Designing Distributed Systems for Heterogeneity

Philip Brighten Godfrey

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-82
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-82.html

May 21, 2009

Designing Distributed Systems for Heterogeneity

by

Philip Brighten Godfrey

B.S. (Carnegie Mellon University) 2002
M.S. (University of California, Berkeley) 2006

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Ion Stoica, Chair
Professor Scott Shenker
Professor David Aldous

Spring 2009

The dissertation of Philip Brighten Godfrey is approved:

Chair Date

Date

Date

University of California, Berkeley

Spring 2009

Designing Distributed Systems for Heterogeneity

Philip Brighten Godfrey

Abstract

Designing Distributed Systems for Heterogeneity

by

Philip Brighten Godfrey
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Ion Stoica, Chair

Modern distributed and networked systems are highly heterogeneous in many dimensions, including available bandwidth, processor speed, disk capacity, security, failure rate, and pattern of failures. The theme of this dissertation is that this heterogeneity can not only be handled, but rather should generally be viewed as an asset.

We begin by introducing a framework, the price of heterogeneity, to model the effect of heterogeneity in parallel and distributed systems. Our results in this framework show broad classes of systems in which heterogeneity cannot be a disadvantage. We then develop practical methods for distributed systems to adapt to and take advantage of heterogeneity. The Y_0 distributed hash table achieves improved load balance, route length, and congestion with low overhead in environments with heterogeneous node capacities, such as bandwidth or processing speed. Addressing heterogeneity in reliability, we show that randomization in node selection strategies typically reduces failure rates—a property that permits better understanding of subtle properties of existing systems, as well as the design of new systems. Finally, we study how to improve stability in the Internet's interdomain routing protocol, while carefully managing tradeoffs with network operators' perferred routes. These results show how both performance and reliability can be improved in heterogeneous environments.

Professor Ion Stoica
Dissertation Committee Chair

To my family

Contents

List of Figures

List of Tables

Acknowledgments

I am greatly indebted to my thesis advisor, Ion Stoica. For more than six years he has been a brilliant and reliable force guiding me towards both questions and answers. He has given me the taste and confidence to do important research, and for that reason his influence on me will last many years. A second person who stands out as greatly shaping my career at Berkeley is Scott Shenker, who has perhaps the highest known bandwidth and lowest latency for emission and absorption of novel research ideas. David Aldous deserves many thanks not only for serving on my dissertation committee, but also for teaching two courses that gave me a solid statistical foundation and enabled me to do some of the research that appears herein.

I have been fortunate to have an amazing set of colleagues at Berkeley. The work in Chapter 2 of this dissertation was joint with Richard Karp [46]; Chapter 3 was with Ion Stoica [48]; Chapter 4 was with Scott Shenker and Ion Stoica [47]; and Chapter 5 was with Matthew Caesar, Ian Haken, Yaron Singer, Scott Shenker, and Ion Stoica [45]. Many friends and colleagues provided helpful comments and discussions on this work: Bryan Clark, Anwitaman Datta, Alex Fabrikant, Rodrigo Fonseca, Karthik Lakshminarayanan, David Molnar, Christos Papadimitriou, Satish Rao, Lakshminarayanan Subramanian, Jane Valentine, Hakim Weatherspoon, and those that I have missed. Alex Fabrikant provided essential $n_0$menclatural assistance. The authors of [8, 14, 52, 106, 115] were generous in sharing their data sets. The authors of [33] shared a simulator which was the starting point for our simulations of Chapter 5. A National Science Foundation Graduate Research Fellowship and Cisco Systems provided financial support for these projects.

During my graduate studies, I very much enjoyed and learned from my collaborations on work not included in this dissertation, with Kamalika Chaudhuri, Alex Dimakis, Igor Ganichev, Brad Karp, John Kubiatowicz, Karthik Lakshminarayanan, Kannan Ramchandran, Satish Rao, David Ratajczak, Sylvia Ratnasamy, Sean Rhea, Sonesh Surana, Kunal Talwar, Martin Wainwright, Yunnan Wu, and Harlan Yu. In addition, I am grateful to my colleagues who made it a pleasure to work here, including those already listed and Byung-Gon Chun, Lisa Fowler, Kris Hildrum, Dilip Joseph, Jayanthkumar Kannan, Henry Lin, Blaine Nelson, Lucian Popa, Michael Schapira, and Hoeteck Wee.

Berkeley has been a unique place to work at the intersection of networking and theory, and I would like to recognize the professors who encouraged me in this area:

Richard Karp, Christos Papadimitriou, Satish Rao, Scott Shenker, and Ion Stoica.

Many people had a strong influence on me and gave me the excitement about mathematics, computer science, and research that led me to pursue graduate studies. In this regard I am indebted to my undergraduate research advisor, Lenore Blum; a set of Carnegie Mellon classmates, Bryan Clark, Kevin Milans, and James Pistole; David Scott of Ripon College, who demonstrated how much fun discrete math could be; Christine Stewart, the dedicated advisor of the math club at Ripon High School; and the man who would never be caught on a street corner without a 2×3 matrix in his back pocket, the Big W himself, Jim Watson.

I leave Berkeley with an unparalleled set of friends. I will not forget the late nights in the Great Hall, sushi dinners, hikes, gigs, countless hours that put a smile on my face, and certain activities best left unmentioned. I cannot imagine what Berkeley would have been like without Ana Ramírez Chang, Bor-Yuh Evan Chang, Omid Etesami, Alex Fabrikant, Joseph D. Flenner, Nazanin Shahrokni, the musicians of Brasshopper, the devious residents of Highland Place, and my friends from International House—who, to my great fortune, are too numerous to mention here. I would also be remiss to leave out Frankie and Stella, Maggie and May, Emslie, Grizza von Grazzelstein, Detroit McMoog, Augustus Gus "Eddie the Meatball" Gunderson, and Ava.

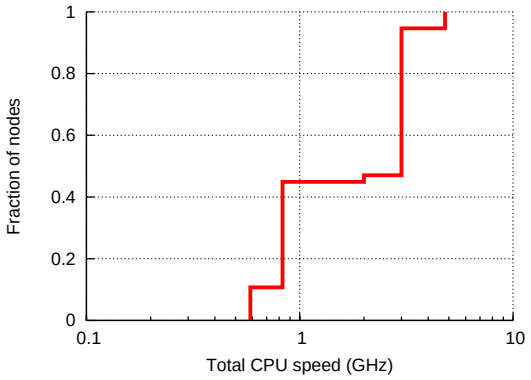
I am especially grateful to my parents, Eric and Ann Marie Godfrey, and to my brother, Forest Godfrey, for their continual encouragement. It was invaluable to be able to retreat to a home filled with family and good food. Finally, I owe an immeasurable debt to Jane Valentine, whose unwavering love and support have made me a better person.

Chapter 1

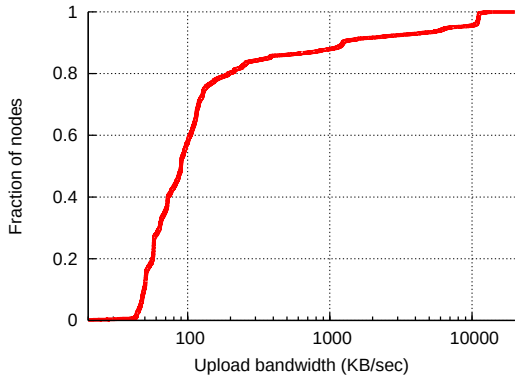
Introduction

Distributed and networked systems have become highly heterogeneous. Rather than running on clusters or supercomputers composed of identical nodes, today's distributed systems have wide variation in participants' failure rates, bandwidth, processing speed, security, and other dimensions. This heterogeneity can result from many factors, ultimately driven by explosive growth of the Internet. Modern Internet applications have giant scale, so that even within a single data center there are various generations of equipment. They may involve specialized nodes, from well-provisioned servers down to nodes whose specialty is fitting in pockets. Participating nodes are often owned and administered by many different entities. And they are deployed globally on top of the largest distributed system of them all, the Internet, which exemplifies all of these characteristics.

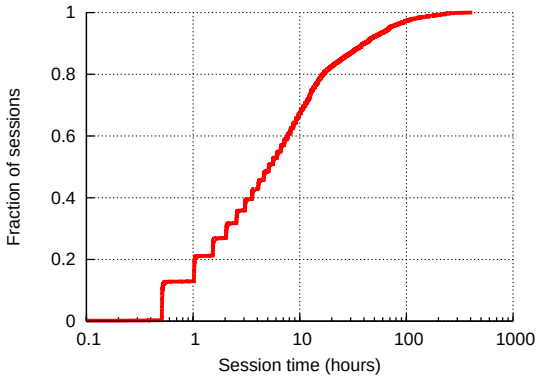
One can get a quantitative sense of this heterogeneity through measurements of particular systems (Figure 1.1). In the following systems, we will compare the 95th and 5th percentiles of distributions to remove outliers. Emulab, a network testbed housed in a data center at the University of Utah, consists of nodes which differ by $8\times$ in total CPU speed (summing the cores) [34]. BitTorrent, the massively popular peer-to-peer file distribution system, has peers whose uplink bandwidths differ by $154\times$ [61]. The voice-over-IP application Skype has a network of superpeers with widely varying reliability: the lengths of their sessions—that is, periods of continuous uptime—differ by $136\times$ [52]. And the session lengths of routes on the Internet differ by about $1,150,000\times$, based on one month of data from Route Views [102]: some sessions lasted the entire period, others were separated by only the one-second measurement granularity, and the entire spectrum in between was well represented.



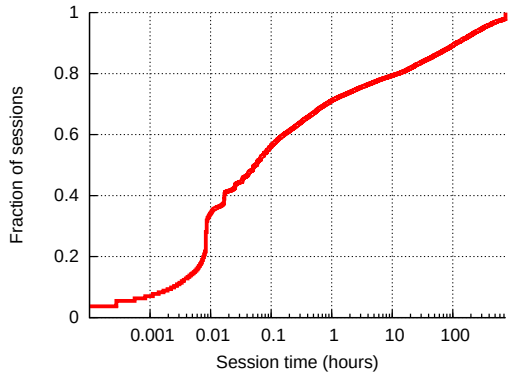
(a) CPU speeds in Emulab [34].



(b) Upload bandwidth of BitTorrent users, as measured by Isdal et al. [61].



(c) Session times of Skype superpeers, as measured by Guha et al. [52].



(d) Session times of Internet routes, derived from one month of Route Views data collected during December 2008 [102].

Figure 1.1: Distributions of CPU speed, bandwidth, and reliability in several distributed systems.

