$ low-value jobs. That is, in this regime, G-VITA "protects" high-value jobs by having them share their server with *exponentially fewer* low-value jobs. This phenomenon lies at the heart of the following result.

Theorem 5.15. *Let* $V \sim \mathrm{SBD}(p)$ *in a system with two identical servers and let* $\mathrm{P} \in \{\mathrm{RND}, \mathrm{JSQ}, \mathrm{VITA}, \mathrm{LAVA}\}$*. There exists an asymptotic regime with* $\rho \to 1$*,* $p \to 1$*, and (*G-VITA parameter*)* $g \to \infty$*, s.t.* $\mathbb{E}[VT]^{\text{G-VITA}}/\mathbb{E}[VT]^{\mathrm{P}} \to 0$.

Proof. We prove the result in the case where P is RND by giving an upper bound on $\mathbb{E}[VT]^{\text{G-VITA}}$, then dividing by $\mathbb{E}[VT]^{\mathrm{RND}} = 1/(\mu - \lambda/2)$ and taking a limit as $\rho \to 1$. We will express p and g as parametric functions of ρ. The result for the remaining policies follows from the fact that the performance of RND is within a bounded factor of that of the other policies in this regime (cf. Corollary 5.12).

Let T_ℓ and T_h be the response times of low-value and high-value jobs, respectively. In order to give an upper bound on $\mathbb{E}[VT]^{\text{G-VITA}}$, we first consider the mean response times of high-value jobs under G-VITA, $\mathbb{E}[T_h]^{\text{G-VITA}}$. High-value jobs arrive to server 2 according to a Poisson process with rate $(1-\lambda)p$, and when one or more such jobs are present, they depart server 2 with rate $h_2 \cdot \mu/(\ell_2 + h_2)$, where ℓ_2 and h_2 are the number of low-value and and high-value jobs present at server 2, respectively. Since $\ell_2 \leq g$ and $h_2 \geq 1$ (when departures of high-value jobs from server 2 are possible) we can lower bound the departure rate of high-value jobs by $\mu/(g+1)$. Consequently,

$$\mathbb{E}[T_h]^{\text{G-VITA}} \leq \frac{1}{\mu/(g(\rho)+1) - (1-p(\rho))\lambda}.$$

Now consider the mean response time of low-value jobs under G-VITA, $\mathbb{E}[T_\ell]^{\text{G-VITA}}$. From Lemma 5.14, we know that as $p \to 1$, this quantity is finite for all $\rho < 1$, as long as $g > \log_2(\rho/(1-\rho))$. Consequently, within this asymptotic regime we allow p and g to vary with ρ as $\rho \to 1$, subject to

$$p(\rho) \geq 1 - \frac{1}{\mathbb{E}[T_\ell]^{\text{G-VITA}}} \quad \text{and} \quad g(\rho) \geq \log_2\left(\frac{\rho}{1-\rho}\right) \quad (g(\rho) \in \mathbb{N}),$$

where the first constraint ensures that $(1-p(\rho))\mathbb{E}[T_\ell]^{\text{G-VITA}} \leq 1$ and the second constraint guarantees stability. Note that these constraints imply that $p(\rho) \to 1$ and $g(\rho) \to \infty$ when $\rho \to 1$ as claimed.[6] We proceed to complete the proof by taking the required limit:

$$\begin{aligned}
\lim_{\rho\to 1} \frac{\mathbb{E}[VT]^{\text{G-VITA}}}{\mathbb{E}[VT]^{\mathrm{RND}}} &\leq \lim_{\rho \to 1}\left(\frac{p(\rho)\left(\frac{1-p(\rho)}{p(\rho)}\right)\mathbb{E}[T_\ell]^{\text{G-VITA}} + (1-p(\rho))\left(\frac{p(\rho)}{1-p(\rho)}\right)\left(\frac{\mu}{g(\rho)+1} - (1-p(\rho))\lambda\right)^{-1}}{1/(\mu - \lambda/2)}\right) \\
&\leq \lim_{\rho\to 1}\left(\frac{1 + \log_2\left(\frac{\rho}{1-\rho}\right)\Big/\mu}{(\mu(1-\rho))^{-1}}\right) = 0.
\end{aligned}$$

□

[6]Note that the right-hand side of the formula bounding $p(\rho)$ actually depends on $p(\rho)$ (i.e., "computing" the bound on $p(\rho)$ involves solving a fixed point problem), but the effect of $p(\rho)$ on this bound vanishes when $p(\rho) \to 1$, as argued in the proof of Lemma 5.14, so a $p(\rho)$ satisfying this constraint may be found without any problems.

5.7.2 G-VITA simulations

In this section we use simulations to numerically compare the performance of G-VITA with JSQ and LAVA under the value distribution (f). We have held the G-VITA parameter fixed at $g = 5$ for all values of ρ plotted. As shown in Fig. 5.3, G-VITA performs very well as $\rho \to 1$, strongly outperforming both JSQ and LAVA, as expected. However, the performance of G-VITA is very poor at low loads. This is because the relatively high G-VITA parameter, g, forces high-value jobs to share their server with unnecessarily many low-value jobs under low loads. Even if we vary g with ρ, consistently strong performance is still unattainable because $g \in \mathbb{N}$.

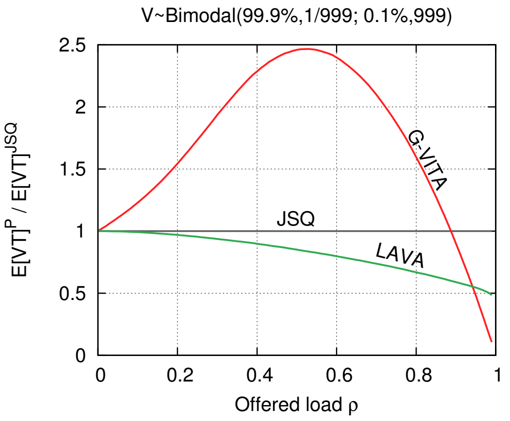


Figure 5.3: G-VITA dominates JSQ and LAVA under value distribution (f) as $\rho \to 1$ in accordance with Theorem 5.15.

One can, however, come up with "G-VITA inspired policies" that do very well across all loads under sharply variable (if not all) distributions. For, example, we could redefine G-VITA to be less "eager" in occupying all g available slots at server 2, e.g., as long as there are fewer than g low value jobs at server 2, dispatch low-value jobs using JSQ, rather than always sending them to server 2. Alternatively, consider a policy that routes low value jobs so as to maintain a fixed ratio between the queue lengths at servers 1 and 2 (e.g., attempt to keep server 1 four times as long as server 2), while sending all high-value jobs to the server with the shorter expected queue length. Such a policy would help isolate high-value jobs from low-value jobs to a greater extent than VITA or LAVA (although not as much as G-VITA), while not hurting high-value jobs too much at lower loads. We have verified via simulation that such policies perform well for all ρ (not shown due to lack of space).

5.8 More complex policies via the First Policy Iteration (FPI)

Thus far, we have considered only simple, intuitive dispatching policies. In this section, we analyze the value-aware dispatching problem in the framework of Markov decision processes (MDP) [78, 108, 114]. This will lead us to policies that often perform better than our existing policies, but are more complex and less intuitive.

We start with a tutorial example to explain the FPI approach. Consider a two-server system. If this system were to use the RND dispatching policy, arrivals would be randomly split between the two servers. Instead of using RND, we propose a "first policy iteration" on RND, which we shall call FPI-RND, whereby, an arrival is dispatched so as to minimize the *overall* $\mathbb{E}[VT]$ (given the current state of system), *under the assumption that all future arrivals will be dispatched via* RND.[7] Note that the assumption on how future jobs are routed is actually inaccurate, as future arrivals will continue to be routed via FPI-RND. Here, RND is referred to as the **basic policy** that is *improved upon* by FPI.

The central notion of FPI is the **value function**,[8] denoted by $\eta_{\mathbf{z}}(\alpha)$, where $\mathbf{z}$ is the system state (i.e., the number of jobs at each server and their values) and α is the basic policy being improved (e.g., RND, VITA, etc.). In defining the value function we view the system as incurring a "penalty" of magnitude v for each unit of time a job of value v spends in the system, so that minimizing $\mathbb{E}[VT]$ is equivalent to minimizing the expected rate at which penalty is incurred. Let $C_{\mathbf{z}}(\alpha, t)$ denote the cumulative penalty incurred under policy α during the time interval $(0, t)$ when the initial state is $\mathbf{z}$, and let $r(\alpha)$ denote the mean (equilibrium) rate at which penalty is incurred under α. More formally,

$$C_{\mathbf{z}}(\alpha, t) \equiv \mathbb{E}\left[\int_0^t \left(\sum_{i=1}^m v_i^{\text{sum}}(\tau)\right) d\tau \,\middle|\, \text{the state at time } \tau = 0 \text{ is } \mathbf{z}\right]^\alpha,$$

$$r(\alpha) \equiv \mathbb{E}\left[\lim_{t\to\infty}\left(\frac{1}{t}\int_0^t \left(\sum_{i=1}^m v_i^{\text{sum}}(\tau)\right) d\tau\right)\right]^\alpha = \lim_{t\to\infty}\left(\frac{\lambda t \cdot \mathbb{E}[VT]^\alpha}{t}\right) = \lambda \cdot \mathbb{E}[VT]^\alpha,$$

where $v_i^{\text{sum}}(\tau)$ denotes the sum of the values of the jobs at server i at time τ. With this framework, we can define the **value function**, $\eta_{\mathbf{z}}(\alpha)$, as the expected difference in cumulative (infinite time-horizon) penalties incurred between a system initially in state $\mathbf{z}$ and a system in equilibrium,

$$\eta_{\mathbf{z}}(\alpha) \equiv \lim_{t\to\infty}\left(C_{\mathbf{z}}(\alpha, t) - r(\alpha) \cdot t\right).$$

Hence, $\eta_{\mathbf{z}_2}(\alpha) - \eta_{\mathbf{z}_1}(\alpha)$ quantifies the benefit of starting in $\mathbf{z}_2$ rather than $\mathbf{z}_1$. In general, value functions enable *policy iteration*, a procedure that, under certain conditions, converges to the optimal policy. Here, due to the complexity of the system, we are limited to only the **first policy iteration** (FPI) step.

In our case, the value function depends on the dispatching policy, α, and the system state, $\mathbf{z} \equiv (z_1, \ldots, z_m)$, where each $z_i \equiv (v_{i,1}, \ldots, v_{i,n_i})$, gives the state of server i. The key idea with policy iteration is to consider the optimal deviation from the *basic policy* α for *one decision*, that is, dispatching one new arrival with value v, and then returning to α so that the expected future costs are given by the value function. The optimal decision corresponds to a new improved policy α',

$$\alpha'(\mathbf{z}, v) = \underset{\mathbf{z}' \in \mathcal{A}(\mathbf{z}, v)}{\text{argmin}}\ \eta_{\mathbf{z}'}(\alpha) - \eta_{\mathbf{z}}(\alpha), \tag{5.4}$$

[7]Actually implementing such a policy requires some calculations, which will be shown later in this section.

[8]The word "value" in "value function" is not directly related to the use of the word "value," as used elsewhere in the chapter.

where $\mathcal{A}(\mathbf{z}, v)$ denotes the states that can result from dispatching a value v arrival to one of the m servers. Note that the resulting policy, $\alpha'(\mathbf{z}, v)$, also depends on the value of the new arrival, v, and although $\alpha'(\mathbf{z}, v)$ always assumes further arrivals will be routed according to α, the actual policy will continue to dispatch according to the one-step optimal deviations described above.

Note that LAVA, discussed in Section 5.4.5, is based on the assumption that no jobs arrive afterwards. In contrast, FPI assumes that after the current decision, each server continues to receives a stream of arrivals, based on how they would be dispatched by the dispatching policy α.

5.8.1 FPI policies

In this section we first determine the value function of a basic policy α and subsequently use it to derive the FPI policy. Due to the complex state-space, it is difficult to determine the value function of an arbitrary basic policy (e.g., LAVA). Therefore, as in [92], we use a static basic policy (e.g., RND or VITA). In this case, the arrival process decomposes to m independent Poisson processes, and it is sufficient to derive the value function for each M/M/1-PS queue separately as $\eta_{\mathbf{z}}(\alpha) = \sum_{i=1}^{m} \eta_{\mathbf{z}_i}^{(i)}(\alpha)$, where $\eta_{z_i}^{(i)}(\alpha)$ denotes the value function of server i in state z_i. Letting v^{sum} be the sum of the values of all jobs in an M/M/1-PS queue in state z, the corresponding value function is given by Proposition 5.16.

Proposition 5.16. *The value function for the M/M/1-PS queue with arrival rate λ, service rate μ, and values V is given by*

$$\eta_z = \frac{\lambda n(n+1)}{2(\mu-\lambda)(2\mu-\lambda)}\,\mathbb{E}[V] + \frac{n+1}{2\mu-\lambda}\,v^{\text{sum}} + c, \tag{5.5}$$

where c is a constant.