This dissertation concerns building distributed systems in such heterogeneous environments, with the theme that although heterogeneity can be challenging to deal with, it should generally be seen as an *advantage* to be exploited in systems design. Many systems take the easy way out and assume identical components. But if we adapt systems to their variable characteristics, we can often achieve *better performance in heterogeneous environments than in comparable homogeneous environments.*

Our contributions build an understanding of the effect of heterogeneity in distributed systems, and develop techniques to adapt to and take advantage of heterogeneity:

Chapter 2 introduces a framework, the price of heterogeneity, to model the effect of heterogeneity in parallel and distributed systems. In this framework, we show broad classes of cases in which heterogeneity cannot be a significant disadvantage.

Chapter 3 shows how to adapt distributed hash tables to heterogeneity in node capacities such as bandwidth or processing speed, obtaining improved load balance, route length, and congestion with low overhead.

Chapter 4 addresses heterogeneity in reliability. We show that randomization in node selection strategies typically reduces failure rates—an often subtle property that permits better understanding and design of distributed systems.

Chapter 5 studies how to improve stability in the Internet's interdomain routing protocol, while carefully managing tradeoffs with network operators' preferred routes.

We outline each of these chapters next.

1.1 Modeling the Effect of Heterogeneity

We begin with a high-level question which will set the stage for later chapters of this thesis: *how does the performance of a system depend on the amount of heterogeneity of its capacity distribution?*

It is not surprising that increasing heterogeneity can either improve or degrade performance, depending on the particular system, its workload, its environment, and its notion of performance. Nevertheless, we are able to show large classes of cases when performance *roughly monotonically improves* as the heterogeneity of the participants increases.

To show this, we introduce a theoretical framework, the **price of heterogeneity (PoH)**, that measures the worst-case effect of increasing heterogeneity. We are given a cost function $g(C, W)$ describing a system's performance as a function of its nodes' capacities C and workload W. For example, in a scheduling problem, $g(C, W)$ could represent the minimal time to complete a set of jobs with lengths W on a set of processors with speeds C. We next formalize the statement that a set of capacities C' is "more heterogeneous" than another set C ($C \preceq C'$) with majorization, a commonly used partial order that is consistent with variance. Finally, the price of heterogeneity of g compares the ratio of costs when moving from any capacities C to any more heterogeneous capacities C', taking the worst case over all W, C, and C':

$$\sup_{W,C,C':\, C \preceq C'} \frac{g(C', W)}{g(C, W)}.$$

The power of this framework is that it abstracts away the details of the workload and capacity distribution: for cost functions that have a small PoH, we are guaranteed that performance cannot significantly degrade in more heterogeneous environments. In addition, when the PoH is small, the homogeneous capacity distribution is provably close to the worst case, which is useful for testing systems.

Our analysis shows that many important models of systems do, in fact, have small PoH. We prove bounds on the PoH ranging from $2 - \frac{1}{n}$ to $O(\log n)$ where n is the number of nodes, for scheduling problems with a broad range of objective functions, with job precedence constraints, and with resource or shared-lock constraints; for a load balancing problem motivated by distributed hash tables; and for a low-diameter network construction problem. Most of the upper bounds are obtained via a "Simulation Lemma" that we introduce, which appears to have fairly wide applicability. One problem we consider, scheduling with release times, has unbounded PoH.

In summary, Chapter 2 provides quantitiatively justified intuition for the view of heterogeneity as an asset, setting the stage for later chapters of this thesis which introduce systems design techniques to take advantage of heterogeneity. Moreover, in Chapter 4, we will build on the framework presented here to analyze the effect of heterogeneous failure patterns.

1.2 Heterogeneity and Load Balance in Distributed Hash Tables

The proposition that heterogeneity is helpful for some system rests on the assumption that the system effectively adapts to its environment. The remaining chapters of this thesis will develop techniques for adapting to and taking advantage of heterogeneity.

Chapter 3 concerns design of hetergeneity-aware distributed hash tables (DHTs). A DHT is a highly scalable distributed storage system that manages objects stored in an identifier (ID) space. Responsibility for the ID space is partitioned among the nodes: each node is given a random ID x and owns the set of IDs that are "closest" to x. To route to the owner of any given ID in order to store or retrieve an object, the DHT maintains an overlay network with routing tables typically of size $O(\log n)$ and route lengths of $O(\log n)$ nodes.

One central challenge in DHT design is how to balance load. Even if all nodes have the same capacity, the random assignment of node IDs results in some nodes owning $O(\log n)$ times their "fair share" of the ID space [114]. And since most DHTs treat all participating nodes equally, the imbalance can significantly increase as the heterogeneity of the system increases.

The technique of virtual servers is used in DHTs to improve load balance, and had also been proposed (though not evaluated) to handle heterogeneity. Each physical node simply instantiates multiple virtual servers with random IDs that act as peers in the DHT. In the case of a homogeneous system, maintaining $\Theta(\log n)$ virtual servers per physical node reduces the load imbalance to a constant factor. To handle heterogeneity, each node picks a number of virtual servers proportional to its capacity. Unfortunately, virtual servers incur a significant cost: a node with k virtual servers must maintain k sets of overlay links.

Our solution, a DHT called Y_0, addresses the above drawbacks. Y_0 is a modified version of the Chord DHT [114] and uses virtual servers, but with a twist. Instead of picking k virtual servers with *random* IDs, a node *clusters* those IDs in a random fraction $\Theta(k/n)$ of the ID space. This allows the node to share a single set of overlay links among all k virtual servers.

Two main benefits result from this technique. First, in both the homogeneous and heterogeneous cases, we prove that our scheme avoids the factor k inflation in overhead, yet still maintains a good load balance. Second, our technique allows higher-capacity nodes to pick a denser set of links in a structured way—rather than the essentially random

set of links that result in the typical virtual server technique. This reduces route length, and hence congestion, in the heterogeneous case.

To numerically evaluate the effect of heterogeneity, we simulate both Y_0 and Chord in several node capacity distributions. We find that both DHTs benefit from heterogeneity in load balance and route length, because we are able to discard very low-capacity nodes, leaving behind a denser and more efficient "core" network. However, Y_0's topology results in a larger improvement, with mean route lengths in a real-word distribution of node capacities being 33% shorter than in a homogeneous environment.

1.3 Minimizing Churn in Distributed Systems

Chapter 4 addresses heterogeneity in reliability: distributed systems typically have widely varying expected time-to-failure both across their components at any given moment in time, and across time for individual components. Our goal is to take advantage of heterogeneity to minimize churn—change in the set of participating nodes due to joins, graceful leaves, and failures—in order to avoid the problems that churn causes, like dropped messages, data inconsistency, increased user-experienced latency, or increased bandwidth use.

We introduce the study by way of an example. Consider an overlay multicast system in which a root node is broadcasting a stream of video to a set of interested users. The nodes are arranged in a multicast tree built as follows: each node, upon arrival or when one of its ancestors in the tree fails, queries m random nodes in the system, and picks as its new parent the node through which it has the lowest latency to the root. Clearly, increasing m better adapts the tree to the underlying topology, but it also has the nonobvious result that *the tree suffers from more churn as m increases*, in the sense that nodes are more frequently disconnected from the root, which has an impact on quality of service. Why does this occur?

Our study will explain this effect. The core model we deal with, which can be used to describe many systems, is as follows: out a set of n nodes, we desire to select k nodes to be "in use"; when one of these fails, we select another available node to replace it. The goal is to select nodes to minimize churn: the failure rate of in-use nodes. We simulate the churn of a set of node selection strategies, using traces of node availability in a diverse set of distributed systems. Common strategies, like picking the longest-lived avail-

able node, perform well. However, two kinds of node selection strategy that are agnostic to nodes' history are particularly interesting. The **Random Replacement** (RR) strategy replaces a failed node with a uniform-random available node. **Preference List** (PL) strategies arise as a result of optimizing for a metric other than churn: rank the nodes according to some fixed preference order, and when a replacement is necessary, pick the most preferred available node. Examples include ranking nodes by latency or by distance between their identifiers (as occurs in DHTs). Note that we use the term PL specifically in the case that the preference order is *not* directly related to churn.

Since they pick nodes in a way unrelated to their stability, it is not surprising that PL strategies result in churn rates on the in-use nodes that are similar to the mean failure rate among all available nodes. One might expect that RR should perform similarly, since it picks uniform-random replacements. However, it turns out that RR achieves much lower churn, typically within a factor of less than 2 of strategies that *intelligently* pick reliable nodes based on their history! We find that this result is quite robust, appearing in traces of node availability in Skype, Gnutella, web sites, corporate desktop PCs, and the wide-area testbed PlanetLab.

To explain the low churn achieved by RR, we analyze it in a stochastic model. With an (unrealistic) exponential session time distribution, RR is no better than Preference Lists, but RR's churn rate decreases as the distributions become more heterogeneous; in the language of Chapter 2, the price of heterogeneity of RR is 1. The intuition is that RR is similar to picking a node at a random point in its history: it is biased towards landing in a longer session of continuous uptime, since the node spends longer in a long session than in a short one.

We can now explain the earlier overlay multicast example: As m increases, node selection moves from being like RR to being like a PL strategy, where the preference list is based on latency. We also demonstrate and explain RR or PL-like behavior in existing designs for the topology of DHT overlay networks, replica placement in DHTs, and anycast server selection. For example, we can reduce request failure rate in the Chord DHT by 29% in a real-world pattern of failures, simply by adding some randomization to the selection of neighbors.

Thus, our results explain subtle performance differences in existing systems, as well as showing that randomization is a simple and general technique for minimizing churn in heterogeneous environments.

1.4 Stabilizing Internet Routing

A number of studies [36,70,124] point to stability as a key problem for the Border Gateway Protocol (BGP), the Internet's interdomain routing protocol. Network failures and recoveries, policy changes, and the BGP convergence process itself currently cause routers to generate roughly 136 prefix update messages per minute; that is, each of the 316,366 IP prefixes is updated on average once every 2.7 days (based on data from Huston [60] as of May 20, 2009, generated from a BGP peering with AS 131072).

This instability causes two main problems. First, in the control plane, route instability has the potential to stress routers' resources. While current average-case update rates do not cause significant CPU utilization [3], there is concern that extreme conditions like bursts of updates from multiple neighbors or future update loads resulting from rapidly increasing routing table size could overload routers, delay convergence, or necessitate more expensive routers. These problems led the Internet Architecture Board to identify update churn as one of the challenges for future scalability of the routing system [84]. A second problem of more immediate impact is in the data plane: route changes cause transient routing anomalies such as forwarding loops. Studies indicate that route instability is a major cause of unreliability in packet delivery—especially for bursts of multiple packet losses, which are the hardest for applications to deal with [68,124]. This problem is especially critical as real-time applications like video and voice become more prevalent.

In Chapter 5, we study how route selection schemes can avoid these changes in routes. Similar to Chapter 4, our techniques function by taking advantage of heterogeneity in the pattern of events. However, rather than considering primarily stability, we now focus on the fundamental tradeoffs that arise from balancing stability with other objectives.

Specifically, we characterize the tradeoffs between **interruption rate**, our measure of stability; **availability** of routes; and **deviation** from the network operator's preferred routes. We develop algorithms to bound the set of feasible points in the tradeoff spaces between these three cost metrics. We also propose a new approach, **Stable Route Selection** (SRS), which uses flexibility in route selection to improve stability without sacrificing availability, and with a controlled amount of deviation. Intuitively, SRS works by using heterogeneity in two ways. First, it exploits heterogeneity across available routes, so that when one route is in a period of instability, another is stable. Second, it exploits heterogeneity across time: most individual routes have a very skewed distribution of session

lengths, with some very long sessions and many short sessions (as we saw in Figure 1.1). This heterogeneity across time enables SRS to avoid many short sessions, and thus many route changes, while deviating from the most preferred route for only a small amount of time.

Our large-scale simulation results show that SRS can significantly improve stability with limited deviation. We implement our protocol in a software router, Quagga, and confirm in cluster deployment that SRS's gains in route stability translate to improved reliability in the data plane. Finally, we evaluate SRS under direct feeds of route update traffic from Internet routers. In this case, we observe much less improvement, but SRS can still improve stability when multiple disjoint paths are available.

1.5 Dissertation Plan

This dissertation proceeds as follows. Chapter 2 models the effect of heterogeneity in parallel and distributed systems. Chapter 3 describes our heterogeneity-aware distributed hash table, Y_0. Chapter 4 studies general techniques for minimizing churn in distributed systems whose components have heterogeneous reliability, with particular focus on the random replacement strategy. Chapter 5 studies how to avoid failures in the context of Internet routing. Finally, we conclude with open problems in Chapter 6.

Chapter 2

Modeling the Effect of Heterogeneity

2.1 Introduction

Chapter 1 established that modern distributed systems—such as peer-to-peer systems and the Internet's routing protocols—operate in highly heterogeneous environments: link and node reliability, and capacity like bandwidth, disk space, and processing speed, vary by orders of magnitude. Moreover, systems often have to cope with environments that change over time or that are difficult for the designer to characterize in advance.

Given this diversity and uncertainty of environments, it is useful to understand how characteristics of the participating components affect performance of the system. This chapter addresses the following basic question: *how does the performance of a system depend on the amount of heterogeneity of its capacity distribution?* As a more concrete special case, suppose that in distributed system A, all nodes have the same capacity; system B has the same total capacity but there is higher variance among the nodes' capacities. Does A or B perform better?

The answer, of course, is "yes": either A or B performs better, depending on the particular system, its workload, its environment, and its notion of performance. As a simple example, consider two parallel systems: a homogeneous system A with ten 1000 MHz processors and a more heterogeneous system B with one 5.5 GHz and nine 500 MHz processors. If the workload consists of ten jobs that take 1 second on a 1 GHz processor, then system A can complete the workload in 1 second. But in system B we could either put at least one job on a slow processor, or all jobs on the fast processor; in either case, the completion time roughly doubles to about 2 seconds. So system A performs better even

though both systems have the same total capacity. On the other hand if the workload is bottlenecked by one large job then system B might be more than 5 times faster.

It may seem, then, that the effect of heterogeneity is sensitive to the particulars of the system and its environment. Is there any hope that one can characterize the *macroscopic* effects of heterogeneity?

This chapter introduces a theoretical framework, the *price of heterogeneity*, that measures the worst-case effect of increasing heterogeneity. Using this framework, we find that although heterogeneity can be either beneficial or detrimental depending on the particular instance, it is nevertheless possible to broadly characterize the effect of heterogeneity in a way that can inform systems design and testing. Specifically:

- Our analysis of the price of heterogeneity for a number of scheduling and graph construction problems shows that for a large class of models of parallel and distributed systems, increasing heterogeneity cannot possibly be very detrimental—regardless of the details of the workload and capacity distribution.
- This analysis also demonstrates that for many models of parallel systems, the homogeneous capacity distribution is provably close to the worst case, which is useful for testing systems.

In summary, this chapter provides quantitiative justification for the view of heterogeneity as an asset, setting the stage for later chapters of this thesis which introduce systems design techniques to take advantage of heterogeneity. Moreover, in Chapter 4, we will build on the framework presented here to analyze the effect of heterogeneous failure patterns.

Model. We focus in this chapter on parallel systems which can be modeled by associating a capacity with each node: that is, a certain amount of a limited resource such as processing speed, bandwidth, memory, or disk space.

After using majorization to quantify the "amount of heterogeneity" of a capacity distribution, we study what we call the **price of heterogeneity (PoH)**. Informally, a cost function $g(C, W)$ describing a system's performance has price of heterogeneity α when for any workload W and capacities C, cost cannot increase by more than a factor α if C becomes arbitrarily more heterogeneous. In the job scheduling example above, W specifies the job

lengths, C specifies the processor speeds, and $g(C, W)$ is the makespan: the minimum completion time of any schedule of jobs W on processors C.

The price of heterogeneity provides characterizes the worst-case increase in cost due to increasing heterogeneity. For example, if heterogeneity always helps, then the price of heterogeneity of the cost function is 1. At a high level, we could hope to classify a parallel system's price of heterogeneity as being either *constant*, in which case increasing heterogeneity can never be much of a disadvantage, or *unbounded*, indicating that increasing heterogeneity can be quite detrimental. By classifying multiple systems in this way, we may begin to gain insight about what characteristics of a problem determine whether heterogeneity is generally good for it.

In addition to providing theoretical insight in these ways, if we have a cost function that is a good model of a real system, a practical application of the price of heterogeneity is to provide test cases that are provably close to the worst possible capacity distribution. This is useful, for example, when testing a system which the designer wishes to be deployable in a wide range of (possibly unknown) capacity distributions. In Section 2.9, we will discuss one such case, load balancing in distributed hash tables.

Connection with parallelism. An important special case restricts capacities so there are m nodes of capacity n/m and $n - m$ of capacity 0. In this case, increasing heterogeneity (according to the definition we will give in Section 2.2) corresponds to decreasing m, and thus decreasing parallelism. As a consequence, the price of heterogeneity upper-bounds the "value of parallelism": the maximum benefit obtained by increasing parallelism at the expense of decreasing processor speed. In queueing systems, it is well known that parallelism can be highly valuable (see Section 2.10). In contrast, many of our results will place limits on the value of parallelism in other scheduling models by upper-bounding the price of heterogeneity.

Results. Our bounds on the price of heterogeneity are summarized in Table 2.1. In this chapter we focus on scheduling problems, but we also give a network design example to show the generality of the model. Most of the upper bounds are obtained via what we call the Simulation Lemma, which shows how to "simulate" a set of capacities C using a more heterogeneous set C' by mapping each machine in C onto a single machine in C'. This lemma may also be useful in contexts other than the price of heterogeneity; for example, an easy corollary is that for any fixed set of capacities, as job lengths become arbitrarily

Problem	Price of heterogeneity	Reference
Minimum makespan scheduling	$= 2 - 1/n$	Thm. 2
Scheduling on related machines	$O(1)$	Cor. 1, 2
Precedence constrained scheduling (PCS)	$O(\log n)$	Cor. 3
PCS restricted to unit-length jobs	≤ 16	Cor. 5
Resource constrained scheduling	$\leq$ PoH of PCS	Thm. 5
Scheduling with release times, job lengths $\in [1, k]$	$\Omega(k)$	Thm. 6
Minimum network diameter, bounded degree	≤ 2	Thm. 7
DHT load balancing	≤ 2	Cor. 7

Table 2.1: Bounds on the price of heterogeneity shown in this chapter.

more homogeneous, optimal makespan can increase by a factor of $2 - o(1)$ and no more.

In addition, we show two lower bounds. First, in a model motivated by queueing systems, we observe that if jobs have release times before which they cannot be executed and we wish to minimize average or maximum job latency, the price of heterogeneity is $\Omega(k)$ when job sizes are in $[1, k]$. Second, we show that the simulation method fails spectacularly for precedence constrained scheduling (PCS): remapping the work of each C-machine onto a single C' machine can inflate the makespan by a factor $n/4$—nearly as bad as simply putting all the jobs on the single fastest machine! Intuitively, with dependence between jobs, lost parallelism is fatal. But our upper bound shows that a more intelligent placement does exponentially better, leading to a $O(\log n)$ PoH. An interesting and apparently nontrivial open question is whether PCS has $\Theta(1)$ price of heterogeneity.

In summary, the "batch" scheduling problems which we study, where all jobs arrive at time 0, have low price of heterogeneity (and hence, little value of parallelism). Even precedence and resource constrained scheduling—which provide a fairly rich set of constraints that can model, for example, relative ordering of jobs and the requirement of jobs to hold shared locks—have $O(\log n)$ PoH. On the other hand, the queueing-motivated scheduling with release times problem has unbounded PoH, even for $n = 2$.

The rest of this chapter is as follows. Our model is specified in Section 2.2. We introduce the Simulation Lemma in Section 2.3, and bound the price of heterogeneity of various cost functions in Sections 2.4-2.8. In Section 2.9, we present a scenario in which our results provide a worst case for testing. We discuss related work in Section 2.10 and conclude in Section 2.11.

2.2 Model

To define what it means for one capacity distribution C' to be more heterogeneous than another distribution C, we use the majorization partial order. Given two nonnegative vectors $C = (c_1, \ldots, c_n)$ and $C' = (c'_1, \ldots, c'_n)$, we say that C' *majorizes* C, written $C' \succeq C$, when

$$\forall k \ \sum_{i=1}^{k} c'_{[i]} \geq \sum_{i=1}^{k} c_{[i]} \quad \text{and} \quad \sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} c_i,$$

where $c_{[i]}$ denotes the ith largest component of C. Note the implicit assumption that elements of the vector represent the same "type" of capacity, so two elements with the same amount of capacity are equivalent.

Majorization is a standard way to compare the imbalance of distributions; see [82] for a general reference. Some of its properties are as follows. Assuming $\sum_{i=1}^{n} c_i = n$, majorization defines a partial order whose bottom $\bot = (1, \ldots, 1)$ is the homogeneous distribution, and whose top $\top = (n, 0, \ldots, 0)$ is the centralized distribution. Two other measures of heterogeneity are variance $\mathrm{var}(C) = \frac{1}{||C||} \sum_{i=1}^{n} (c_i - ||C||/n)^2$ and negative entropy $-H(C) = \sum_{i=1}^{n} c_i \log_2 c_i$. Although variance and negative entropy disagree on the ordering of vectors in general, majorization is consistent with both, in the sense that $C' \succeq C$ implies $\mathrm{var}(C') \geq \mathrm{var}(C)$ and $-H(C') \geq -H(C)$.