Proof. Deferred to appendix. □

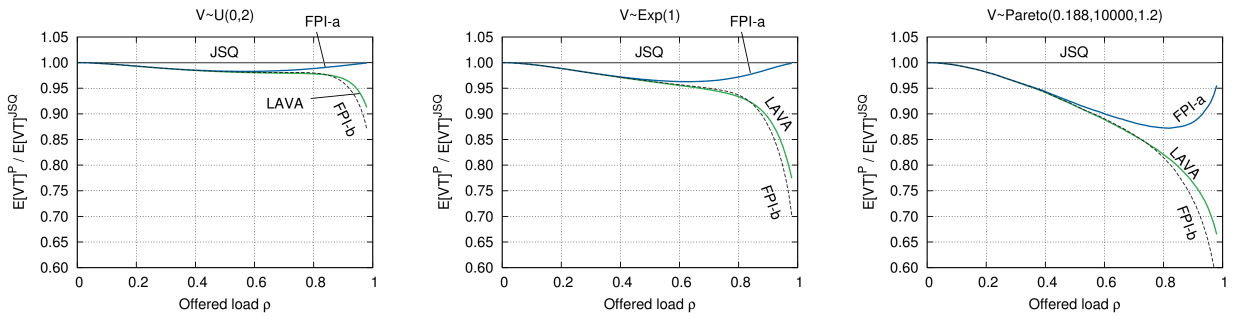
As previously mentioned, we assume a static basic policy α. Since α is static, it necessarily defines an independent server-specific Poisson arrival process at each server with arrival rate λ_i and value distribution V_i. Consequently, these server-specific arrival processes (λ_i, V_i), allow us to use the value function from (5.5) to derive the FPI policy using (5.4).

Proposition 5.17. *For a static basic policy α, yielding server-specific arrival processes (λ_i, V_i), the corresponding FPI policy routes a job of value v to the server given by*

$$\alpha'(v) = \underset{i}{\operatorname{argmin}} \quad \frac{1}{2\mu_i-\lambda_i}\left(\frac{\lambda_i\,\mathbb{E}[V_i](n_i+1)}{\mu_i-\lambda_i} + v_i^{\text{sum}} + (n_i+2)v\right). \tag{5.6}$$

Proof. Deferred to appendix. □

Remark: We note that letting $\lambda_i \to 0$ in (5.6) reduces to LAVA given in (5.2), and letting $\lambda_i \to \mu_i$ in (5.6) reduces to JSQ.



(a) (b) (c)

Figure 5.4: Performance of FPI-a, FPI-b, and LAVA relative to JSQ in a two-server system, with value distributions (a) Uniform, (b) Exponential, (c) Pareto from Table 5.1. Each point corresponds to the mean performance with about 100 million jobs.

5.8.2 Enhancing FPI policies using discounting

In this section we provide a novel idea for enhancing FPI policies. Inspired by LAVA, which ignores future arrivals completely, we consider modifications of FPI policies where we discount the impact of future arrivals on dispatching. We identify the terms corresponding to the harm caused to future jobs (the term with $\mathbb{E}[V_i]$) the present jobs (the terms with v_i^{sum} and v) in (5.6), and introduce an additional *weight parameter* γ to discount the former. This yields

$$\boxed{\alpha'_\gamma(v) = \underset{i}{\operatorname{argmin}} \quad \frac{1}{2\mu_i - \lambda_i}\left(\gamma \cdot \frac{\lambda_i\,\mathbb{E}[V_i](n_i+1)}{\mu_i - \lambda_i} + v_i^{\text{sum}} + (n_i+2)v\right).} \tag{5.7}$$

Suppose that the servers are identical, $\mu_i = \mu$, and the basic policy α balances the load ($\lambda_i = \lambda/m$). Then,

$$\boxed{\alpha'_\gamma(v) = \underset{i}{\operatorname{argmin}} \quad \left(\gamma \cdot \frac{\lambda\,\mathbb{E}[V_i](n_i+1)}{\mu - \lambda/m} + v_i^{\text{sum}} + n_i v\right).} \tag{5.8}$$

We observe that when $\gamma = 1$, this results in the original FPI policy presented in Proposition 5.17, while when $\gamma = 0$, this results in the LAVA policy (regardless of the basic policy used).

5.8.3 FPI simulations

In this section we use simulations to numerically compare the performance of two FPI policies with JSQ and LAVA:

1. **FPI-a** uses a weight parameter of $\gamma = 1$ (i.e., the FPI policy presented in Proposition 5.17) and is based on RND.
2. **FPI-b** uses a weight parameter of $\gamma = 1/20$ (i.e., future arrivals are heavily discounted). Rather than being based on RND, this policy is based on a variation of VITA where load is equalized between the two servers.

The numerical results relative to JSQ are shown in Fig. 5.4. When ρ is low, all FPI policies perform like LAVA. Meanwhile, the performance of FPI-a converges to that of JSQ as $\rho \to 1$, in accordance with the remark made after Proposition 5.17. Consequently, FPI-a performs worse than LAVA. The FPI-b policy, however, outperforms LAVA under high ρ.

5.9 Conclusion

This chapter presents the first comprehensive study of dispatching policies that aim to minimize value-weighted response times under Process-Sharing scheduling. We propose a large number of novel dispatching policies and compare these under a range of workloads, showcasing the fact that the value distribution and load can greatly impact the ranking of the policies. We also prove several intriguing results on the asymptotic behavior of these policies. Note that while we have assumed that job values are known exactly, most of our results generalize easily where jobs belong to classes and only the mean value of each class is known.

As value-driven dispatching is a very new problem, there remains ample room for future work on analyzing the policies in this chapter and proposing new ones. Other directions for extending the results in this chapter include considering more complicated arrival processes and job size distributions, possibly with correlations between consecutive arrivals. Additionally, one could consider alternative value-weighted response time metrics, including higher moments and distribution tails.

Chapter 6

Concluding Remarks

In this dissertation, we presented a new methodological contribution, Clearing Analysis on Phases (CAP) for solving a broad class of multi-dimensional Markov chains (Chapter 2), with applications to the analysis of a variety of queueing systems. We then applied CAP to address what we have called the malware cleanup problem (Chapter 3), where one must address a trade-off between availability and the maintenance of security. We also explored routing problems in the presence of strategic servers in call centers (Chapter 4), and in the presence of heterogeneous values in web server farms (Chapter 5). We conclude by discussing broad directions for future work in these areas.

We believe that the development of the CAP method (as presented in Chapter 2) is of interest to the research community primarily for two reasons. First, it is a new tool that can be employed by practitioners to obtain exact closed-form or exact numerical solutions for the limiting probability distribution of queue lengths in queueing systems that can be modeled by class $\mathbb{M}$ Markov chains. Second, it represents an additional way that one can view quasi-birth-death process Markov chains, that can hopefully further theoretical understanding of these Markov chains. In order to further advance CAP as a computational tool for practitioners, it would be of interested to initiate an extensive computational study that compares CAP without other state-of-the-art exact and inexact methods for solving Markov chains. This would require the careful fine-tuned implementations of the CAP method along with spectral methods, matrix analytic methods, ETAQA, RRR, and chain truncation approaches.

In developing and analyzing the CAP method, it became clear that there are fundamental connections between CAP and other exact methods such as exact matrix-geometric methods and RRR, and the chains that can be solved by these methods. In fact, there seems to be a hierarchy of QBDs in terms of the "difficulty" of solving these chains in exact closed-form, and chains exhibiting certain features (such as bi-directional phase transitions), seem to only have solutions that require one to solve higher-order polynomials. Future research could focus on classifying this hierarchy and formally identifying how certain features contribute to the "algebraic complexity" of the limiting probability distributions of such chains.

Our work in Chapter 3 on the malware cleanup problem represents early steps in using tools from the operations research community (i.e., queueing theory) to better understand tradeoffs that are of interest to the computer security community. Moreover, our work highlights the need to jointly consider performance concerns and security concerns, especially at attack timescales where the two concerns are comparable in importance. Future steps in this direction would likely involve an empirical component with as many parameters as possible based on measured observations, followed by an implementation and comparison of several of the cleanup policies described in Chapter 3 of this dissertation.

In Chapter 4 of this dissertation, we introduced a call center model in which servers behave strategically, and addressed the problem of routing in such a setting. Several natural questions arise from this work: How do our findings generalize to other contexts in which workers behave strategically? To what extent do our findings have real-world implications in settings where perhaps not all servers are identical in their talents or capabilities? How can a manager implement (or at least mimic) the rate-based routing policies examined in our paper when only empirical *service completion rates*, and not actual service rates can be accurately observed? What changes if servers can dynamically adjust their service rates in real time? Moving beyond these questions, there appears to be a growing need to model real-world queueing systems as being composed of strategic servers, especially with the rise of sharing economies where the "employees" of a service system are private contractors who choose when and how they work. We hope that the ideas presented in Chapter 4 will prove useful in the study of these newly emerging contexts.

Chapter 5 considered a problem of identifying how to immediately route jobs to one of several *a priori* identical servers employing the processor sharing scheduling discipline in a setting where job values were assumed to be heterogeneous. Job heterogeneity can exist across other dimensions, which would also create interesting consequence for how jobs should be routed in a "PS server farm" setting. One such dimension would be varying job *affinities* for different servers. What if, say due to data locality, some jobs would run faster on one server than another. How should this source of heterogeneity influence the optimal design of routing policies? An extreme variation on this setting would be if some jobs could only be served by a subset of the servers. How would this extreme variation affect routing policy design? Yet another direction for exploring the design of routing policies would be studying the impact of routing policies in contexts one can also control the scheduling policy at each server (e.g., if one could employ priority scheduling based on job values at the servers instead of being forced to use processor sharing). Does the choice of routing policy become inconsequential once one has control over scheduling? We believe that the interaction between routing and scheduling represents an intriguing and largely unexplored area in queueing theory.

Appendix A

Supplement to Chapter 2

A.1 An alternative interpretation of the Laplace transform

Let X be a nonnegative random variable, with well-defined Laplace transform $\psi(\cdot)$ (i.e., ψ is defined on all positive reals), cumulative distribution function, $F_X(\cdot)$, and probability density function, $f_X(\cdot)$; note that X may have nonzero probability mass at $+\infty$, in which case $\int_0^\infty f_X(t)\,dt < 1$ (where we interpret the integral as being evaluated on $\{t \in \mathbb{R} \colon 0 \le t < \infty\}$). Then for any constant $w > 0$, we have the following interpretation of ψ:

$$\begin{aligned}\psi(w) &= \int_0^\infty e^{-wt} f_X(t)\,dt \\ &= e^{-wt}F_X(t)\big|_0^\infty + \int_0^\infty F_X(t)\left(we^{-wt}\right)\,dt \\ &= \mathbb{P}\{X \le \zeta_w\},\end{aligned}$$

where $\zeta_w \sim \mathrm{Exponential}(w)$ is a random variable independent of X.

A.2 Complete proof of Theorem 2.3

Proof. We prove the theorem via strong induction on the phase, m. Specifically, for each phase m, we will show that $\pi_{(m,j)}$ takes the form $\pi_{(m,j)} = \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0}$ for all $j \ge j_0 + 1$, and show that $\{c_{m,k}\}_{0\le k\le m-1}$ satisfies

$$c_{m,k} = \begin{cases} \dfrac{r_k r_m}{\lambda_m(r_k - r_m)(1-\phi_m(\alpha_m) r_k)}\left(\displaystyle\sum_{i=k}^{m-1}\sum_{\Delta=-1}^{1} c_{i,k}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right) & \text{if } r_m, r_k > 0 \\ \dfrac{\displaystyle\sum_{i=k}^{m-1}\sum_{\Delta=-1}^{1} c_{i,k}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}}{\mu_m(1-r_k)+\alpha_m} & \text{if } r_k > r_m = 0 \\ 0 & \text{if } r_k = 0, \end{cases}$$

while $c_{m,m} = \pi_0 - \sum_{k=0}^{m-1} c_{m,k}$. Finally, after completing the inductive proof, we justify that the remaining linear equations in the proposed system are ordinary balance equations together with the normalization constraint.