For our purposes, a **cost function** is a function $g : \mathcal{C} \times \mathcal{W} \to \mathbb{R}^+$, where $\mathcal{C} \subseteq \mathbb{R}^n$ is the set of legal node capacity vectors and $\mathcal{W}$ is arbitrary additional problem-specific information. Typically, $g(C, W)$ will represent the cost of the optimal solution to some combinatorial problem with node capacities C and workload W. However, one could also examine, for example, the cost of approximate solutions produced by a particular algorithm. We can now define our main metric.

Definition 1 *The* **price of heterogeneity** *(PoH) of a cost function $g : \mathcal{C} \times \mathcal{W} \to \mathbb{R}^+$ is*

$$\sup_{W, C, C' :\, C \preceq C'} \frac{g(C', W)}{g(C, W)},$$

where $W \in \mathcal{W}$ and $C, C' \in \mathcal{C}$.

A PoH of 5/4 would say that for any capacities C and $C' \succeq C$, distribution C' can handle any workload with cost at most 25% higher than C. That is, as heterogeneity increases, performance cannot get much worse.

The price of heterogeneity can be viewed as a generalization of Schur concavity. A function g is *Schur concave* when $C' \succeq C$ implies $g(C') \leq g(C)$. One could say that g is *α-approximately Schur concave* when $C' \succeq C$ implies $g(C') \leq \alpha \cdot g(C)$. Then $g(C, W)$ has PoH α if and only if $g(C, W)$ is α-approximately Schur concave in C for every W.

The price of heterogeneity naturally brings to mind the "price of homogeneity": the worst-case increase in cost as capacities become more *homogeneous*. It is easy to see that, for the cost functions considered in this chapter, increasing homogeneity can be quite harmful. In any of the scheduling problems, replacing machine speeds $(n, 0, \ldots, 0)$ with the more homogeneous speeds $(1, 1, \ldots, 1)$ results in a factor n slowdown when processing any single job. We therefore focus on the price of heterogeneity, but note that it would be interesting to find natural situations in which the price of homogeneity yields useful insight.

2.3 The Simulation Lemma

A natural way to show that the heterogeneous capacities C' are as good as the more homogeneous capacities C is to "simulate" C using C'. More specifically, we would assign C-nodes to C'-nodes according to some $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$, and show that each C'-node i can "simulate" the work previously performed by the subset of C-nodes $f^{-1}(i)$. This is a fairly restrictive technique which cannot capture the structure important to some cost functions (see especially Section 2.5). Nevertheless, we will see that the simulation technique is applicable to a number of important problems. To prepare for those results, in this section we use the simulation technique to produce convenient sufficient conditions to obtain a $O(1)$ PoH (Theorem 1).

For most natural cases, a prerequisite for the simulation technique to succeed is that the total capacity simulated by each C'-node i is not much more than its own capacity c'_i:

Definition 2 *For capacity vectors C and $C' \succeq C$, an* α-simulation of C by C' *is a function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $\sum_{j \in f^{-1}(i)} c_j \leq \alpha c'_i$, for all i.*

It is NP-complete to decide whether a 1-simulation exists (see Appendix A.1). The main result of this section is that a $(2 - 1/n)$-simulation always exists.

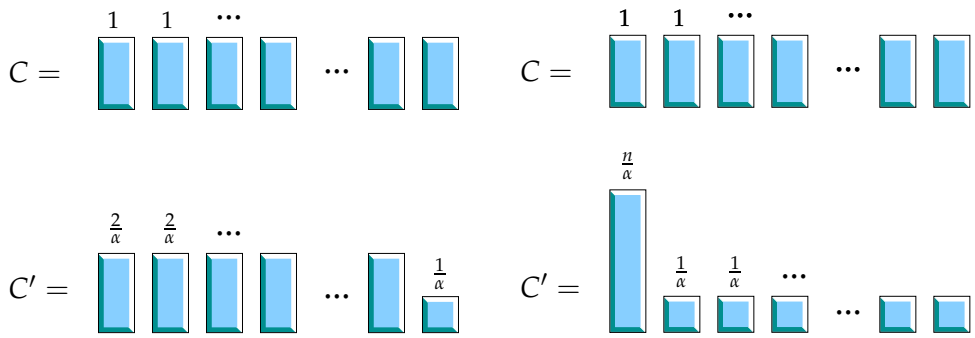


Figure 2.1: Two families of examples showing the tightness of the Simulation Lemma. Here $\alpha = 2 - 1/n$. In both examples, every assignment of C to C' gives some element of C' at least α times its capacity.

Lemma 1 ***(Simulation Lemma)*** *For any capacity distributions C and $C' \succeq C$, a $(2 - 1/n)$-simulation exists and can be found in time $O(n \log n)$.*

The bound is exactly tight, as exhibited in Figure 2.1. In the remainder of this section, we prove the lemma, and then use it to provide sufficient conditions for a cost function to have constant price of heterogeneity (Theorem 1). In later sections, we will see that a number of optimization problems satisfy those conditions.

Proof: Let $\alpha = 2 - \frac{1}{n}$. The following algorithm produces an α-simulation $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$. Begin by sorting the two capacity vectors in decreasing order. Maintain a vector of *available capacities* $A = (a_1, \ldots, a_n)$. Initially, $A = (0, \ldots, 0)$. For each $i = 1$ to n, perform the following steps:

1. Set $a_i \leftarrow c'_i$.
2. Let $j \in \{1, \ldots, i\}$ be such that $a_j \geq c_i/\alpha$.
3. Set $f(i) \leftarrow j$ and $a_j \leftarrow a_j - c_i/\alpha$.

The algorithm can be implemented in $O(n \log n)$ time by storing A in a heap and taking j to be the maximum element. It remains to be shown that (1) in each iteration, a suitable j satisfying $a_j \geq c_i/\alpha$ can be found, and (2) the resulting f is an α-simulation.

We show (1) first. After Step 1 of the ith iteration, the total capacity that has been added to A is $\sum_{k=1}^{i} c'_k$, and the total capacity that has been subtracted is $\sum_{k=1}^{i-1} c_k/\alpha$. So the

total capacity remaining in A after Step 1 of the ith iteration is

$$\begin{aligned}
\sum_{k=1}^{i} c'_k - \sum_{k=1}^{i-1} \frac{c_k}{\alpha} &= \frac{c_i}{\alpha} + \sum_{k=1}^{i} c'_k - \sum_{k=1}^{i} \frac{c_k}{\alpha} \\
&\geq \frac{c_i}{\alpha} + \left(1 - \frac{1}{\alpha}\right) \sum_{k=1}^{i} c_k \quad \text{(since } C' \succeq C\text{)} \\
&\geq \frac{c_i}{\alpha} + i \cdot \left(1 - \frac{1}{\alpha}\right) c_i \quad \text{(since } c_1 \geq \cdots \geq c_i\text{)} \\
&= i \cdot \left(\frac{c_i}{i\alpha} + \left(1 - \frac{1}{\alpha}\right) c_i\right).
\end{aligned}$$

Moreover, at step i there are $\leq i$ positive entries of A, so some entry must be $\geq \frac{c_i}{i\alpha} + \left(1 - \frac{1}{\alpha}\right) c_i$. Plugging in $\alpha = 2 - 1/n$ and noting that $i \leq n$ shows that this expression is at least c_i/α. Thus, a suitable j can be found.

We now show (2), *i.e.*, that $\sum_{i \in f^{-1}(j)} c_i \leq \alpha c'_j$ for each j. Note that a_j first became positive by setting $a_j = c'_j$. Each time we set $f(i) \leftarrow j$ for some i, the capacity assigned to entry j increased by c_i, and a_j decreased by c_i/α. Since $a_j \geq 0$ always, the total capacity assigned to j is $\leq \alpha c'_j$. ■

Theorem 1 *Suppose a cost function g satisfies the following properties:*

1. *$g(C, W)$ is nonincreasing in each component of C;*
2. *$g(C, W)$ is a symmetric function of the components of C;*
3. *$g(\frac{1}{2} \cdot C, W) \leq \beta \cdot g(C, W)$ for all C and W; and*
4. *$g(D, W) \leq g(C, W)$, where D is formed from C by replacing components i and j with $c_i + c_j$ and 0, respectively, for any C, W, i, and j.*

Then the price of heterogeneity of g is $\leq \beta$.

Proof: Let C and C' be capacity distributions such that $C' \succeq C$. We must show $g(C', W) \leq \beta \cdot g(C, W)$. Let f be a 2-simulation as given by the Simulation Lemma, in which, for each i, $2c'_i \geq \sum_{j \in f^{-1}(i)} c_j \stackrel{def}{=} e_i$. Let $E = (e_1, \ldots, e_n)$. We have

$$\begin{aligned}
g(C', W) &\leq \beta \cdot g(2C', W) && \text{(Property 3)} \\
&\leq \beta \cdot g(E, W) && \text{(Property 1 and } 2C' \geq E\text{)} \\
&\leq \beta \cdot g(C, W) && \text{(repeated application of Properties 2 and 4).}
\end{aligned}$$

■

2.4 Scheduling on Related Machines

We now apply the results of the previous section to the problem of *scheduling on related machines*. We are given a set J of jobs, each with a length $\ell(j)$, and an n-vector C of processor speeds. We must schedule the jobs on our n machines so that each machine is executing at most one job at any time. Machine i completes each job j in time $\ell(j)/c_i$, so if it is given jobs J_i, it can finish its jobs in time $t_i = \ell(J_i)/c_i$, where $\ell(J) := \sum_{j \in J} \ell(j)$.

There are many variants of this problem since there are many possible objective functions, *i.e.*, measures of the cost of a schedule. The most common is the *makespan*: the time until the last job (equivalently, processor) finishes. We begin by analyzing the price of heterogeneity of this variant of the problem (Section 2.4.1). We then generalize the result for other objective functions (Sections 2.4.2 and 2.4.3) before noting a complementary property of the distribution of job lengths (Section 2.4.4).

2.4.1 Minimum Makespan Scheduling

In this section we analyze the price of heterogeneity of the minimum makespan scheduling problem. More formally, we find the PoH of the cost function $g(C, J)$ defined as the minimum makespan of any schedule of jobs J on processors C.

Theorem 2 *The price of heterogeneity of minimum makespan scheduling is* $2 - 1/n$.

Our proof illustrates the basic technique we will use in later bounds on the PoH. For concreteness of exposition and to obtain a slightly tighter bound, we use the Simulation Lemma directly, rather than Theorem 1. Unlike our later results, in this case we provide matching lower and upper bounds. The lower bound transfers from that of the Simulation Lemma (Figure 2.1) because both the lemma and the makespan consider the maximum amount of work assigned to a machine.

The proof will use a simple but important fact:

Fact 1 *For any schedule of jobs on processors of speeds $c_1, \ldots, c_k$ ("parallel schedule"), there is a serial schedule of those jobs on a single processor of speed $c_1 + \cdots + c_k$ ("serial schedule") such that each job completes before or at the same time as it did in the parallel schedule.*

Proof: Schedule jobs on the single processor in order of their completion time in the parallel schedule, with ties broken arbitrarily. Consider any job j and suppose its completion

time in the parallel schedule is t. In the parallel schedule, the total length of all jobs completed by time t must be $\leq \sum_{i=1}^{k} t \cdot c_i$. The serial schedule executes those jobs before any others, so they complete by time $\leq \left(\sum_{i=1}^{k} t \cdot c_i\right) / (c_1 + \cdots + c_k) = t$. ■

This fact is related to Weighted Fair Queueing's simulation of General Processor Sharing [31, 91]. WFQ guarantees that each packet *approximately* meets its deadline in the idealized GPS model: it may be delayed only by the maximum time it takes to send a single packet [91]. From Fact 1 it can be seen that in the special case that all packets are queued at time 0, WFQ *exactly* meets its deadline or is sent early.

We are now ready prove the theorem.

Proof of Theorem 2: We begin with the upper bound. Given any machine speeds C and $C' \succeq C$, and any schedule of jobs J on machines C with makespan M, it is sufficient to produce a schedule of the jobs on the C'-machines with makespan $2M$.

Suppose jobs $J_k \subseteq J$ are scheduled on machine k in the C-schedule. Let $f : C \to C'$ be the mapping defined by the Simulation Lemma. For each k, schedule jobs J_k on C'-machine $f(k)$. Now let $F(i) := f^{-1}(i)$ be the set of C-machines mapped to C'-machine i, and let $s = \sum_{k \in F(i)} c_k$ be the total speed of these machines. By Fact 1, a machine of speed s could complete the jobs assigned to C'-machine i in time $\leq M$. By the Simulation Lemma, $c'_i \geq s/(2 - 1/n)$, so each C'-machine i completes its jobs in time $\leq (2 - 1/n)M$.

To show the lower bound, we can use either pair of capacity vectors in Figure 2.1, in both cases with n jobs of length 1. The reader can verify that $OPT(C, J) = 1$, but $OPT(C', J) \geq 2 - 1/n$. ■

2.4.2 General Objective Functions of Job Completion Times

Fact 1 is actually much stronger than was necessary to bound the makespan: it bounds the completion time of *each individual* job, not just the last. This property lets us analyze a large class of objective functions.

Let $h : \mathbb{R}^m \to \mathbb{R}^+$ be a function of the job completion times. We say h is *β-bounded* when $h(2\mathbf{t}) \leq \beta \cdot h(\mathbf{t})$ for all $\mathbf{t}$. Examples of 2-bounded objective functions sometimes used to evaluate the quality of a schedule are the maximum and mean job completion time and the L_p-norm of the job completion times, *i.e.*, $h(\mathbf{t}) = \left(\sum_{i=1}^{m} t_i^p\right)^{1/p}$, for $p \geq 1$. The squared completion time, $h(\mathbf{t}) = \sum_i t_i^2$, is 4-bounded. The objective function h may be asymmetric,

as is possible in the case of weighted mean job completion time, which for any weighting of the jobs is 2-bounded.

Corollary 1 *Suppose $h : \mathbb{R}^m \to \mathbb{R}^+$ is a nondecreasing, β-bounded function of the job completion times. Let $g(C, J)$ be the minimal value of h over all schedules of jobs J on machines C. Then g has PoH $\leq \beta$.*

Proof: We apply Theorem 1. Property 1 results from the fact that job completion times are inversely proportional to processor speed and h is nondecreasing. Property 2 is true since optimal job completion times do not depend on the order in which the machines are listed. Property 3 follows from β-boundedness of h, and Property 4 follows from Fact 1 and the fact that h is nondecreasing. ■

2.4.3 General Objective Functions of Machine Completion Times

We may similarly consider bounded functions h of the *machine* completion times. Here, following Theorem 1, we require that h is a symmetric function of its arguments. In any case, since the PoH compares instances with the same set of jobs but a different set of machines, giving machines identities makes less sense than giving jobs identities as the previous section's asymmetry allowed.

Corollary 2 *Suppose $h : \mathbb{R}^n \to \mathbb{R}^+$ is a nondecreasing symmetric β-bounded function of the machine completion times. Let $g(C, J)$ be the minimal value of h over all schedules of jobs J on machines C. Then g has PoH $\leq \beta$.*

Proof: Again applying Theorem 1, Property 1 is satisfied as in Corollary 1. Properties 2 and 3 follow from symmetry and β-boundedness, respectively, of h. Finally, Fact 1 shows that when merging machines, the completion time of the last *job* does not increase, so the merged *machine's* completion time must be at most that of one of the machines it replaced. This combined with the fact that h is nondecreasing satisfies Property 4. ■

An interesting open problem would be to obtain tighter bounds for the L_p-norm of machine completion times as a function of p. For the L_1-norm in particular, the PoH is 1 since the optimal assignment places all tasks on the fastest machine, and that machine is always at least as fast in C' as in C.

2.4.4 A Complementary Result

We observe that the Simulation Lemma can also be used to describe the effect of heterogeneity of job length distributions, in a way closely analogous to the price of heterogeneity. Theorem 2 showed that as capacities C become more heterogeneous, the minimum makespan $OPT(C, J)$ can't become much worse, for any fixed job lengths J—thus confirming, within a factor of 2, the intuition that more heterogeneous capacities are better. Similarly, in this section we show that as *the job lengths* become more *homogeneous*, the makespan can't become much worse, for any fixed node capacities—thus confirming, within a factor of 2, the intuition that more homogeneous job lengths are better.

Theorem 3 *Let J and J' be vectors of m job lengths with $J' \succeq J$. For any C, $OPT(C, J) \leq (2 - 1/m) \cdot OPT(C, J')$.*

Proof: Let $f : J \to J'$ be a $(2 - 1/m)$-simulation, which exists by the Simulation Lemma. Then if J'-job j is executed on machine i in the optimal schedule, we place the J-jobs $f^{-1}(j)$ on machine i. Since f is a $(2 - 1/m)$-simulation, this increases total length of jobs placed on i, and hence the completion time of any machine, by at most a factor $2 - 1/m$. ■

Examples analogous to those of Figure 2.1 show a matching lower bound for the above theorem: that is, there exist vectors of m jobs J and $J' \succeq J$ and node capacities C for which $OPT(C, J) = (2 - 1/m) \cdot OPT(C, J')$.

2.5 Precedence Constrained Scheduling

In the precedence constrained scheduling (PCS) problem [43], we are given node capacities C, a set J of jobs, a length $\ell(j)$ for each $j \in J$, and a partial order $\prec_J$ on J. We must schedule the jobs on the nodes as before, with the added constraint that if $j_1 \prec_J j_2$ then job j_1 must complete by the time j_2 begins. The cost is the minimum makespan of such a schedule.

The key difficulty in transferring the simulation technique to PCS lies in adapting Fact 1. When merging the work of two machines of capacities c_1 and c_2 into one machine of capacity $c_1 + c_2$, it is no longer sufficient to show that the *completion time* of each job does not increase. To satisfy precedence constraints without a global modification of the

schedule, one would have to devise a schedule for which the *start time* of each job does not decrease.

In fact, we show that the direct application of the simulation technique cannot possibly succeed: there are instances for which having each C'-machine perform the work of some subset of the C-machines must result in at least a factor $n/4$ inflation of the makespan **(Theorem 4)**. This is perhaps surprisingly bad, since one can obtain a factor n inflation by putting all the jobs on the single fastest machine, completely ignoring the other $n-1$ machines! Intuitively, the simulation technique performs poorly since mapping several C-machines onto one C'-machine reduces parallelism. The result is that long jobs will occasionally interrupt a sequence of short jobs, which in turn causes idle time on all machines whose jobs depended on those short jobs.

However, the simulation technique can be applied in an LP relaxation of PCS [24], intuitively because that LP lets a single machine run multiple jobs in parallel. This produces an upper bound on PCS's PoH of $O(\log n)$ in the general case **(Corollary 3)** and $O(1)$ when there are only a constant number of distinct machine speeds **(Corollary 4)**. If job lengths vary by at most a constant factor, then we can show an analog of Fact 1 and obtain a $O(1)$ PoH **(Corollary 5)**. Finally, we show that the general case of PCS has a property which is necessary, but not sufficient, for $O(1)$ PoH: the homogeneous distribution is within a constant factor of the worst case **(Corollary 6)**.

2.5.1 A Lower Bound for the Simulation Technique

The following theorem shows that there are PCS workloads for which having each C'-machine perform the work of some subset of the C-machines must result in a factor $n/4$ inflation of the optimal makespan.

Theorem 4 *There exist capacity vectors C and $C' \succeq C$ and an instance $(C, J, \ell, \prec_J)$ of precedence constrained scheduling with an optimal schedule of makespan OPT which maps jobs to machines according to $h : J \to \{1, \ldots, n\}$, such that for any $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$, scheduling instance $(C', J, \ell, \prec_J)$ by placing job i on machine $f(h(i))$ has makespan $\geq \frac{1-o(1)}{4} \cdot n \cdot OPT$.*

Proof: We take $C = (1, \ldots, 1)$ and $C' = (2, \ldots 2, 0, \ldots, 0)$, i.e. $n/2$ machines of speed 2. The problem instance is as follows. We have n groups of jobs, indexed 1 through n. Group i consists of k^{n-i} jobs of length k^i. We choose a convenient k later. The optimal C-schedule

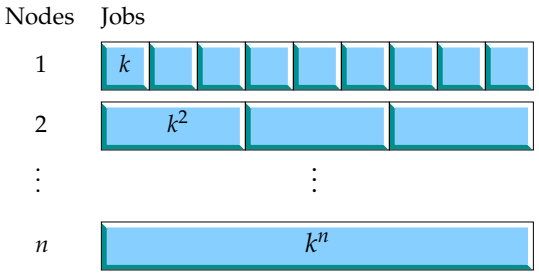


Figure 2.2: An instance of PCS on which the simulation technique fails.

places group i on machine i, as shown in Figure 2.2. The set of precedence constraints is the maximum set for which the above schedule is valid. That is, we have a constraint $j_1 \rightarrow j_2$ iff job j_1 completes by the time job j_2 starts. Note that the resulting makespan on the C-machines is k^n, and this is optimal since no machine is idle until all jobs are complete.

Now suppose that we map the C-machines to C'-machines according to some $f : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$, and we restrict ourselves to executing the group-i jobs on C'-machine $f(i)$ as in the theorem statement. We seek to lower-bound the makespan of any such schedule.