Base case:

We begin our strong induction by verifying that the claim holds for the base case (i.e., for $m = 0$). By the ergodicity requirement on class $\mathbb{M}$ Markov chains, $\lambda_0 > 0$, leaving two sub-cases when $m = 0$: the case where $\mu_0 > 0$, and the case where $\mu_0 = 0$. In the first case, where $\mu_0 > 0$, Equation (2.5) yields

$$\mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right] = \Omega_0 r_0^{j-j_0-1}(1 - r_0\phi_0(\alpha_0)) = \frac{r_0^{j-j_0}}{\lambda_0}.$$

Now consider the other sub-case, where $\mu_0 = 0$, recalling that in this case, we have $r_0 = \lambda_0/(\lambda_0 + \alpha_0)$. We calculate $\mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right]$ for this case, by noting that transitions within states in P_0 cannot decrease the level, as follows: starting at state $(0, j_0 + 1)$, we either never visit state $(0, j)$ before leaving P_0, or we visit state $(0, j)$ *exactly once* before leaving P_0. The latter occurs with probability

$$\left(\frac{\lambda_0}{\lambda_0 + \alpha_0}\right)^{j-j_0-1} = r_0^{j-j_0-1},$$

in which case, we spend an average of $1/(\lambda_0 + \alpha_0) = r_0/\lambda_0$ units of time in state $(0, j)$. Hence, we find that

$$\mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right] = r_0^{j-j_0-1}\left(\frac{r_0}{\lambda_0}\right) = \frac{r_0^j}{\lambda_0},$$

which coincides with our finding for the case where $\mu_0 > 0$.

In both cases, applying Theorem 2.1 yields

$$\begin{aligned}\pi_{(0,j)} &= \pi_{(0,j_0)}\lambda_0\mathbb{E}_{(0,j_0+1)}\left[T_{(0,j)}^{P_0}\right] = \pi_{(0,j_0)}\lambda_0\left(\frac{r_0^{j-j_0}}{\lambda_0}\right) = \pi_{(0,j_0)}r_0^{j-j_0}\\ &= c_{0,0}r_0^{j-j_0},\end{aligned}$$

where $c_{0,0} = \pi_{(0,j_0)}$. Hence, $\pi_{(0,j)}$ takes the claimed form. Moreover, $c_{0,0}$ satisfies the claimed constraint as $c_{0,0} = \pi_{(0,j_0)} - \sum_{k=0}^{m-1} c_{m,k} = \pi_{(0,j_0)} - 0 = \pi_{(0,j_0)}$, because the sum is empty when $m = 0$. Note that when $m = 0$, $\{c_{m,k}\}_{0\le k<m\le M}$ is empty, and hence, there are no constraints on these values that require verification.

Inductive step:

Next, we proceed to the inductive step and assume the induction hypothesis holds for all phases $i \in \{0, 1, \ldots, m-1\}$. In particular, we assume that $\pi_{(i,j)} = \sum_{k=0}^{i} c_{i,k}r_k^{j-j_0}$ for all $i < m$. For convenience, we introduce the notation

$$\Upsilon_{m,j} \equiv \lambda_m\mathbb{E}_{(m,j_0+1)}\left[T_{(m,j)}^{P_m}\right] \quad \text{and} \quad \Psi_{m,k,j} \equiv \sum_{\ell=1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right].$$

Using this notation, we apply Theorem 2.1 and the induction hypothesis, which yields[1]

$$\begin{aligned}
\pi_{(m,j)} &= \pi_{(m,j_0)}\lambda_m \mathbb{E}_{(m,j_0+1)}\left[T^{P_m}_{(m,j)}\right] + \sum_{i=0}^{m-1}\sum_{\ell=1}^{\infty}\sum_{\Delta=-1}^{1}\pi_{(i,\ell-\Delta)}\alpha_i\langle m-i;\Delta\rangle \mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\
&= \pi_{(m,j_0)}\Upsilon_{m,j} + \sum_{i=0}^{m-1}\sum_{\ell=1}^{\infty}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle\left(\sum_{k=0}^{i} c_{i,k} r_k^{\ell-j_0-\Delta}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right]\right) \\
&= \pi_{(m,j_0)}\Upsilon_{m,j} + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\left(\sum_{\ell=1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right]\right) \\
&= \pi_{(m,j_0)}\Upsilon_{m,j} + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\Psi_{m,k,j}. \qquad \text{(A.1)}
\end{aligned}$$

We proceed to compute $\Upsilon_{m,j}$ and $\Psi_{m,k,j}$ separately in the following cases:

- **Case 1**: $\lambda_m, \mu_m > 0$
- **Case 2**: $\lambda_m > \mu_m = 0$
- **Case 3**: $\mu_m > \lambda_m = 0$
- **Case 4**: $\mu_m = \lambda_m = 0$

Computations for Case 1 ($\lambda_m, \mu_m > 0$)**:**

When $\lambda_m, \mu_m > 0$, Equation (2.5) yields $\Upsilon_{m,j} = \lambda_m \mathbb{E}_{(m,j_0+1)}\left[T^{P_m}_{(m,j)}\right] = r_m^{j-j_0}$. We also find that

$$\begin{aligned}
\Psi_{m,k,j} &= \sum_{\ell=1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\
&= \sum_{\ell=j_0+1}^{j} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] + \sum_{\ell=j+1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\
&= \Omega_m\left(\sum_{\ell=j_0+1}^{j} r_k^{\ell-j_0} r_m^{j-\ell}\left(1-(r_m\phi_m(\alpha_m))^{\ell-j_0}\right)\right. \\
&\qquad \left. + \sum_{\ell=j+1}^{\infty} r_k^{\ell-j_0}\phi_m(\alpha_m)^{\ell-j}\left(1-(r_m\phi_m(\alpha_m))^{j-j_0}\right)\right) \\
&= \frac{r_k r_m (r_k^{j-j_0} - r_m^{j-j_0})}{\lambda_m (r_k - r_m)(1-\phi_m(\alpha_m) r_k)},
\end{aligned}$$

[1] Note that $\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}$ is not well-defined when $r_k = 0$, as 0^{-1} and 0^0 are not well-defined. However, this is just a convenient formal manipulation which will remain true if we assign any real value to $\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}$ as $\Psi_{m,k,j} = 0$ in the $r_k = 0$ case, and the "contribution" to the sum by an index k such that $r_k = 0$ is also 0. One can verify that this is "harmless" by examining such k indices in isolation. Note further that we have also used the fact that $\pi_{(i,j_0)}$ also satisfies the claimed form for all $i < m$, which is true as $c_{i,i} = \pi_{(i,j_0)} - \sum_{k=0}^{i-1} c_{i,k}$ (from the inductive hypothesis) implies that $\pi_{(i,j_0)} = \sum_{k=0}^{i} c_{i,k} = \sum_{k=0}^{i} c_{i,k} r_k^0$, except that once again values of $r_k = 0$ yield undefined quantities of the form 0^0. Once again, this is a convenient formal manipulation that will not affect our results if we simply assign $0^0 = 1$ in this context.

where the last equality follows from well known geometric sum identities. Note that this expression is well-defined because $r_k \neq r_m$ by assumption and $r_m \phi_m(\alpha_m) \neq 1$.

Computations for Case 2 ($\lambda_m > \mu_m = 0$):

When $\lambda_m > \mu_m = 0$, we recall that $r_m = \lambda_m/(\lambda_m + \alpha_m)$ and compute $\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right]$ as follows: starting at state (m,ℓ), we either never visit state (m,j) before leaving P_m, or we visit state (m,j) *exactly once* before leaving P_m. If $\ell > j$, we never visit state (m,j) before leaving P_m (and so $\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] = 0$), but if $\ell \leq j$, we visit state (m,j) *exactly once* before leaving P_m with probability $r_m^{j-\ell}$, and this visit will last an average time of $1/(\lambda_m + \alpha_m) = r_m/\lambda_m$, yielding

$$\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] = r_m^{j-\ell}\left(\frac{r_m}{\lambda_m}\right) = \frac{r_m^{j-\ell+1}}{\lambda_m}.$$

In particular, $\Upsilon_{m,j} = \lambda_m \mathbb{E}_{(m,j_0+1)}\left[T^{P_m}_{(m,j)}\right] = r_m^{j-j_0}$, coinciding with the expression for $\Upsilon_{m,j}$ from Case 1, and furthermore, we have

$$\begin{aligned}
\Psi_{m,k,j} &= \sum_{\ell=j_0+1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\
&= \sum_{\ell=j_0+1}^{j} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] + \sum_{\ell=j+1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] \\
&= \sum_{\ell=j_0+1}^{j} \frac{r_k^{\ell-j_0} r_m^{j-\ell+1}}{\lambda_m} \\
&= \frac{r_k r_m (r_k^{j-j_0} - r_m^{j-j_0})}{\lambda_m (r_k - r_m)} = \frac{r_k r_m (r_k^{j-j_0} - r_m^{j-j_0})}{\lambda_m (r_k - r_m)(1 - \phi_m(\alpha_m) r_k)}.
\end{aligned}$$

which coincides with the expression for $\Psi_{m,k,j}$ that we found in Case 1. The last equality follows by noting that in this case we have $\phi_m(s) \equiv 0$, and hence $1 - \phi_m(\alpha_m) r_k = 1$.

Computations for Case 3 ($\mu_m > \lambda_m = 0$):

When $\mu_m > \lambda_m = 0$, we have $\Upsilon_{m,j} = \lambda_m \mathbb{E}_{(m,j_0+1)}\left[T^{P_m}_{(m,j)}\right] = 0$. Next, we compute $\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right]$ as follows: starting at state (m,ℓ), if $\ell < j$, we never visit j before leaving P_m, while if $\ell \geq j$ we will visit j exactly once with probability $\mu_m^{\ell-j}/(\mu_m + \alpha_m)^{\ell-j}$ and this visit will last an average duration of $1/(\mu_m + \alpha_m)$ units of time. Consequently, $\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] = 0$ in the former case and

$$\mathbb{E}_{(m,\ell)}\left[T^{P_m}_{(m,j)}\right] = \frac{\mu_m^{\ell-j}}{(\mu_m + \alpha_m)^{\ell-j+1}}$$

in the latter case. Finally, we have

$$\begin{aligned}\Psi_{m,k,j} &= \sum_{\ell=j_0+1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] \\ &= \sum_{\ell=j_0+1}^{j-1} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] + \sum_{\ell=j}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] \\ &= \sum_{\ell=j}^{\infty}\frac{r_k^{\ell-j_0}\mu_m^{\ell-j}}{(\mu_m+\alpha_m)^{\ell-j+1}} = \frac{r_k^{j-j_0}}{\mu_m(1-r_k)+\alpha_m}.\end{aligned}$$

Computations for Case 4 ($\mu_m = \lambda_m = 0$):

When $\mu_m = \lambda_m = 0$, we again have $\Upsilon_{m,j} = \lambda_m \mathbb{E}_{(m,j_0+1)}\left[T_{(m,j)}^{P_m}\right] = 0$, as in Case 3. Next, we compute $\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right]$ as follows: in this case any visit to P_m will consist entirely of one visit to the initial state in P_m, as there are no transitions to other states in the same phase. Hence, $\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = \alpha_m$ if $\ell = j$, and $\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = 0$ otherwise. Consequently,

$$\begin{aligned}\Psi_{m,k,j} &= \sum_{\ell=1}^{\infty} r_k^{\ell-j_0}\mathbb{E}_{(m,\ell)}\left[T_{(m,j)}^{P_m}\right] = r_k^{j-j_0}\mathbb{E}_{(m,j)}\left[T_{(m,j)}^{P_m}\right] = \frac{r_k^{j-j_0}}{\alpha_m} \\ &= \frac{r_k^{j-j_0}}{\mu_m(1-r_k)+\alpha_m},\end{aligned}$$

which coincides with the expression for $\Psi_{m,k,j}$ that we found in Case 3. The last equality follows by noting that $\mu_m = 0$, and hence $\mu_m(1-r_k) = 0$.

Completing the inductive step:

We now proceed to substitute the results of our computations into Equation (A.1). Since $\Upsilon_{m,j}$ can be given by the same expression for both Case 1 and 2, and the same holds for $\Psi_{m,k,j}$, we consider these two cases together, and note that they jointly make up the case where $r_m > 0$. For $j \geq j_0 + 1$,

$$\begin{aligned}\pi_{(m,j)} &= \pi_{(m,j_0)}\Upsilon_{m,j} + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\Psi_{m,k,j} \\ &= \pi_{(m,j_0)}r_m^{j-j_0} + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\left(\frac{r_k r_m(r_k^{j-j_0}-r_m^{j-j_0})}{\lambda_m(r_k-r_m)(1-\phi_m(\alpha_m)r_k)}\right) \\ &= \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0},\end{aligned}$$

where we have collected terms with

$$c_{m,k} = \frac{r_k r_m\left(\sum_{i=k}^{m-1}\sum_{\Delta=-1}^{1} c_{i,k}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)}{\lambda_m(r_k-r_m)(1-\phi_m(\alpha_m)r_k)} \qquad (0 \leq k < m \leq M\colon r_m, r_k > 0)$$

and $c_{m,k} = 0$ when $r_m > r_k = 0$ and $c_{m,m} = \pi_{(m,j_0)} - \sum_{k=0}^{m-1} c_{m,k}$, as claimed.