Define a group as *obstructing* if it is assigned by f to a machine which is also assigned a group of smaller jobs. Let $g_1, \ldots, g_m$ be the indexes of the obstructing groups, with $g_1 \leq \cdots \leq g_m$. Note $m \geq n/2$ since there are n groups and only $n/2$ machines with positive capacity. Let $t(g_i)$ be the time spent executing group g_i during which no job from any larger obstructing group is being executed. Note that the makespan of the schedule is $\geq \sum_{i=1}^{m} t(g_i)$. We now lower-bound each $t(g_i)$. First we need a key fact:

Fact 2 *While any job from an obstructing group g_i is executing, at most $2k^{g_i - j - 1}$ jobs in any smaller-indexed group $j < g_i$ can execute on any other machine.*

Proof: Let x be a g_i job, and let D be the set of group-j jobs executed on any machine during x. We wish to upper-bound $|D|$.

Since g_i is obstructing, there is some smaller group on the same machine. Let Y be the set of those smaller jobs. To handle boundary cases cleanly, augment Y with two "marker jobs" γ_1 and γ_2, both of zero length, with γ_1 at the beginning of the chain of dependencies in Y and γ_2 at the end. We may assume w.l.o.g. that γ_1 is the first job executed on its machine and γ_2 is the last.

Since a machine can only execute one job at a time, there exist two jobs $y_1, y_2 \in Y$ such that y_1's immediate successor is y_2, y_1 is executed before x, and x is executed before y_2. Thus, since y_1 has completed when x starts, D cannot include any jobs on which y_1 depends. Similarly, since y_2 has not yet completed, D cannot include any jobs which depend on y_2. Thus, D includes only group-j jobs that, according to the precedence constraints, can execute concurrently with y_1 or y_2. The total length of such jobs is at most the length of y_1 plus the length of y_2, which is $\leq 2k^{g_i-1}$. Since each group-j job has length k^j, we have $|D| \leq 2k^{g_i-1}/k^j = 2k^{g_i-j-1}$, as desired. ■

Now consider some obstructing group g_j. By Fact 2, the number of g_j-jobs executed during a job of length k^{g_i} is $\leq 2k^{g_i-g_j-1}$. Since there are k^{n-g_i} jobs of length k^{g_i}, the total number of g_j-jobs executed during longer obstructing jobs is

$$\sum_{i=j+1}^{m} k^{n-g_i} \cdot 2k^{g_i-g_j-1} \;\leq\; 2\sum_{i=j+1}^{n} k^{n-g_j-1} \;\leq\; 2n \cdot k^{n-g_j-1}.$$

Since there are k^{n-g_j} group-g_j jobs to begin with, the number of group-g_j jobs not executed during longer obstructing jobs is $\geq k^{n-g_j} - 2n \cdot k^{n-g_j-1} = (1-o(1))k^{n-g_j}$ for $k = n^2$ (recall k is arbitrary). The time per job is $k^{g_j}/2$ since all C'-machines have speed 2. Thus, we have that $t(g_j) \geq (1-o(1))k^{n-g_j} \cdot k^{g_j}/2 = \frac{1}{2}(1-o(1))k^n = \frac{1}{2}(1-o(1)) \cdot OPT$.

Since this is true for all obstructing groups, we have that the makespan of the C'-schedule is at least $\sum_{j=1}^{m} t(g_j) \geq m \cdot \frac{1}{2}(1-o(1)) \cdot OPT$. As noted above, $m \geq n/2$, which proves the theorem. ■

2.5.2 Upper Bounds

We begin with an upper bound for the general case of PCS.

Corollary 3 *The PoH of precedence constrained scheduling is $O(\log n)$.*

Proof sketch: Chudak and Shmoys [24] gave a linear programming relaxation of PCS which formed the basis of their $O(\log n)$-approximation algorithm, which is the best known. This LP relaxation does not include the constraint that a machine executes at most one job at a time. It is thus easy for one fast machine to simulate the work of several slow machines, so we can apply the Simulation Lemma to show that the optimal solutions to the LP have $O(1)$ PoH. By the main result of [24], the optimal solution to PCS is at most $O(\log n)$ times the LP's solution. ■

The full proof appears in Appendix A.2. We also obtain a second corollary based on the results of [24]:

Corollary 4 *Restricted to instances with a constant number of distinct machine speeds, PCS has PoH $O(1)$.*

Proof: Follows from the result of [24] that the optimal values of the LP relaxation are within $O(1)$ of the true optimum when there are $O(1)$ distinct machine speeds. ■

The following corollary obtains a constant PoH for a special case of PCS.

Corollary 5 *The PoH of precedence constrained scheduling with unit-size jobs is ≤ 16.*

Proof: Consider any capacities C, $C' \succeq C$, and jobs J, and suppose the best schedule on the C-machines executes job j on machine $m(j)$ during $[t(j),\ t(j) + 1/c_{m(j)})$. It is sufficient to show a C' schedule such that each job is executed within $[16 \cdot t(j),\ 16 \cdot (t(j) + 1/c_{m(j)})]$. We first modify the schedule to make it more convenient: round the C-machine speeds down to the nearest power of 2, and execute each job j at a time which is a multiple of $1/c_{m(j)}$. Each of these modifications at most doubles the length of the schedule. Let C^* and $t^*(\cdot)$ be the resulting capacities and execution times.

Now let f be given by the Simulation Lemma. Consider the machines $f^{-1}(i)$ for some i. First, we merge each pair of machines $m_1, m_2 \in f^{-1}(i)$ for which $c^*_{m_1} = c^*_{m_2}$ by replacing them with a machine of capacity $2c^*_{m_1}$. We revise the execution time of each job j as $t^*(j) = t^*(j)$ if $m(j) = m_1$, and $t^*(j) = t^*(j) + \frac{1}{2} \cdot 1/c^*_{m(1)}$ if $m(j) = m_2$. Completion times do not increase since the machine capacity has doubled, and jobs do not overlap since each $t^*(j)$ was a multiple of $1/c_{m(j)}$. Iterating this merging of the machines in $f^{-1}(i)$, we are left with k machines $m_1, \ldots, m_k$ of unique power-of-two capacities $2^1, \ldots, 2^k$ (some may be missing).

We can now schedule these machines' jobs on a single machine m of capacity 2^{k+1} without changing the range of time in which each job is executed, as follows. Break time into slots of length $1/2^{k+1}$, the length necessary to process one job on machine m. For each job j_1 on machine m_k, there are two available slots within its scheduled time $[t^*(j_1),\ t^*(j_1) + 1/2^k]$. Place each job in one of these slots arbitrarily. For each job j_2 on machine m_{k-1}, there still remain two available slots within the larger time $[t^*(j_2),\ t^*(j_2) + 1/2^{k-1}]$, so we can recursively schedule the jobs on machines $m_1, \ldots, m_{k-1}$ in the same manner.

The Simulation Lemma guarantees $c'_i \geq \frac{1}{2}\sum_{\ell \in f^{-1}(i)} c_{m_\ell} \geq \frac{1}{2} \cdot 2^k$. We used a machine of speed 2^{k+1}, so this increases the makespan by a factor of 4. Combining this with our earlier modification of the schedule, the corollary follows. ■

Our final upper bound for PCS shows that the homogeneous distribution is within a constant factor of the worst case.

Corollary 6 *Let $OPT(C', W)$ be the optimal makespan of an instance W of PCS with capacities C'. Then $OPT(C', W) \leq 4 \cdot OPT(\perp, W)$ for any C', where $\perp = (1, \ldots, 1)$.*

Proof: Produce distribution D from C' by setting to 0 any element i with $c'_i \leq \frac{1}{2}$. We will show $OPT(C', W) \leq OPT(D, W) \leq 4 \cdot OPT(\perp, W)$. The first inequality is trivial since $c'_i \geq d_i \forall i$. For the second inequality, schedule the jobs on D using Graham's classic list scheduling algorithm [49] as follows: at time 0, we iteratively place any job whose precedence constraints have been satisfied onto any idle machine of positive capacity, until no such jobs remain or all machines are busy. Whenever any job finishes, we repeat that greedy placement procedure. Let S be the resulting schedule.

To bound the length of S, we need only slightly adapt the standard bounds used for Graham's algorithm. Note that there exists a precedence-constrained chain of jobs $j_1 \to \cdots \to j_k$ such that at any time during S, either (1) all machines are busy, or (2) some job j_i in the chain is executing. Let L_1 and L_2, respectively, be the total length of time spent in each of these states. To bound L_1, we use the fact that the amount of time that all machines can be busy is at most the total length of all jobs divided by the total machine speed, which for D is $\geq \frac{n}{2}$; so $L_1 \leq \frac{2}{n} \cdot \sum_{j \in J} \ell(j)$. But $OPT(\perp, W) \geq \frac{1}{n}\sum_{j \in J} \ell(j)$ since the total capacity in $\perp$ is n, and thus $L_1 \leq 2 \cdot OPT(\perp, W)$. To bound L_2, note that every job in the chain $j_1 \to \cdots \to j_k$ is executed at speed $\geq \frac{1}{2}$, while $OPT(\perp, W)$ must also execute this chain serially using machines of speed 1. Thus, $L_2 \leq 2 \cdot OPT(\perp, W)$ and $OPT(D, W) \leq L_1 + L_2 \leq 4 \cdot OPT(\perp, W)$. ■

2.6 Resource Constrained Scheduling

Resource constrained scheduling [43] (RCS) generalizes minimum makespan scheduling by adding some number k of *resources* specified by the problem instance, each of which is shared across all the processors. Each job j is then associated with not only a length but

also a resource requirement $r_i(j) \in [0,1]$ for each resource $i \in \{1,\ldots,k\}$. We require that $\sum_{j \in E_t} r_i(j) \leq 1$ for all times t and each resource i, where E_t denotes the set of jobs executing at time t. A resource requirement could represent, for example, a required fraction of the available bandwidth on a shared network interface, exclusive write access to a particular lock ($r_i(j) = 1$), or read access to the lock ($r_i(j) = 1/|J|$, where $|J|$ is the number of jobs).

We next upper-bound the PoH of RCS by that of PCS. In fact, we prove a somewhat stronger claim:

Theorem 5 *The PoH of scheduling with both resource and precedence constraints is at most the PoH of PCS.*

Proof: Suppose the PoH of PCS is $\alpha = \alpha(n)$. Consider any C, $C' \succeq C$, and workload instance W_1 which may include both precedence and resource constraints. Let S be an optimal schedule of W_1 on the C-machines. Construct workload W_2 from W_1 by removing all resource constraints, and adding every possible precedence constraint which maintains feasibility of S: that is, we add a precedence constraint $j_1 \rightarrow j_2$ for any jobs j_1, j_2 such that j_1 completes by the time j_2 begins in S.

To prove the theorem we will show that $OPT(C', W_1) \leq OPT(C', W_2) \leq \alpha \cdot OPT(C, W_2) \leq \alpha \cdot OPT(C, W_1)$, where $OPT(C, W)$ denotes the minimum makespan of a schedule with workload W on machines C. The second inequality follows from the fact that the PoH of PCS is α. The third follows from the fact that S, an optimal schedule for W_1, is also feasible for W_2.

For the first inequality, it is sufficient that any feasible schedule of W_2 is also feasible for W_1: that is, the added precedence constraints are at least as restrictive as the removed resource constraints. For any feasible schedule of W_2, let E_t be the set of jobs running at any time t. We will show $\sum_{j \in E_t} r_i(j) \leq 1$, so that the resource constraints of W_1 are satisfied at all times. It is sufficient to show that the jobs E_t are executed concurrently at some time during schedule S. Suppose j_1 is the job in E_t which begins last in S, at time t_1; and j_2 is the job in E_t which finishes first in S, at time t_2. Clearly, $t_1 < t_2$ or else there would be a precedence constraint $j_2 \rightarrow j_1$ in W_2, which there cannot be since all jobs in E_t are run concurrently. So at time $(t_1 + t_2)/2$ in S, all jobs in E_t have started, and none have finished, so all are running. ■

2.7 Scheduling With Release Times

The last scheduling problem we consider is *scheduling with release times*. We must produce an offline schedule of jobs J on machines C as in scheduling on related machines, except that we are also given for each job $j \in J$ a *release time* $r(j)$ before which j may not be executed. Our cost function $g(C, (J, r))$ is the minimal *total response time* of any schedule of jobs J with release times r on machines C. We define total response time as the sum over all jobs j of the time j spends in the system normalized by its length: $\frac{t(j)+\ell(j)/c-r(j)}{\ell(j)}$, where $t(j)$ is the start time of job j and c is the capacity of the machine on which it is run.

Similar release time constraints appear in Garey and Johnson [43], but we borrow the response time objective from queuing systems such as [128], in which it is known that decreasing parallelism—i.e., increasing heterogeneity—can significantly increase response time (see discussion in Section 2.10).

It is easy to observe that even moving from two machines to one can be quite disadvantageous. As in PCS, reduced parallelism causes short jobs to be held up by long jobs.

Theorem 6 *The price of heterogeneity of scheduling with release times with job sizes in* $[1, k]$ *is* $\Omega(k)$.

Proof sketch: Let $C = (1, 1)$ and $C' = (2, 0)$. Suppose J consists of mk jobs of size 1 arriving at times $0, 1, \ldots, mk - 1$ and m jobs of size k arriving at times $0, k, 2k, \ldots, mk - k$. These can be scheduled as they arrive on the C-machines, for a total response time of $\Theta(mk)$. Now consider scheduling these jobs on the single C'-machine of nonzero capacity. Either $\Theta(m)$ long jobs are delayed for time $\Theta(km)$ until all short jobs are complete, or each of $\Theta(m)$ long jobs delays $\Theta(k)$ short jobs for time $\Theta(k)$ each. Picking $m = k^2$, in either case total response time is $\Omega(k^4)$, compared with $\Theta(k^3)$ for the C-machines. ■

The full proof appears in Appendix A.3.

2.8 Network Construction

In designing a communication network, a typical goal is to minimize the number of hops between any two nodes, subject to bounds on the maximum number of links

incident to each node. For example, in placing physical links between nodes of a supercomputer or cluster, each node may have a limited number of network ports. In an overlay multicast network, each link may involve forwarding a stream of multicast data, so the degree of a node would be limited by its available bandwidth. Constructing such networks with low maximum latency between nodes involves a classic tradeoff [26] between degree and diameter.

In this section we will study how the optimal diameter changes as the degree bounds become more heterogeneous. Note that in the following formulation, we do not make use of the "workload" parameter of the cost function. Also, in the proof, we do not apply the Simulation Lemma because the capacities specify hard constraints which cannot be violated (Condition 3 of Theorem 1 would not be satisfied).

Definition 3 *(Minimum Graph Diameter) Given positive integer degree bounds $C = (c_1, \ldots, c_n)$, $MinDiam(C)$ is the minimum diameter of a graph G in which* $\deg(i) \leq c_i$ *for all nodes i.*

Theorem 7 *The price of heterogeneity of MinDiam is* ≤ 2.

Proof: We will show $MinDiam(C') \leq 2 \cdot TREE(C') \leq 2 \cdot TREE(C) \leq 2 \cdot MinDiam(C)$, where $TREE(\cdot)$ is the minimal height of a rooted tree with given degree bounds. The first inequality follows from the fact that a tree's diameter is at most twice its height, and the third follows from the fact that the shortest-path tree rooted at any node of a graph has height at most the graph's diameter.

For the second inequality, we will exhibit a sequence of trees $T_1, \ldots, T_k$ and a corresponding sequence of degree bounds $C = C^1, C^2, \ldots, C^k = C'$ such that each tree T_i satisfies degree bounds C^i and the height of the trees do not increase as we proceed through the sequence. Moreover, we let T_1 be the optimal tree for degree bounds C, allowing us to conclude that $TREE(C') \leq TREE(C)$.

In each step, we produce C^{i+1} from C^i by transferring one unit of capacity (a unit bound on the degree) from some node j to ℓ, where $c^i_\ell \geq c^i_j$; a standard fact is that such a sequence of transfers always exists when $C' \succeq C$ [82]. To produce tree T_{i+1} from tree T_i, we first test whether $\text{level}(\ell) \leq \text{level}(j)$, where $\text{level}(\cdot)$ denotes distance from the root. If this is not true, we transfer $c^i_\ell - c^i_j$ child subtrees from node ℓ to node j (or as many as exist if ℓ has fewer subtrees) and swap the labeling of the nodes, so that we may assume w.l.o.g. that $\text{level}(\ell) \leq \text{level}(j)$. Finally, we transfer one child subtree from node j to node

ℓ to match the degree bounds C^{i+1} (if j has a subtree). Since these operations all involve moving subtrees closer to the root, the height can only decrease. ■

2.9 A Worst Case for Testing

In this section, we discuss how the price of heterogeneity can provide a worst case for testing, using load balancers for distributed hash tables (DHTs) as an example.

Most DHTs have been designed without knowledge of their eventual deployment environment, which might be a homogeneous cluster, a worldwide managed system like PlanetLab [93] (whose nodes vary in memory, disk space, and processor speed by factors of 4 to 14[1]), or a peer-to-peer system like Gnutella (whose nodes vary in bottleneck bandwidth by at least three orders of magnitude [107]). With such a wide range of target deployments, it may be valuable to test under a capacity distribution which would bound the system's performance in *any* deployment scenario. If we have a cost function $g(C, W)$ which models the system well, and if g has PoH α, then the system's cost under homogeneous capacities is within a factor α of the worst case regardless of the workload W, assuming n and the total capacity are fixed. We next argue that in the case of DHT load balancing, it is possible to produce such a cost function g which is a reasonable model of the system.

Several proposed DHT load balancers [44, 71] as well as Amazon's Dynamo storage system [30] assign ownership of objects stored in the system by first partitioning the objects among *virtual nodes*, and then placing virtual nodes on physical nodes. Each virtual node has an associated load, such as the rate of incoming requests for objects stored on it. The goal is to assign virtual nodes to physical nodes in a load-balanced way.

More specifically, suppose we desire to minimize the mean latency experienced by users of the system. Further assume that the latency experienced by a user connected to physical node i is u_i/c_i, where u_i is the total number of users connected to i among all of its virtual servers. If virtual server j has $\ell(j)$ users, and the set of virtual servers J_i is assigned to physical node i, we can write the users' total latency by summing over physical nodes as $g(C, J) = \sum_i \ell(J_i)^2/c_i$, where $\ell(J_i) = \sum_{j \in J_i} \ell(j) = u_i$. The following corollary implies that

[1] As of February 16, 2005, CoMon [92] reported memory between 0.49 and 1.98 GB and disk size between 32.7 and 264.7 GB among PlanetLab nodes. By September 29, 2006, these ranges had increased, with 0.49-3.78 GB of memory, 25.5-363 GB of disk, and CPU speeds ranging from 0.7 to 3.6 GHz. Data was unavailable for some nodes.

if the DHT load balancer finds assignments of virtual to physical nodes that are within a factor α of optimal, then mean latency will be within a factor 2α of the mean latency under homogeneous capacities, for any pattern of load on the virtual servers.

Corollary 7 *The price of heterogeneity of $g(C, J)$ is ≤ 2.*

Proof: Applying Theorem 1, Properties 1, 2, and 3 with $\beta = 2$ are immediate. Property 4 follows since for any $x, y, c_1, c_2 > 0$,

$$\frac{(x+y)^2}{c_1+c_2} \quad \leq \quad \frac{x^2}{c_1} + \frac{y^2}{c_2},$$

which can be shown by several lines of algebra. ■

2.10 Related Work

To the best of our knowledge, the price of heterogeneity is the first proposed model which can broadly characterize the effect of heterogeneity. However, some particular systems have been studied, which we describe next.

In many systems, it has been recognized that a heterogeneous capacity distribution is significantly preferable to a homogeneous one. For example, heterogeneity in the participating nodes' bandwidth constraints can reduce route lengths in distributed hash tables (DHTs) [48, 96] and in unstructured peer-to-peer file sharing systems [23], and can improve load balance in DHTs [44]. In supercomputing, designs using a few fast processors and many slower processors have been evaluated against homogeneous systems [5,6]. These studies generally look at specific capacity and workload distributions. The price of heterogeneity is complementary since we examine the worst case over all capacity distributions and workloads.

Closer to our model, Yang and de Veciana [131] studied a branching process model of a BitTorrent-like content distribution system in its *transient* phase, such as during the arrival of a flash crowd. The analysis showed that expected service capacity increases as the distribution of node bandwidth becomes more heterogeneous, in the sense of increasing convex orderings (which generalize majorization to random variables).

As mentioned in the introduction, an important special case of our model is when capacities are restricted so that there are m nodes of capacity n/m and $n - m$ of capacity 0.

The price of heterogeneity upper-bounds the increase in cost as m decreases. In queueing theory, a well known result is that among M/M/m queues (m servers of speed n/m with exponential job service times), $m = 1$ is optimal [112]. However, for various other job service time distributions, mean response time may be minimized when $m > 1$ (see [128] and the references therein). Intuitively, this is because parallelism keeps many small jobs from being held up by one big job. This fact corresponds to the super-constant price of heterogeneity of scheduling with release times (Section 2.7).

2.11 Summary

In this chapter we introduced the price of heterogeneity, which quanitifies the worst-case increase in cost as heterogeneity increases. We gave constant or logarithmic bounds on the price of heterogeneity of several well-known job scheduling and graph construction problems, indicating that in many cases, increasing heterogeneity can never be much of a disadvantage. These results provide the quantitatively justified intuition and theoretical framework that will inform subsequent chapters of this thesis.

We have left a number of directions for future research. First, our bounds could be tightened; resolving the question of whether precedence constrained scheduling has constant price of heterogeneity is of particular interest.