The expressions for $\Upsilon_{m,j}$ and $\Psi_{m,k,j}$ also coincide across Cases 3 and 4 (although they are distinct from their Case 1 and 2 counterparts), so we also consider these two cases together, noting that they jointly make up the case where $\lambda_m = r_m = 0$:

$$\begin{aligned}
\pi_{(m,j)} &= \pi_{(m,j_0)}\Upsilon_{m,j} + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\Psi_{m,k,j} \\
&= 0 + \sum_{k=0}^{m-1}\sum_{i=k}^{m-1}\left(c_{i,k}\sum_{\Delta=-1}^{1}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}\right)\left(\frac{r_k^{j-j_0}}{\mu_m(1-r_k)+\alpha_m}\right) \\
&= \sum_{k=0}^{m} c_{m,k} r_k^{j-j_0}
\end{aligned}$$

where we have collected terms with

$$c_{m,k} = \frac{\displaystyle\sum_{i=k}^{m-1}\sum_{\Delta=-1}^{1} c_{i,k}\alpha_i\langle m-i;\Delta\rangle r_k^{\Delta}}{\mu_m(1-r_k)+\alpha_m} \qquad (0 \le k < m \le M\colon r_m, r_k > 0)$$

and $c_{m,k} = 0$ when $r_m = r_k = 0$. Observe that since $r_m = 0$, it appears that we can allow $c_{m,m}$ to take any real value, so in order to satisfy the induction hypothesis, we set $c_{m,m} = \pi_{(m,j_0)} - \sum_{k=0}^{m-1} c_{m,k}$ in the $r_m = 0$ case as well. Also note that we have set $c_{m,k} = 0$ when $r_k = 0$ in both the $r_m > 0$ and $r_m = 0$ cases. This completes the inductive step and the proof by induction.

The balance equations and normalization constraint:

The equations with $\pi_{(m,j_0)}$ and π_x in their left-hand sides in our proposed system are ordinary balance equations (that have been normalized so that there are no coefficients on the left-hand side).

It remains to verify that the final equation, which is the normalization constraint:

$$\begin{aligned}
1 &= \sum_{x\in\mathcal{N}}\pi_x + \sum_{m=0}^{M}\pi_{(m,j_0)} + \sum_{m=0}^{M}\sum_{j=j_0+1}^{\infty}\pi_{(m,j)} \\
&= \sum_{x\in\mathcal{N}}\pi_x + \sum_{m=0}^{M}\sum_{k=0}^{M} c_{m,k} + \sum_{m=0}^{M}\sum_{k=0}^{m-1}\sum_{j=j_0+1}^{\infty} c_{m,k} r_k^{j-j_0} \\
&= \sum_{x\in\mathcal{N}}\pi_x + \sum_{m=0}^{M}\sum_{k=0}^{m}\frac{c_{m,k}r_k}{1-r_k}.
\end{aligned}$$

□

A.3 Negative binomial lemmas

These lemmas are used to derive our main results, and are likely known, but to make the paper self-contained we both state and prove them.

Lemma A.1. *For each $\beta \in (0,1)$, we have*

$$\sum_{\ell=j_0}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell-(j_0-1)} = \frac{\beta}{(1-\beta)^{n+1}}.$$

Proof. Having a negative binomial distribution with parameters $n+1$ and $(1-\beta)$ in mind, we observe that

$$\begin{aligned}
\sum_{\ell=j_0}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell-(j_0-1)} &= \beta \sum_{\ell=j_0}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell-j_0} \\
&= \frac{\beta}{(1-\beta)^{n+1}} \sum_{\ell=j_0}^{\infty} \binom{\ell - j_0 + n}{(n+1)-1} \beta^{\ell-j_0}(1-\beta)^{n+1} \\
&= \frac{\beta}{(1-\beta)^{n+1}} \sum_{k=0}^{\infty} \binom{(k+n+1)-1}{(n+1)-1} \beta^{k}(1-\beta)^{n+1} \\
&= \frac{\beta}{(1-\beta)^{n+1}} \sum_{\ell=n+1}^{\infty} \binom{\ell-1}{(n+1)-1} \beta^{\ell-(n+1)}(1-\beta)^{n+1} \\
&= \frac{\beta}{(1-\beta)^{n+1}}.
\end{aligned}$$

□

The next lemma shows how to compute a truncated version of the above series.

Lemma A.2. *For $\beta \neq 1$, we have*

$$\begin{aligned}
\sum_{\ell=j_0}^{j-1} \binom{\ell - j_0 + n}{n} \beta^{\ell-(j_0-1)} &= \frac{\beta - \beta^{j-(j_0-1)}}{(1-\beta)^{n+1}} \\
&\quad - \sum_{k=1}^{n} \left[\binom{j-j_0+k}{k} - \binom{j-j_0+k-1}{k-1} \right] \frac{\beta^{j-(j_0-1)}}{(1-\beta)^{n+1-k}}.
\end{aligned}$$

Proof. Starting with the left-hand-side, we have

$$\begin{aligned}
\sum_{\ell=j_0}^{j-1}\binom{\ell-j_0+n}{n}\beta^{\ell-(j_0-1)} &= \sum_{\ell=j_0}^{j-1}\sum_{x=j_0}^{\ell}\binom{x-j_0+n-1}{n-1}\beta^{\ell-(j_0-1)}\\
&= \sum_{x=j_0}^{j-1}\sum_{\ell=x}^{j-1}\binom{x-j_0+n-1}{n-1}\beta^{\ell-(j_0-1)}\\
&= \sum_{x=j_0}^{j-1}\binom{x-j_0+n-1}{n-1}\beta^{x-(j_0-1)}\sum_{\ell=x}^{j-1}\beta^{\ell-x}\\
&= \frac{1}{(1-\beta)}\sum_{x=j_0}^{j-1}\binom{x-j_0+n-1}{n-1}\beta^{x-(j_0-1)}(1-\beta^{j-x})\\
&= \frac{1}{(1-\beta)}\sum_{x=j_0}^{j-1}\binom{x-j_0+n-1}{n-1}\beta^{x-(j_0-1)}\\
&\quad -\frac{1}{1-\beta}\binom{j-j_0+n-1}{n}\beta^{j-(j_0-1)}\\
&= \frac{1}{(1-\beta)}\sum_{x=j_0}^{j-1}\binom{x-j_0+n-1}{n-1}\beta^{x-(j_0-1)}\\
&\quad -\frac{1}{1-\beta}\left[\binom{j-j_0+n}{n}-\binom{j-j_0+n-1}{n-1}\right]\beta^{j-(j_0-1)}.
\end{aligned}$$

Setting

$$\begin{aligned}
a_n &= \sum_{\ell=j_0}^{j-1}\binom{\ell-j_0+n}{n}\beta^{\ell-(j_0-1)},\\
b_n &= \left[\binom{j-j_0+n}{n}-\binom{j-j_0+n-1}{n-1}\right]\beta^{j-(j_0-1)}
\end{aligned}$$

we see that for each $n \in \{1, 2, 3, \ldots\}$ we have

$$a_n = \frac{a_{n-1}}{1-\beta} - \frac{b_n}{1-\beta}$$

where

$$a_0 = \frac{\beta-\beta^{j-(j_0-1)}}{1-\beta}.$$

The solution to this recursion is given by

$$a_n = \frac{a_0}{(1-\beta)^n} - \sum_{k=1}^{n}\frac{b_k}{(1-\beta)^{n+1-k}}$$

or, equivalently,

$$a_n = \frac{1-\beta^{j-(j_0-1)}}{(1-\beta)^{n+1}} - \sum_{k=1}^{n}\left[\binom{j-j_0+n}{n}-\binom{j-j_0+n-1}{n-1}\right]\frac{\beta^{j-(j_0-1)}}{(1-\beta)^{n+1-k}},$$

which completes our derivation. □

The next lemma can be viewed as a generalization of Lemma A.1.

Lemma A.3. *For* $\beta \in (0,1)$,

$$\sum_{\ell=j}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell - j} = \frac{1}{(1-\beta)^{n+1}} + \sum_{k=1}^{n} \left[\binom{j - j_0 + k}{k} - \binom{j - j_0 + k - 1}{k-1} \right] \frac{1}{(1-\beta)^{n+1-k}}.$$

Proof. The key to deriving this series is to use both Lemmas A.1 and A.2. Here

$$\begin{aligned}
\sum_{\ell=j}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell - j} &= \beta^{j_0 - j} \sum_{\ell - j_0}^{\infty} \binom{\ell - j_0 + n}{n} \beta^{\ell - j_0} \\
&\quad - \beta^{(j_0 - 1) - j} \sum_{\ell = j_0}^{j-1} \binom{\ell - j_0 + n}{n} \beta^{\ell - (j_0 - 1)} \\
&= \frac{\beta^{j_0 - j}}{(1-\beta)^{n+1}} - \beta^{(j_0-1)-j} \left[\frac{\beta - \beta^{j - (j_0 - 1)}}{(1-\beta)^{n+1}} \right] \\
&\quad + \sum_{k=1}^{n} \left[\binom{j - j_0 + k}{k} - \binom{j - j_0 + k - 1}{k-1} \right] \frac{1}{(1-\beta)^{n+1-k}} \\
&= \frac{1}{(1-\beta)^{n+1}} + \sum_{k=1}^{n} \left[\binom{j - j_0 + k}{k} - \binom{j - j_0 + k - 1}{k-1} \right] \frac{1}{(1-\beta)^{n+1-k}}.
\end{aligned}$$

thus proving the claim. □

Appendix B

Supplement to Chapter 5

B.1 Proof of Proposition 5.4

Proposition 5.4. *VITA is the optimal (i.e., $\mathbb{E}[VT]$-minimizing) static policy for any two-server system with identical service rates. Furthermore, VITA unbalances the load, whereas all load balancing static policies achieve the same performance as RND.*

Proof. First observe that any static policy can be described by a measurable function $\varphi(\cdot)$, such that $\varphi(v)$ is the probability that a job with value v is routed to server 1, and consequently, $1-\varphi(v)$ gives the probability of routing a job to server 2. For example, RND is given by $\varphi(v) = 1/2$ for all v, while VITA is given by $\varphi(v) = 1$ for all values v above some threshold and $\varphi(v) = 0$ for all values v below that threshold.

Describing static policies by such functions, the $\mathbb{E}[VT]$-minimizing policy is given by $\varphi(\cdot)$ in the solution of the following minimization problem:

$$\begin{aligned}
\min_{p_i;m_i;\varphi(\cdot)} \quad & \frac{m_1}{\mu - p_1\lambda} + \frac{m_2}{\mu - p_2\lambda} \\
\text{s.t.} \quad & \varphi\colon \mathbb{R}^+ \to [0,1] \text{ is measurable} \\
& m_1 = \int_0^\infty v\varphi(v)\,dF(v), \quad m_2 = \int_0^\infty v(1-\varphi(v))\,dF(v) \\
& p_1 = \int_0^\infty \varphi(v)\,dF(v), \quad p_2 = \int_0^\infty (1-\varphi(v))\,dF(v) \\
& p_1\lambda < \mu_1, \quad p_2\lambda < \mu_2 \\
& p_1 \geq p_2
\end{aligned}$$

where F is the c.d.f. of the value distribution. Here, we can interpret p_i as the fraction of jobs sent to server i and m_i as the average value of the jobs sent to server i weighted by the fraction of jobs sent to server i (i.e., the average value of the jobs sent to server i multiplied by p_i). We note that $p_1 + p_2 = 1$ and $m_1 + m_2 = \mathbb{E}[V]$. Moreover, we note that although the constraint $p_1 \geq p_2$ need not hold for all feasible static policies, this restriction is without loss of generality,[1]

[1]If $p_1 < p_2$, one can interchange p_1 and p_2, interchange m_1 and m_2, and replace φ with $1-\varphi$ to obtain the same objective value with $p_1 > p_2$, as required)

and simplifies the feasible region.

Now fix p_1 and p_2 at their optimal values, which simplifies the optimization problem: we must now minimize a weighted sum of m_1 and m_2 subject to $m_1 + m_2 = \mathbb{E}[V]$ and bounds on m_1 and m_2. Since $p_1 \geq p_2$, we have $1/(\mu - p_1\lambda) \geq 1/(\mu - p_2\lambda)$, and hence, the coefficient of m_1 in this weighted sum is greater than that of m_2. Consequently, since $m_1, m_2 \geq 0$, the objective function is minimized by making m_1 as small as possible, subject to the lower bound on m_1 imposed by the fixed value of p_1. This means that we want to send as many higher value jobs to server 2 as possible, so we must have an optimal φ function given by

$$\varphi(v) = \begin{cases} 1 & v < \xi \\ \varphi(\xi) & v = \xi \\ 0 & v > \xi \end{cases},$$

where ξ and $\varphi(\xi)$ satisfy

$$\int_0^{\xi} dF(v) + \varphi(\xi)\mathrm{P}\{V = \xi\} = p_1,$$

with the integral is evaluated on an open interval. Since p_1 was chosen optimally, by assumption, we can conclude that ξ is the *optimal* dispatching threshold. Therefore, the optimal φ function describes the VITA policy exactly; so we may conclude that VITA is the optimal static policy.

Since the load at server i is $p_i\lambda/\mu$, in order to prove that VITA unbalances the load, we need only prove that $p_1 > p_2$ (i.e., $p_1 > 1/2$) in the optimal solution. Assume, by way of contradiction, that $p_1 = p_2$. Under VITA (and hence, in the optimal solution), m_1 (m_2) corresponds to the portion of $\mathbb{E}[V]$ made up of jobs lying below (above) the median of V, and hence $m_1 < m_2$. Now consider increasing p_1 (and consequently, decreasing p_2) by some small $\delta > 0$, while preserving a VITA-like threshold policy (i.e., $\varphi(v)$ is monotonically decreasing and $\{0, 1\}$-valued for all v, except at perhaps one threshold point).

Consequently, m_1 will increase and m_2 will decrease by some value $\epsilon(\delta)$, since the $\delta/2$ least valuable fraction of jobs that were being sent to server 2 will be rerouted to server 1. We argue that for sufficiently small $\delta > 0$, we must have $\epsilon(\delta) \leq c\delta$ for some constant c. For example, if $\delta < 1/4$, we must have $\epsilon(\delta) \leq c\delta$, where c is the upper quartile of the value distribution, as all rerouted values will have value at most c.

Finally, let Δ be the change in the objective function due to increasing p_1 from $1/2$ to $1/2+\delta$. For $\delta > 0$ small enough to ensure that $\mu > \lambda(1/2 + \delta)$, we must have

$$\frac{1}{\mu - \lambda(1/2 + \delta)} - \frac{1}{\mu - \lambda(1/2 - \delta)} \geq 0,$$

which allows us to establish that

$$\begin{aligned}
\Delta &\equiv \frac{m_1+\epsilon(\delta)}{\mu-\lambda(1/2+\delta)} + \frac{m_2-\epsilon(\delta)}{\mu-\lambda(1/2-\delta)} - \frac{m_1+m_2}{\mu-\lambda/2} \\
&\leq \frac{m_1+c\delta}{\mu-\lambda(1/2+\delta)} + \frac{m_2-c\delta}{\mu-\lambda(1/2-\delta)} - \frac{m_1+m_2}{\mu-\lambda/2}, \\
\lim_{\delta\to 0}\left(\frac{\Delta}{\delta}\right) &\leq \lim_{\delta\to 0}\left(\frac{1}{\delta}\right)\left(\frac{m_1+c\delta}{\mu-\lambda(1/2+\delta)} + \frac{m_2-c\delta}{\mu-\lambda(1/2-\delta)} - \frac{m_1+m_2}{\mu-\lambda/2}\right) \\
&= \frac{4\lambda(m_1-m_2)}{(\mu-\lambda/2)^2} < 0,
\end{aligned}$$

as $m_1 < m_2$. Hence, the objective function can be decreased by a slight increase in p_1, which provides the desired contradiction: a load balancing policy is suboptimal, and since we have shown VITA to be optimal, it unbalances load.

Finally, any load balancing static policy, including RND, that dispatches according to some function φ obtains

$$\mathbb{E}[VT] = \frac{\int_0^\infty v\varphi(v)\,dF(v) + \int_0^\infty v(1-\varphi(v))\,dF(v)}{\mu-\lambda/2} = \frac{\int_0^\infty v\,dF(v)}{\mu-\lambda/2} = \frac{\mathbb{E}[V]}{\mu-\lambda/2},$$

which does not depend on φ, completing the proof. □

B.2 Proof of Lemma 5.10

[2] **Lemma 5.10.** *Let $V \sim \mathrm{SBD}(p)$ in a system with two identical servers. As $p \to 1$:*

- The limiting distribution of the number of low value jobs, N_ℓ, under LAVA converges weakly to the limiting distribution of the total number of jobs, N, under JSQ, and $\mathbb{E}[N_\ell]^{\text{LAVA}} \to \mathbb{E}[N]^{\text{JSQ}}$.
- The limiting distribution of the number of high-value jobs, N_h, under LAVA converges weakly to the zero distribution, and $\mathbb{E}[N_h]^{\text{LAVA}} \to 0$.
- The limiting distribution of the length of the shorter queue (i.e., the instantaneous minimum length of the two queues), N_{short}, under LAVA converges to the limiting distribution of N_{short} under JSQ, and $\mathbb{E}[N_{\text{short}}]^{\text{LAVA}} \to \mathbb{E}[N_{\text{short}}]^{\text{JSQ}}$.

In proving Lemma 1, we make use of Sublemma 1 below. This result is a special case of a result due to Karr [88].

Sublemma 1.*Let $M_1, M_2, \ldots, M_n, \ldots$ be a sequence of ergodic Markov chains defined on the same countable space A, each with its own transition rate matrix Q_n and unique nowhere-zero limiting distribution π_n, uniquely solving $\pi_n Q_n = \mathbf{0}$ and $\pi_n \cdot \mathbf{1} = 1$. Furthermore, let M be a (possibly non-ergodic) Markov chain with transition rate matrix Q and unique limiting distribution π, uniquely solving $\pi Q = \mathbf{0}$ and $\pi \cdot \mathbf{1} = 1$ such that $Q_n \to Q$ uniformly. Then $\pi_n \to \pi$ in the sense of weak convergence.*

[2] I would like to thank Gautam Iyer for his assistance with some technical details that have been used in this proof.

Proof of Sublemma 1. If π_n uniquely solves the linear system $\pi_n(\mathbf{1}, Q_n) = (1, \mathbf{0})$ while π uniquely solves the linear system $\pi(\mathbf{1}, Q) = (1, \mathbf{0})$, and $Q_n \to Q$ uniformly (and thus, $(\mathbf{1}, Q_n) \to (\mathbf{1}, Q)$ uniformly), we must have $\pi_n \to \pi$ uniformly. It follows that $\pi_n \to \pi$ in the sense of weak convergence. □

Note that since M may not be ergodic, π may be zero somewhere. This occurs when all transitions (in Q_n) into some nonempty proper subset of states $B \subsetneq A$ converge to 0. Note that since π is unique, it is guaranteed that there are paths made up of transitions with nonzero rates (in Q) from all states in B to the non-transient portion of the state space of M.

Proof of Lemma 5.10. The three individual results follow in a straightforward way after the proper application of Sublemma 1 to the Markov chains of interest.

For $p \in (1/2, 1)$, let the Markov chain $M^{(p)}$ (with transition rate matrix $Q^{(p)}$) denote the underlying Markov chain of the two-server system under LAVA when $V \sim \mathrm{SBD}(p)$, with state space $A \equiv \{(\ell_1, h_1, \ell_2, h_2)\colon \ell_1, h_1, \ell_2, h_2 \in \mathbb{N}_{\geq 0}\}$, where ℓ_i and h_i track the number of low-value and high-value jobs at server i, respectively. By Theorem 5.6, the LAVA policy is stable (so long as the value distribution has a nonzero lower bound and a finite upper bound, as is the case here). Moreover, all states communicate with one another, so these Markov chains are ergodic. Consequently, each Markov chain, $M^{(p)}$, has a unique nowhere-zero limiting distribution, $\pi^{(p)}$, which uniquely solves $\pi^{(p)}Q^{(p)} = \mathbf{0}$ and $\pi^{(p)} \cdot \mathbf{1} = 1$.

Next, define the Markov chain $M^{(1)}$ over the state space A by letting its transition rate matrix, $Q^{(1)}$, be given by replacing every instance of p in $Q^{(p)}$ with 1.[3] Observe that the states $(\ell_1, h_1, \ell_2, h_2)$ with $h_1 = h_2 = 0$ all communicate with one another, but states with $h_1 > 0$ or $h_2 > 0$ are inaccessible from the aforementioned states (transition rates entering these states from the other states in the chain are zero), making this a non-ergodic Markov chain. The non-transient states of $M^{(1)}$ are exactly those states where $h_1 = h_2 = 0$. Since all transition rates of $M^{(p)}$ are either constant in p or equal to λp, $\lambda p/2$, $\lambda(1-p)$, or $\lambda(1-p)/2$ (in accordance with the LAVA policy), we see that as $p \to 1$, the transition rate matrices $Q^{(p)} \to Q^{(1)}$ uniformly.

Now let M^{JSQ} (with transition rate matrix Q^{JSQ}) denote the underlying Markov chain of the two-server system under JSQ, with state space $\{(j_1, j_2)\colon j_1, j_2 \in \mathbb{N}_{\geq 0}\}$, where j_i tracks the number of jobs at server i. Note that the composition of jobs is unimportant to the evolution of the system under JSQ, and hence, this Markov chain does not track this composition, and nor does not depend on p. Observe that M^{JSQ} is exactly the same Markov chain as $M^{(1)}$ with the transient (inaccessible) portion removed: we identify states (j_1, j_2) with states $(j_1, 0, j_2, 0)$, and remove the other states. Now observe that by Theorem 5.6, we know that the JSQ system is stable, and moreover all states communicate with one another, so M^{JSQ} has a unique nowhere-zero limiting distribution π^{JSQ}, which uniquely solves $\pi^{\mathrm{JSQ}}Q^{\mathrm{JSQ}} = \mathbf{0}$ and $\pi^{\mathrm{JSQ}} \cdot \mathbf{1} = 1$. It follows that $M^{(1)}$ also has a unique limiting distribution, $\pi^{(1)}$, uniquely solving $\pi^{(1)}Q^{(1)} = \mathbf{0}$ and $\pi^{(1)} \cdot \mathbf{1} = 0$, where $\pi^{(1)}$ coincides with π^{JSQ} on the states $(\ell_1, h_1, \ell_2, h_2)$ where $h_1 = h_2 = 0$, and is equal to zero on all other states.

Now consider an arbitrary increasing sequence $p_1, p_2, \ldots p_n, \ldots \in (0, 1)$, such that $p_n \to 1$. Define a sequence of Markov chains $M_1, M_2, \ldots, M_n, \ldots$ (with transition rate matrices

[3]We cannot associate $M^{(1)}$ directly with the two-server system under LAVA when $V \sim \mathrm{SBD}(1)$, as $\mathrm{SBD}(1)$ is not a well-defined distribution.

$Q_1, Q_2, \ldots, Q_n, \ldots$), where $M_n = M^{(p_n)}$, and let $M = M^{(1)}$, $Q = Q^{(1)}$ and $\pi = \pi^{(1)}$. We may now apply Sublemma 1, obtaining $\pi_n \to \pi$ in the sense of weak convergence, from which it follows that as $p \to 1$, $\pi^{(p)} \to \pi^{\text{JSQ}}$ on the states $(\ell_1, h_1, \ell_2, h_2)$, where $h_1 = h_2 = 0$ (using the identification previously explained), and $\pi^{(p)}$ converges to the zero distribution, elsewhere. It follows from this convergence that as $p \to 1$:

- The limiting distributions of $N_\ell \equiv \ell_1 + \ell_2$ under LAVA converges to the limiting distribution of $N \equiv j_1 + j_2$ under JSQ.
- The limiting distribution of $N_h \equiv h_1 + h_2$ converges to the zero distribution.
- The limiting distribution of $N_{\text{short}} \equiv \min\{\ell_1 + h_1, \ell_2 + h_2\}$ converges to limiting distribution of $N_{\text{short}} \equiv \min\{j_1, j_2\}$ under JSQ.

To complete the proof, we must show that the expectations also converge. This does not follow immediately from the convergence of the limiting distributions, as N_ℓ, N_h, and N_{short} are *unbounded* functions on the state space $A \equiv \{(\ell_1, h_1, \ell_2, h_2) \colon \ell_1, h_1, \ell_2, h_2 \in \mathbb{N}_{\geq 0}\}$. Now observe that for any $p \in (1/2, 1)$, at any given time we have $N_\ell, N_h, N_{\text{short}} \leq N \equiv \ell_1 + h_1 + \ell_2 + h_2$ under LAVA. Using the convergence of limiting probabilities established above, together with the fact that N bounds the random variables of interest, we may apply the de la Vallée-Poussin Lemma (cf. [110]) to obtain a sufficient condition for $\mathbb{E}[N_\ell]^{\text{LAVA}} \to \mathbb{E}[N]^{\text{JSQ}}$, $\mathbb{E}[N_h]^{\text{LAVA}} \to 0$, and $\mathbb{E}[N_{\text{short}}]^{\text{LAVA}} \to \mathbb{E}[N_{\text{short}}]^{\text{JSQ}}$. In order for the expectations to converge as claimed, it is sufficient to show that for any fixed $\rho \in (0, 1)$, there exists some $\delta > 0$ and $K < \infty$, such that for all $p \in (1 - \delta, 1)$, we have $\mathbb{E}[N^2]^{\text{LAVA}} < K$. That is, we want to show that the second moment of the number of jobs in the LAVA system is bounded for all p in a neighborhood of 1.

In proving that this sufficient condition holds, it will be useful to introduce a policy, which we call LAVA′, defined form $V \sim \text{SBD}(p)$, as follows:

- if there are no high-value jobs in the system, low-value jobs are dispatched randomly (i.e., according to RND);
- if there are one or more high-value jobs in the system, low-value jobs are routed to server 1;
- high-value jobs are always routed to server 2.

We argue that N under LAVA with load $\rho < 1$, is stochastically no greater than N under LAVA′ with load $\rho' \equiv 2\rho/(1+\rho) < 1$. In the absence of a high-value job, LAVA dispatches according to JSQ, while LAVA′ dispatches according to RND. Moreover, observe that each queue in a system under RND with load ρ' has as many jobs (in expectation) as an entire two-server system under RND with load ρ, and hence, more jobs than either queue of a system under JSQ with load ρ. Therefore, ignoring the effect of high-value jobs, each queue of the LAVA′ system with load ρ' is stochastically longer than either queue of the LAVA system under load ρ.

Meanwhile, LAVA′ sends high-value jobs to the same server, and allows them to always create a "stopper" effect at that server (i.e., all low-value jobs are sent to the other server). This "stoppering" behavior causes the queue at the other server to be much longer than it otherwise would be. Although a similar phenomenon occurs under LAVA, this effect is more pronounced under LAVA$'$, as the latter policy sends *all* high-value jobs to server 2.[4] Hence, N under LAVA$'$ with load ρ' (with associated arrival rate $\lambda' \equiv 2\mu \cdot \rho'$ and service rate μ) stochastically dominates N under LAVA with load ρ.

We proceed to show that $\mathbb{E}[N^2]^{\text{LAVA}'}$ is finite for any given $\rho' < 1$ and p sufficiently close to 1. Clearly, the contribution to $\mathbb{E}[N^2]^{\text{LAVA}'}$ from server 2 is bounded, because (for all p sufficiently close to 1) server 2 receives jobs with mean interarrival that are always less than $1/\mu$.

Turning our attention to server 1, we may view the arrival process to server 1 as alternating between a Poisson process with rate $\lambda' p/2$ (when there are no high-value jobs at server 2), and a Poisson process with rate $\lambda' p$ (when there are one or more high-value job at server 2). The duration during which server 1 receives jobs at the higher arrival rate of $\lambda' p$ corresponds to a "high-value busy period" started by the arrival of the first high-value job to server 2, and concluded when server 2 is no longer serving any high-value jobs. We write B_h to denote the length of this busy period. We upper bound N by assuming that all *additional* arrivals which would arrive during this high-value busy period arrive at the same time as a "batch arrival."[5] That is, we can view server 1 as receiving jobs according to a Poisson process with a fixed rate, except whenever server 2 receives a high-value job (when it previously had none), server 1 will receive many jobs at once. The number of jobs, A_{B_h}, making up this batch of "many jobs," will be distributed like the number of *additional* arrivals server 1 would have received in B_h time. Hence, we can upper bound the queue length at server 1 with the number of jobs, N, in an M^Y/M/1 system (cf. [62]),[6] with arrival rate $\lambda' p/2 + \lambda'(1-p) = \lambda'(1-p/2)$ and "batch size," Y, distributed as follows:

$$Y \sim \begin{cases} 1 & \text{w.p. } \dfrac{p}{2-p} \\ \\ A_{B_h} & \text{w.p. } \dfrac{2-2p}{2-p}. \end{cases}$$

Here, we are again overestimating N by assuming that *each* high-value arrival sent to server 2 (rather than only the first high-value arrival to start each high-value busy period) causes A_{B_h} additional jobs to be sent to server 1.

Next, we show that for all p sufficiently close to 1, B_h has all finite moments, and hence A_{B_h} and Y have finite moments, and finally, N has finite moments. We may overestimate B_h with B_h^*, the duration of a high-value busy period under the assumption server 2 serves low-value jobs ahead of high-value jobs, rather than employing Processor-Sharing. Note that this alternative scheduling policy can only lengthen the busy period, so B_h^* is indeed stochastically greater than B_h. Under this alternative scheduling policy, when a high-value job arrives to server 2, it starts a busy period of length B_h^*, with Laplace transform

$$\widetilde{B_h^*}(s) = \widetilde{W}\left(s + \lambda'(1-p)\left(1 - \widetilde{B}_h(s)\right)\right),$$

[4]The higher load under LAVA$'$ (i.e., $\rho' > \rho$) also contributes to longer queue lengths under LAVA$'$.

[5]We note that receiving these additional arrivals at once may (rarely) allow for server 1 to work on jobs before they would arrive in the original system without batching. Even with this possibility, sending future arrivals earlier can only cause N to (stochastically) increase, rather than decrease.

[6]We use the notation M^Y/M/1 to refer to the system more commonly denoted by M^{X}/M/1 in order to prevent ambiguity in the use of the random variable X in this paper, as X has previously denoted service requirements.

where W is the random variable giving the amount of work at server 2 seen by the first high-value arrival [70]. We know that for all p sufficiently close to 1, W has all finite moments and its Laplace transform, $\widetilde{W}(s)$, exists. Hence, $\widetilde{B_h^*}(s)$ is well-defined, and B_h^* has all finite moments. Since B_h^* stochastically dominates B_h, it follows that B_h must also have all finite moments, and its Laplace transform, $\widetilde{B}_h(s)$, exists. Consequently, the z-transforms of both A_{B_h} and Y exist (establishing that both have all finite moments) and are given by

$$\widehat{A_{B_h}}(z) = \widetilde{B}_h(\lambda' p/2 \cdot (1-z)),$$
$$\widehat{Y}(z) = \frac{pz}{2-p} + \frac{2-2p}{2-p} \cdot \widehat{A_{B_h}}(z).$$

Moreover, for all p sufficiently close to 1, the aggregate arrival rate to server 1, $\lambda'(1-p/2)\cdot\mathbb{E}[Y]$, is less than the departure rate, μ. This fact, combined with the existence of $\widehat{Y}(z)$, guarantees that the number of jobs, N, in the M^Y/M/1 system of interest has a well-defined z-transform, and hence, N has all finite moments (see [62] for details). Therefore, the number of jobs at server 1 under LAVA′ has all finite moments for all p sufficiently close to 1, and hence, $\mathbb{E}[N^2]^{\mathrm{LAVA}'}$ is finite.

Finally, for any $\rho' < 1$, $\mathbb{E}[N^2]^{\mathrm{LAVA}'}$ must be bounded for all p sufficiently close to 1, because an increase in p leads to shorter high-value busy period durations, B_h, smaller batch sizes, Y, and less frequent batch arrivals of size A_{B_h}, in exchange for a vanishingly higher "low-traffic" arrival rate to server 1. Hence, $\mathbb{E}[N^2]^{\mathrm{LAVA}'}$ is eventually decreasing in p as $p \to 1$. It follows that for each ρ', there exists some $K < \infty$ such that $\mathbb{E}[N^2]^{\mathrm{LAVA}'} < K$ for all p in a neighborhood of 1. Consequently, $\mathbb{E}[N^2]^{\mathrm{LAVA}}$ is bounded for any $\rho < 1$, which establishes that the expectations of interest converge as claimed. □

B.3 Proof of Lemma 5.14.

Lemma 5.14. *Let* $V \sim \mathrm{SBD}(p)$ *in a system with two identical servers. As* $p \to 1$*, the* G-VITA *policy with parameter* g *under load* ρ *is stable whenever*

$$g > -\frac{\log(2-2\rho)}{\log(2\rho)} - 1, \qquad \textit{or alternatively,} \qquad g > \log_2\left(\frac{\rho}{1-\rho}\right).$$

Proof. As $p \to 1$, server 2 is clearly stable, as there are at most g low-value jobs at this server by definition, and high-value jobs arrive according to a Poisson process with rate $(1-p)\lambda \to 0$. Hence, the question of stability primarily concerns server 1. The arrival process to server 1 is non-Poisson, but we can still measure the time-average arrival rate to this server. Observe that server 1 receives jobs whenever there are exactly g low-value jobs at server 2. Moreover, as $p \to 1$, the impact of the high-value jobs on the number of low-value jobs at server 2 becomes negligible. Consequently, we may view the number of low-value jobs at server 2 as being distributed like the total number of jobs in an M/M/1/g system, with arrival rate $p\lambda \to \lambda$ and departure rate, μ. Hence, we may treat the arrival process to server 1 as the "loss process" of this

M/M/1/g system. The time-average arrival rate associated with this loss-process is known to be (cf. PASTA)

$$\lambda\left(\frac{(\lambda/\mu)^g(1-\lambda/\mu)}{1-(\lambda/\mu)^{g+1}}\right)=\lambda\left(\frac{(2\rho)^g(1-2\rho)}{1-(2\rho)^{g+1}}\right).$$

The system is stable if and only if the time-average arrival rate is less than the service rate, μ, which corresponds to the condition

$$\frac{(2\rho)^{g+1}(1-2\rho)}{1-(2\rho)^{g+1}}<1.$$

Simplifying, we have the stability condition $g>-\log(2-2\rho)/\log(2\rho)-1$. The alternative condition is stronger, but still valid because it can be shown that $-\log(2-2\rho)/\log(2\rho)-1<\log_2(\rho/(1-\rho))$ for all $\rho\in(1/2,1)$ (algebra omitted), while when $\rho\in(0,1/2)$, any $g\geq 0$ would suffice as both bounding quantities are negative. $\square$

B.4 Proof of Proposition 5.16

Proposition 5.16. *The value function for the M/M/1-PS queue with arrival rate λ, service rate μ, and values V is given by*

$$\eta_z=\frac{\lambda n(n+1)}{2(\mu-\lambda)(2\mu-\lambda)}\,\mathbb{E}[V]+\frac{n+1}{2\mu-\lambda}\,v^{\text{sum}}+c,$$

where c is a constant.

Proof. In proving this result, we invoke two earlier results. First, in an M/M/1-PS queue with n jobs, the mean response time of each job is given by (Sengupta and Jagerman, [117]),

$$\mathbb{E}[T|n]=\frac{n+1}{2\mu-\lambda}, \tag{B.1}$$

which interestingly is finite even if the system is somewhat overloaded (i.e., $\mu<\lambda<2\mu$). Second, the value function *with respect to mean response time* (rather than value-weighted mean response time) in an M/M/1 system under any work-conserving scheduling discipline (e.g., FCFS or PS) is given by (see [3, 92])

$$\frac{n(n+1)}{2(\mu-\lambda)}+c', \tag{B.2}$$

where c' is some constant. The value function (B.2) can be broken into the sum of (i) the *total* mean response time of the n jobs, and (ii) the expected *total additional* response time experienced by all future arrivals due to the n jobs currently in the system. Here, (i) is given by

$$n\cdot\mathbb{E}[T|n]=\frac{n(n+1)}{2\mu-\lambda}, \tag{B.3}$$

and hence, (ii) is obtained by subtracting (B.3), from (B.2), yielding

$$\left(\frac{n(n+1)}{2(\mu-\lambda)}+c\right)-\frac{n(n+1)}{2\mu-\lambda}=\frac{\lambda n(n+1)}{2(\mu-\lambda)(2\mu-\lambda)}+c'. \tag{B.4}$$

Next observe that since V and T are independent in an M/M/1-PS, we obtain the expected total additional *penalty* incurred by all future arrivals due to the n jobs currently in the system by multiplying (B.4) by $\mathbb{E}[V]$:

$$\frac{\lambda n(n+1)}{2(\mu-\lambda)(2\mu-\lambda)} \cdot \mathbb{E}[V] + c' \cdot \mathbb{E}[V]. \tag{B.5}$$

Moreover, the mean penalty incurred by the n jobs currently in the system is obtained by multiplying (B.1) by the total value of those jobs, v^{sum}:

$$\frac{n+1}{2\mu-\lambda} \cdot v^{\text{sum}}. \tag{B.6}$$

Finally to obtain the value function of interest (the one with respect to $\mathbb{E}[VT]$), we add (B.5) and (B.6), yielding the desired result (with $c = c' \cdot \mathbb{E}[V]$), completing the proof. □

Note that Proposition 5.16 is a new result. In contrast to our setting, [3, 92] give a value function for the M/M/1/-PS queue with respect to $\mathbb{E}[T]$ rather than $\mathbb{E}[VT]$. Meanwhile [83] gives a value function where the state-information includes the (remaining) service times.

B.5 Proof of Proposition 5.17

Proposition 5.17. *For a static basic policy α, yielding server-specific arrival processes (λ_i, V_i), the corresponding FPI policy, routes a job of value v to the server given by*

$$\alpha'(v) = \underset{i}{\operatorname{argmin}} \quad \frac{1}{2\mu_i - \lambda_i}\left(\frac{\lambda_i\,\mathbb{E}[V_i](n_i+1)}{\mu_i-\lambda_i} + v_i^{\text{sum}} + (n_i+2)v\right).$$

Proof. First, recall that with a static basic policy, the system decomposes to m independent M/M/1-PS queues, and the value function of the whole system is therefore the sum of the queue-specific value functions,

$$\eta_{\mathbf{z}}(\alpha) = \eta_{z_1}^{(1)}(\alpha) + \ldots + \eta_{z_m}^{(m)}(\alpha). \tag{B.7}$$

Each $\eta_{z_i}^{(i)}(\alpha)$ is given by (5.5) with the queue-specific $(\lambda_i, \mathbb{E}[V_i])$ defined by α, and the queue-specific service rate μ_i.

Given the value function is available, we can carry out the FPI step (5.4). For clarity, we omit the basic policy α from the notation. The admission penalty of a job with value v is equal to the change in the value function,[7]

$$a(v,i) = \eta_{\mathbf{z}\oplus(v,i)} - \eta_{\mathbf{z}},$$

where $\mathbf{z} \oplus (v,i)$ denotes the resulting state when a job with value v is added to server i. Given η_z is the sum of queue-specific terms (B.7), the change in $\eta_z(\alpha)$ is local to server i, and the admission penalty becomes

$$a(v,i) = \left(\eta_{z_1}^{(1)} + \ldots + \eta_{z_i\oplus v}^{(i)} + \ldots + \eta_{z_m}^{(m)}\right) - \left(\eta_{z_1}^{(1)} + \ldots + \eta_{z_m}^{(m)}\right) = \eta_{z_i\oplus v}^{(i)} - \eta_{z_i}^{(i)}. \tag{B.8}$$

[7]There are no immediate penalties associated with any state changes.

From Proposition 5.16 and (B.8) we have

$$a(v,i) = \frac{\lambda_i \, \mathbb{E}[V_i](n_i+1)}{(\mu_i-\lambda_i)(2\mu_i-\lambda_i)} + \frac{v_i^{\text{sum}}}{2\mu_i-\lambda_i} + \frac{(n_i+2)v}{2\mu_i-\lambda_i}, \tag{B.9}$$

where the first and second terms corresponds to the expected total additional penalty incurred by the future arrivals and the n_i jobs currently at server i, respectively. The last term is the expected penalty incurred by the new arrival of value v. FPI chooses the queue with the smallest expected penalty, $\alpha'(v) = \operatorname*{argmin}_i a(v,i)$, yielding (5.6). □

Bibliography

[1] Cisco systems localdirector. http://www.cisco.com/c/en/us/products/routers/localdirector-400-series/. 5.1

[2] Microsoft sharepoint 2010 load balancer. http://www.loadbalancer.org/sharepoint.php. 5.1

[3] S. Aalto and J. Virtamo. Basic packet routing problem. In *The thirteenth Nordic teletraffic seminar NTS-13*, pages 85–97, Trondheim, Norway, August 1996. B.4, B.4

[4] Joseph Abate and Ward Whitt. Transient behavior of the M/M/1 queue via Laplace transforms. *Advances in Applied Probability*, pages 145–178, 1988. 2.4.1, 2.4.1

[5] I. Adan and J Resing. A class of Markov processes on a semi-infinite strip. Technical Report 99-03, Eindhoven University of Technology, Department of Mathematics and Computing Sciences, 1999. 2.1.3, 3.2.3

[6] I.J.B.F. Adan, G.J. van Houtum, and J. van der Wal. Upper and lower bounds for the waiting time in the symmetric shortest queue system. *Annals of Operations Research*, 48: 197–217, 1994. 5.1

[7] Z. Aksin, M. Armony, and V. Mehrotra. The modern call-center: A multi-disciplinary perspective on operations management research. *Prod. Oper. Manag.*, 16(6):665–688, 2007. 4.1

[8] G. Allon and I. Gurvich. Pricing and dimensioning competing large-scale service providers. *M&SOM*, 12(3):449–469, 2010. 4.1.2

[9] E. Altman, U. Ayesta, and B.J. Prabhu. Load balancing in processor sharing systems. *Telecommunication Systems*, 47(1):35–48, 2011. 5.1, 5.2

[10] J. Anton. One-minute survey report #488: Agent compensation & advancement, 2005. Document Tracking Number SRV488-080305. 4.1

[11] M. Armony. Dynamic routing in large-scale service systems with heterogeneous servers. *Queueing Syst. Theory Appl.*, 51(3-4):287–329, 2005. 4.1.2

[12] M. Armony and A. R. Ward. Fair dynamic routing in large-scale heterogeneous-server systems. *Operations Research*, 58:624–637, 2010. 4.1.2, 4.3

[13] Søren Asmussen. *Applied probability and queues*, volume 51. Springer Science & Business Media, 2003. 2.1.1

[14] R. Atar. Scheduling control for queueing systems with many servers: Asymptotic optimality in heavy traffic. *Ann. Appl. Probab.*, 15(4):2606–2650, 2005. 4.1.2

[15] R. Atar, Y. Y. Shaki, and A. Shwartz. A blind policy for equalizing cumulative idleness. *Queueing Syst.*, 67(4):275–293, 2011. 4.1.2

[16] Eitan Bachmat and Hagit Sarfati. Analysis of size interval task assigment policies. *Performance Evaluation Review*, 36(2):107–109, 2008. 5.1

[17] Matt Bishop. *Computer security: art and science*, volume 200. Addison-Wesley, 2012. 3.1

[18] Flavio Bonomi. On job assignment for a parallel system of processor sharing queues. *IEEE Trans. Comput.*, 39(7):858–869, July 1990. 5.1, 5.4.5

[19] S. Borst, A. Mandelbaum, and M. I. Reiman. Dimensioning large call centers. *Operations Research*, 52(1):17–34, 2004. 4.1.2

[20] M. Bramson, Y. Lu, and B. Prabhakar. Randomized load balancing with general service time distributions. In *Proceedings of the ACM Special Interest Group on Computer Systems Performance, SIGMETRICS 2010*, June 2010. 5.1

[21] Larry Bridwell. Computer virus prevalence survey. *ICSA Labs*, 2004. 3.1

[22] L Bright and Peter G Taylor. Calculating the equilibrium distribution in level dependent quasi-birth-and-death processes. *Stochastic Models*, 11(3):497–525, 1995. 2.1.1

[23] Carey Bunks, Dan McCarthy, and Tarik Al-Ani. Condition-based maintenance of machines using hidden markov models. *Mechanical Systems and Signal Processing*, 14(4): 597–612, 2000. 3.2.2

[24] Maximiliano Caceres. Syscall proxying - simulating remote execution, 2002. URL `http://www.coresecurity.com/files/attachments/SyscallProxying.pdf`. 3.1

[25] G. P. Cachon and P. T. Harker. Competition and outsourcing with scale economies. *Manage. Sci.*, 48(10):1314–1333, 2002. 4.1.2

[26] G. P. Cachon and F. Zhang. Obtaining fast service in a queueing system via performance-based allocation of demand. *Manage. Sci.*, 53(3):408–420, 2007. 4.1.2

[27] P. Cahuc and A. Zylberberg. *Labor Economics*. MIT Press, 2004. 4.1.1

[28] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):1–49, 2002. 5.1

[29] Center for Strategic and International Studies. Net losses: Estimating the global cost of cybercrime, June 2014. URL `http://www.surfline.com/surf-news/maldives-surf-access-controversy-update_75296/`. [Online; posted 9-June-2014]. 3.1

[30] Ram Chakka and Isi Mitrani. Heterogeneous multiprocessor systems with breakdowns: performance and optimal repair strategies. *Theoretical Computer Science*, 125(1):91–109, 1994. 3.2.1

[31] Carri W Chan, Vivek F Farias, and Gabriel Escobar. The impact of delays on service times in the intensive care unit. Technical report, Working Paper, 2014. 2.2

[32] S.-F. Cheng, D. M. Reeves, Y. Vorobeychik, and M. P. Wellman. Notes on equilibria in symmetric games. In *International Workshop On Game Theoretic And Decision Theoretic Agents (GTDT)*, pages 71–78, 2004. 4.4.2

[33] Gianfranco Ciardo and Evgenia Smirni. ETAQA: an efficient technique for the analysis of QBD-processes by aggregation. *Performance Evaluation*, 36:71–93, 1999. 2.1.3

[34] Gianfranco Ciardo, Alma Riska, and Evgenia Smirni. Equiload: a load balancing policy for clustered web servers. *Performance Evaluation*, 46:101–124, 2001. 5.1

[35] Gianfranco Ciardo, Weizhen Mao, Alma Riska, and Evgenia Smirni. ETAQA-MG1: an efficient technique for the analysis of a class of M/G/1-type processes by aggregation. *Performance Evaluation*, 57(3):235–260, 2004. 2.1.3

[36] Y. Cohen-Charash and P. E. Spector. The role of justice in organizations: A meta-analysis. *Organ. Behav. and Hum. Dec.*, 86(2):278–321, 2001. 4.1.2, 4.3

[37] J. A. Colquitt, D. E. Conlon, M. J. Wesson, C. O. L. H. Porter, and K. Y. Ng. Justice at the millennium: A meta-analytic review of 25 years of organizational justice research. *J. Appl. Psychol.*, 86(3):425–445, 2001. 4.1.2, 4.3

[38] Darpa Cyber Grand Challenge, 2016. URL `https://cgc.darpa.mil/`. 3.1

[39] F. de Véricourt and Y.-P. Zhou. Managing response time in a call-routing problem with service failure. *Operations Research*, 53(6):968–981, 2005. 4.1.1

[40] Mohammad Delasay, Armann Ingolfsson, and Bora Kolfal. Modeling load and overwork effects in queueing systems with adaptive servers. Technical report, Working paper, 2015. 2.2

[41] J.R. Diamant, W.Y. Hsu, D.H. Lin, and E.C. Scoredos. Automatic detection of vulnerability exploits, May 27 2014. URL `https://www.google.com/patents/US8739288`. US Patent 8,739,288. 3.1

[42] Sherwin Doroudi, Esa Hyytiä, and Mor Harchol-Balter. Value driven load balancing. *Performance Evaluation*, 79:306–327, 2014. 1

[43] Sherwin Doroudi, Brian Fralix, and Mor Harchol-Balter. Clearing analysis on phases: Exact limiting probabilities for skip-free, unidirectional, quasi-birth-death processes. *arXiv preprint arXiv:1503.05899*, 2015. 1, 3.2.3, 3.3.2, 7

[44] Steve Drekic and Winfried K Grassmann. An eigenvalue approach to analyzing a finite source priority queueing model. *Annals of Operations Research*, 112(1-4):139–152, 2002. 3.2.1

[45] Muhammad El-Taha and Bacel Maddah. Allocation of service time in a multiserver system. *Management Science*, 52(4):623–637, 2006. 5.1

[46] A. Ephremides, P. Varaiya, and J. Walrand. A simple dynamic routing problem. *IEEE Transacactions on Autonomic Control*, AC-25(4):690–693, 1980. 5.1

[47] A. K. Erlang. On the rational determination of the number of circuits. In E. Brockmeyer, H. L. Halstrom, and A. Jensen, editors, *The Life and Works of A. K. Erlang*, pages 216–221. The Copenhagen Telephone Company, 1948. 4.1.2

[48] F5 Products. Big-IP. http://www.f5.com/products/big-ip. 5.1

[49] Hanhua Feng and Vishal Misra. Mixed scheduling disciplines for network flows. *SIGMETRICS Perform. Eval. Rev.*, 31:36–39, September 2003. 5.1

[50] Dennis W. Fife. Scheduling with random arrivals and linear loss functions. *Management Science*, 11(3):429–437, January 1065. 5.1, 5.4.4

[51] G. J. Foschini and J. Salz. A basic dynamic routing problem and diffusion. *IEEE Transactions on Communications*, 26(3):320–327, 1978. 5.6.1, 5.6.3, 5.6.3

[52] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012. 2.2.1

[53] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 153–166. ACM, 2013. 2.1.3, 3.2.3

[54] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward. *Queueing Systems*, 77(2):177–209, 2014. 2.1.3, 3.2.3

[55] N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial, review, and research prospects. *M&SOM*, 5(2):79–141, 2003. 4.1, 4.1.1

[56] Michele Garetto, Weibo Gong, and Don Towsley. Modeling malware spreading dynamics. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1869–1879. IEEE, 2003. 3.2

[57] O. Garnett, A. Mandelbaum, and M. Reiman. Designing a call center with impatient customers. *M&SOM*, 4(3):208–227, 2002. 4.1.2

[58] Kevin M Gatzlaff and Kathleen A McCullough. The effect of data breaches on shareholder wealth. *Risk Management and Insurance Review*, 13(1):61–83, 2010. 3.1

[59] X. Geng, W. T. Huh, and M. Nagarajan. Strategic and fair routing policies in a decentralized service system. Working paper, 2013. 4.1.2

[60] S. M. Gilbert and Z. K. Weng. Incentive effects favor nonconsolidating queues in a service system: The principal-agent perspective. *Manage. Sci.*, 44(12):1662–1669, 1998. 4.1.2

[61] Ragavendran Gopalakrishnan, Sherwin Doroudi, Amy R. Ward, and Adam Wierman. Routing and staffing when servers are strategic. *Operations Research*, 2014. Forthcoming. 1

[62] Donald Gross, John F Shortle, James M Thompson, and Carl M Harris. *Fundamentals of queueing theory*. John Wiley & Sons, 2013. B.2

[63] Varun Gupta, Mor Harchol-Balter, Karl Sigman, and Ward Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation*, 64(9-12):1062–1081, October 2007. 5.1, 5.1, 5.4.2, 5.4.5

[64] I. Gurvich and W. Whitt. Scheduling flexible servers with convex delay costs in many-

server service systems. *M&SOM*, 11(2):237–253, 2007. 4.1.2

[65] B. Haji and S. M. Ross. A queueing loss model with heterogenous skill based servers under idle time ordering policies, 2013. Working paper. 4.4.1

[66] S. Halfin and W. Whitt. Heavy-traffic limits for queues with many exponential servers. *Operations Research*, 29(3):567–588, 1981. 4.1.2

[67] Lani Haque and Michael J Armstrong. A survey of the machine interference problem. *European Journal of Operational Research*, 179(2):469–482, 2007. 3.2.1

[68] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013. 4.1

[69] Mor Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49 (2), 2002. 5.1

[70] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013. 2.1.1, 2.4.1, 5.1, B.2

[71] Mor Harchol-Balter, Mark Crovella, and Cristina Murta. On choosing a task assignment policy for a distributed server system. *IEEE Journal of Parallel and Distributed Computing*, 59:204–228, 1999. 5.1

[72] Mor Harchol-Balter, Karl Sigman, and Adam Wierman. Asymptotic convergence of scheduling policies with respect to slowdown. *Perform. Eval.*, 49(1-4):241–256, September 2002. 5.2

[73] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew Young. Surprising results on task assignment in server farms with high-variability workloads. In *ACM Sigmetrics 2009 Conference on Measurement and Modeling of Computer Systems*, pages 287–298, 2009. 5.1

[74] R. Hassin and M. Haviv. *To Queue or Not to Queue: Equilibrium Behavior in Queueing Systems*. Kluwer, 2003. 4.1.2

[75] Qi-Ming He. *Fundamentals of matrix-analytic methods*. Springer, 2014. 2.1.1

[76] W. Hopp and W. Lovejoy. *Hospital Operations: Principles of High Efficiency Health Care*. Financial Times Press, 2013. 4.1

[77] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, 2012. 2.1.2

[78] Ronald A. Howard. *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. Wiley Interscience, 1971. 5.8

[79] Yih Huang, David Arsenault, and Arun Sood. Closing cluster attack windows through server redundancy and rotations. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 2, pages 12–pp. IEEE, 2006. 3.1, 3.2

[80] Lorine A Hughes and Gregory J DeLone. Viruses, worms, and trojan horses: Serious crimes, nuisance, or both? *Social science computer review*, 25(1):78–98, 2007. 3.1

[81] Esa Hyytiä. Lookahead actions in dispatching to parallel queues. *Performance Evaluation*, 70(10):859–872, 2013. (IFIP Performance'13). 5.2

[82] Esa Hyytiä and Samuli Aalto. Round-robin routing policy: Value functions and mean performance with job- and server-specific costs. In *7th International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, Torino, Italy, December 2013. 5.2

[83] Esa Hyytiä, Jorma Virtamo, Samuli Aalto, and Aleksi Penttinen. M/M/1-PS queue and size-aware task assignment. *Performance Evaluation*, 68(11):1136–1148, November 2011. (IFIP Performance'11). 5.1, B.4

[84] Esa Hyytiä, Samuli Aalto, and Aleksi Penttinen. Minimizing slowdown in heterogeneous size-aware dispatching systems. *ACM SIGMETRICS Performance Evaluation Review*, 40: 29–40, June 2012. (ACM SIGMETRICS/Performance conference). 5.2

[85] A. J. E. M. Janssen, J. S.H. van Leeuwaarden, and B. Zwart. Refining square-root safety staffing by expanding Erlang C. *Operations Research*, 59(6):1512–1522, 2011. 4.1.2

[86] E. Kalai, M. I. Kamien, and M. Rubinovitch. Optimal service speeds in a competitive environment. *Manage. Sci.*, 38(8):1154–1163, 1992. 4.1.2

[87] S Karlin and HM Taylor. *A first course in stochastic processes*. Acadmic Press, New York, 1975. 2.4.1

[88] Alan F Karr. Weak convergence of a sequence of markov chains. *Probability Theory and Related Fields*, 33(1):41–48, 1975. B.2

[89] J.F.C. Kingman. Two similar queues in parallel. *Biometrika*, 48:1316–1323, 1961. 5.1

[90] L. Kocaga, M. Armony, and A. R. Ward. Staffing call centers with uncertain arrival rates and co-sourcing, 2013. Working paper. 4.1.2

[91] B. Krishnamoorthi. On Poisson queue with two heterogeneous servers. *Operations Research*, 11(3):321–330, 1963. 4.4.2

[92] K. R. Krishnan. Joining the right queue: a state-dependent decision rule. *IEEE Transactions on Automatic Control*, 35(1):104–108, January 1990. 5.8.1, B.4, B.4

[93] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, 1999. 2.1.1, 2.3.1, 2.3.1, 2.3.1, 3.2.3

[94] Guy Latouche and V Ramaswami. A logarithmic reduction algorithm for quasi-birth-death processes. *Journal of Applied Probability*, pages 650–674, 1993. 2.1.1

[95] Y. Levy and U. Yechiali. An M/M/s queue with servers' vacations. *INFOR*, 14:153–163, 1976. 2.1.3

[96] Daming Lin, Viliam Makis, et al. Recursive filters for a partially observable system subject to random failure. *Advances in Applied Probability*, 35(1):207–227, 2003. 3.2.2

[97] W. Lin and P. Kumar. Optimal control of a queueing system with two heterogeneous servers. *IEEE Trans. Autom. Contr.*, 29(8):696–703, 1984. 4.1.1

[98] D. Liu and Y.Q. Zhao. Determination of explicit solution for a general class of Markov processes. *Matrix-Analytic Methods in Stochastic Models*, page 343, 1996. 2.1.1

[99] Patricia Y Logan and Stephen W Logan. Bitten by a bug: a case study in malware infection. *Journal of Information Systems Education*, 14(3):301, 2003. 3.1

[100] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R. Larus, and Albert Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Perform. Eval.*, 68(11):1056–1071, November 2011. 5.1

[101] J.C.S. Lui, R.R. Muntz, and D.F. Towsley. Bounding the mean response time of the minimum expected delay routing policy: an algorithmic approach. *IEEE Transactions on Computers*, 44(12):1371–1382, 1995. 5.1

[102] V Makis and X Jiang. Optimal replacement under partial observations. *Mathematics of Operations Research*, 28(2):382–394, 2003. 3.2.2

[103] G. S. Mokaddis, C. H. Matta, and M. M. El Genaidy. On Poisson queue with three heterogeneous servers. *International Journal of Information and Management Sciences*, 9: 53–60, 1998. 4.4.2

[104] A. Mukhopadhyay and R. R. Mazumdar. Analysis of load balancing in large heterogeneous processor sharing systems. http://arxiv.org/abs/1311.5806, November 2013. 5.1

[105] Marcel F Neuts. *Matrix-geometric solutions in stochastic models: an algorithmic approach*. Courier Dover Publications, 1981. 2.1.1, 3.2.3

[106] Tuan Phung-Duc. Exact solutions for M/M/c/Setup queues. *arXiv preprint arXiv:1406.3084*, 2014. 2.1.3

[107] Marco Pistoia and Corinne Letilley. IBM websphere performance pack: Load balancing with IBM secureway network dispatcher, October 1999. IBM Redbooks. 5.1

[108] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005. 5.8

[109] V Ramaswami and Guy Latouche. A general class of Markov processes with explicit matrix-geometric solutions. *Operations-Research-Spektrum*, 8(4):209–218, 1986. 2.1.1

[110] Malempati M Rao and Randall J Swift. *Probability theory with applications*, volume 582. Springer, 2006. B.2

[111] J. Reed and Y. Shaki. A fair policy for the G/GI/N queue with multiple server pools., 2013. Preprint. 4.1.2

[112] Alma Riska and Evgenia Smirni. Exact aggregate solutions for M/G/1-type Markov processes. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 86–96. ACM, 2002. 2.1.3

[113] Alma Riska and Evgenia Smirni. ETAQA solutions for infinite Markov processes with repetitive structure. *INFORMS Journal on Computing*, 19(2):215–228, 2007. 2.1.3

[114] Sheldon M. Ross. *Applied Probability Models with Optimization Applications*. Holden-Day Inc., 1970. 5.8

[115] T. L. Saaty. Time-dependent solution of the many-merver Poisson queue. *Operations Research*, 8(6):755–772, 1960. 4.4.2

[116] Jori Selen, Ivo Adan, Vidyadhar Kulkarni, and Johan van Leeuwaarden. The snowball effect of customer slowdown in critical many-server systems. *arXiv preprint arXiv:1502.02856*, 2015. 2.2

[117] B. Sengupta and D.L. Jagerman. A conditional response time of M/M/1 processor-sharing queue. *AT&T Bell Lab. Techn. J.*, 64(2):409–421, February 1985. B.4

[118] Andrei Sleptchenko, Jori Selen, Ivo Adan, and Geert-Jan van Houtum. Joint queue length distribution of multi-class, single server queues with preemptive priorities. *arXiv preprint arXiv:1411.3176*, 2014. 2.5.3

[119] Andreas Stathopoulos, Alma Riska, Zhili Hua, and Evgenia Smirni. Bridging ETAQA and ramaswami's formula for the solution of M/G/1-type processes. *Performance Evaluation*, 62(1):331–348, 2005. 2.1.3

[120] KJELL Stordahl. The history behind the probability theory and the queuing theory. *Telektronikk*, 103(2):123, 2007. 1

[121] Symantec. Severity assessment, 2016. URL `https://www.symantec.com/content/en/us/about/media/securityintelligence/SSR-Severity-Assesment.pdf`. 3.1

[122] T. Tezcan. Optimal control of distributed parallel server systems under the Halfin and Whitt regime. *Mathematics of Operations Research*, 33:51–90, 2008. 4.1.2

[123] T. Tezcan and J. Dai. Dynamic control of N-systems with many servers: Asymptotic optimality of a static priority policy in heavy traffic. *Operations Research*, 58(1):94–110, 2010. 4.1.2

[124] B. Van Houdt and J.S.H. van Leeuwaarden. Triangular M/G/1-Type and Tree-Like Quasi-Birth-Death Markov Chains. *INFORMS Journal on Computing*, 23(1):165–171, 2011. 2.1.1, 3.2.3

[125] J.S.H. van Leeuwaarden and E.M.M. Winands. Quasi-birth-and-death processes with an explicit rate matrix. *Stochastic models*, 22(1):77–98, 2006. 2.1.1, 2.1.2, 2.3.2, 2.3.5

[126] J.S.H. van Leeuwaarden, M.S. Squillante, and E.M.M. Winands. Quasi-birth-and-death processes, lattice path counting, and hypergeometric functions. *Journal of Applied Probability*, 46(2):507–520, 2009. 2.1.1, 2.1.2, 2.3.2, 2.3.5

[127] A. R. Ward and M. Armony. Blind fair routing in large-scale service systems with heterogeneous customers and servers. *Operations Research*, 61:228–243, 2013. 4.1.2

[128] Pieter Wartenhorst. N parallel queueing systems with server breakdown and repair. *European Journal of Operational Research*, 82(2):302–322, 1995. 3.2.1

[129] R.W. Weber. On optimal assignment of customers to parallel servers. *Journal of Applied Probability*, 15:406–413, 1978. 5.1

[130] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14: 181–189, 1977. 5.1, 5.6.1, 5.6.2

[131] Shanchieh Yang and G. de Veciana. Size-based adaptive bandwidth allocation: optimizing the average QoS for elastic flows. In *IEEE INFOCOM*, volume 2, pages 657–666, 2002. 5.2

[132] Wei T Yue and Metin Çakanyıldırım. A cost-based analysis of intrusion detection system configuration under active or passive response. *Decision Support Systems*, 50(1):21–31,

2010. 3.2

[133] Q Qiushi Zhu. *Maintenance optimization for multi-component systems under condition monitoring*. PhD thesis, Technische Universiteit Eindhoven, 2015. 3.2.2