Second, one could analyze other cost functions, such as scheduling with random (rather than adversarial) workloads, schedules obtained by heuristics (rather than the optimal schedules), or the Nash equilibria of network congestion and load balancing games [66, 119]. Note that our results, which bound the PoH of optima, can yield bounds for the latter two items. For example, if a game has price of anarchy α and its social optima have price of heterogeneity β, then the Nash equilibria have price of heterogeneity $\leq \alpha\beta$. However, it may be interesting to analyze the PoH of common heuristics which do not give good approximation ratios. Relatedly, Suri et al. [119] have asked whether the price of anarchy itself decreases when machine speeds in their load balancing game become heterogeneous. Our framework may have relevance in answering that question.

A third direction is to broaden our model. Extending the notion of heterogeneity to allow nodes to have multiple kinds of capacity, or in general more than one attribute, may broaden its applicability.

Chapter 3

Heterogeneity and Load Balance in Distributed Hash Tables

The first two chapters of this thesis demonstrated that significant heterogeneity is prevalent in modern distributed systems, and that heterogeneity provably cannot be a significant disadvantage in broad classes of systems. But this analysis assumed systems which adapt to their heterogeneous environments; designing such a system can be non-trivial. It is typically easier to assume that all participants have equal roles, rather than automatically adapting to arbitrary heterogeneous environments. The remaining chapters of this thesis will develop techniques for adapting to and taking advantage of heterogeneity.

We begin with the design of hetergeneity-aware distributed hash tables (DHTs), which are highly scalable distributed storage systems. Early DHTs were designed primarily for nodes with equal capabilities and had not been evaluated in heterogeneous scenarios. Additionally, in both the homogeneous and heterogeneous cases, previous methods of balancing load in DHTs incurred a high overhead either in terms of routing state or in terms of load movement generated by nodes arriving or departing the system.

This chapter introduces a new protocol, called Y_0, which applies to the Chord DHT a general technique that we develop called Low Cost Virtual Server Selection. Y_0 balances load with minimal overhead under the typical assumption that the load is uniformly distributed in the identifier space. In particular, we prove that Y_0 can achieve near-optimal load balancing, while moving little load to maintain the balance and increasing the size of

the routing tables by at most a constant factor. Simulations based on real-world and synthetic capacity distributions indicate that Y_0 reduces the ratio of the maximum load to the mean load on a node from Chord's $O(\log n)$ to less than 3.6 without increasing the number of links that a node needs to maintain.

In addition, we study the effect of heterogeneity on both DHTs, demonstrating significantly reduced average route length as node capacities become increasingly heterogeneous. However, Y_0 better adapts the topology of its overlay network to hetergeneous environments. For a real-word distribution of node capacities with $n = 32,768$ nodes, Chord's mean route length is 22% shorter than in a homogeneous system, while Y_0's is 33% shorter. The asymptotic improvements (i.e., as $n \to \infty$) are empirically 23% and 55% for Chord and Y_0, respectively.

3.1 Introduction

The distributed hash table (DHT) is a flexible and general infrastructure that can support a large variety of applications including file sharing [27, 83], storage systems [67], query processing [57], name services [123], and communication services [20, 113, 134]. A DHT manages a global identifier (ID) space that is partitioned among n nodes organized in an overlay network. To partition the space, each node is given a unique ID x and owns the set of IDs that are "closest" to x. Each object is given an ID, and the DHT stores an object at the node which owns the object's ID. To locate the owner of a given ID, a DHT typically implements a greedy lookup protocol that contacts $O(\log n)$ other nodes, and requires each node to maintain a routing table of size $O(\log n)$.

One central challenge in DHT design is how to balance load across the nodes in the system. Even in the case of a homogeneous system where all nodes have the same capacity, DHTs can exhibit an $O(\log n)$ imbalance factor [114], meaning that some nodes own $O(\log n)$ times the average share of the ID space. Since most DHTs treat all participating nodes equally, the imbalance can significantly increase as the heterogeneity of the system increases.

Two classes of solutions have been proposed so far to address this challenge. Solutions in the first class use the concept of **virtual servers** [27, 63]. Each physical node instantiates one or more virtual servers with random IDs that act as peers in the DHT. In the case of a homogeneous system, maintaining $\Theta(\log n)$ virtual servers per physical node

reduces the load imbalance to a constant factor. To handle heterogeneity, each node picks a number of virtual servers proportional to its capacity. Unfortunately, virtual servers incur a significant cost: a node with k virtual servers must maintain k sets of overlay links. Typically $k = \Theta(\log n)$, which leads to an asymptotic increase in overhead.

The second class of solutions uses just a single ID per node [65, 77, 87]. However, all such solutions must reassign IDs to maintain the load balance as nodes arrive and depart the system [87]. This can result in a high overhead because it involves transferring objects and updating overlay links. In addition, none of these solutions handles heterogeneity directly, although they could be combined with the virtual server technique.

In this chapter, we present a simple DHT protocol, called Y_0, that addresses the above drawbacks. Y_0 is based on the concept of virtual servers, but with a twist: instead of picking k virtual servers with random IDs, a node clusters those IDs in a random fraction $\Theta(k/n)$ of the ID space. This allows the node to share a single set of overlay links among all k virtual servers. As a result, we can show that the number of links per physical node is still $\Theta(\log n)$, even with $\Theta(\log n)$ virtual servers per physical node.

In addition, we show that heterogeneity, rather than being an issue, can be an asset. Higher-capacity nodes have a denser set of overlay links and lower-capacity nodes are less involved in routing, which results in reduced route length compared to the homogeneous case. While both Chord and Y_0 see improvement, Y_0's is more significant because its placement of virtual servers provides more control over the topology.

Like most previous DHT work, we operate under the uniform load assumption: the load of each node is proportional to the size of the ID space it owns. This is reasonable when all objects generate similar load (*e.g.*, have the same size), the object IDs are randomly chosen (*e.g.*, are computed as a hash of the object's content), and the number of objects is large compared to the number of nodes (*e.g.*, $\Omega(n \log n)$). Alternately, we can unconditionally balance the *expected* load over uniform-random choices of object IDs.

Our main contributions are the following:

- We introduce a heterogeneity-aware ID selection algorithm for ring-based DHTs, Low-Cost Virtual Server Selection (LC-VSS). We prove that LC-VSS can balance the ID space partitioning within a factor $(1 + \varepsilon)$ of optimal for any $\varepsilon > 0$, and that while the system size and average capacity remain relatively stable, the amount of load movement to maintain that balance is nearly optimal.

- We prove that LC-VSS can be used with arbitrary overlay topologies while increasing route length by at most an additive constant and outdegree by at most a constant factor, even with $\Theta(\log n)$ virtual servers. Furthermore, our construction provides some flexibility in neighbor selection, even if the underlying topology lacks it.
- We apply LC-VSS to Chord and extensively evaluate the resulting protocol, called Y_0. Simulations in various capacity distributions indicate that Y_0 ensures that all nodes have less than 3.6 times their fair share of the ID space with no more overlay links than in Chord with a single virtual server. Furthermore, we show that heterogeneity decreases route length in Chord and more significantly in Y_0, with Y_0's route lengths roughly 33% shorter in a real-world distribution than in the homogeneous case for $n = 32,768$, or 55% shorter asymptotically.

The rest of this chapter is organized as follows. Section 3.2 discusses our model, the ID selection problem, and the technique of virtual servers. Section 3.3 introduces LC-VSS and its application to Chord to produce the Y_0 DHT. Section 3.4 gives theoretical guarantees on Y_0's performance when it is generalized to arbitrary overlay topologies. Section 3.5 evaluates Y_0 and Chord through simulation. Section 3.6 discusses related work and Section 3.7 concludes.

3.2 Preliminaries

3.2.1 Model and Assumptions

We assume a system with n physical nodes. Node v has a fixed **capacity** c_v. Capacities are normalized so that the average capacity is 1; that is, $\sum_v c_v = n$. Our use of a scalar capacity assumes that there is a single important resource on which nodes are constrained, such as storage space, processing speed, or last-mile bandwidth.

We say that an event happens with high probability (w.h.p.) when it occurs with probability $1 - O(n^{-1})$.

We assume that, with high probability, every node v can estimate n and its own capacity c_v fairly accurately. Specifically, with high probability, for all v node v produces an estimate $\tilde{n}_v$ of n such that $\frac{1}{\gamma_n} n \leq \tilde{n}_v \leq \gamma_n n$. (Thus, the probability that *any* node has a poor estimate is $O(n^{-1})$.) Similarly, with high probability, for all v node v produces an

estimate $\tilde{c_v}$ such that $\frac{1}{\gamma_c} c_v \leq \tilde{c}_v \leq \gamma_c c_v$. We further assume these estimates are unbiased. Estimation of n is discussed in [77]. Since capacities are normalized, to estimate c_v, a node will need an estimate of the average capacity. One may obtain a crude estimate through random sampling of other nodes, such as the successors in the ID space. While this estimate is unbiased, it can have high variance depending on the capacity distribution. The techniques of [88] could be applied to DHTs to estimate both n and the average capacity and would provide guarantees on the quality of the estimate for any capacity distribution.

We assume a DHT that manages a unit-size circular ID space, *i.e.*, $[0, 1) \subseteq \mathbb{R}$ with arithmetic modulo 1. We assume the DHT uses consisting hashing [63] to partition the ID space among the nodes as in Chord. Each node v picks an ID $\mathrm{id}(v) \in [0, 1)$ and is assigned ownership of the region $(\mathrm{id}(w), \mathrm{id}(v)]$ where $\mathrm{id}(w)$ is the nearest preceding node's ID. A node may pick multiple IDs (virtual servers) in which case it owns the union of the associated regions.

3.2.2 The ID Selection Problem

Under the uniform load assumption, the load on a node will be proportional to the size of the ID space it owns. Thus, picking a load-balanced partitioning amounts to selecting appropriate IDs for the nodes, which we call the **ID selection problem**. A good solution has the following properties.

Low maximum share Let the **share** of node v be the fraction f_v of the ID space assigned to it, divided by its "fair share" of the ID space:

$$\mathrm{share}(v) = \frac{f_v}{c_v / n}.$$

To achieve a good load balance, the maximum share of any node in the system should be as low as possible—ideally 1, in which case the load on each node would be exactly proportional to its capacity.

Low load movement To maintain load balance, nodes may need to select different IDs when other nodes arrive or depart. This can be costly because reassigning ownership of the ID space implies data movement and changes to the overlay connections. Thus, we desire little change in the ID space partitioning upon node arrivals and departures.

Low normalized degree The **normalized degree** of a node v is $\deg(v)/c_v$, where $\deg(v)$ is the number of distinct nodes to which v maintains connections or which maintain connections to v in the overlay network used for routing. We wish to minimize the average and maximum normalized degree. ID selection directly impacts node degree since the overlay topology is defined in terms of the partitioning of the ID space.

The main reason for the last objective is not to reduce the memory footprint of the routing table, but to reduce the overhead of the control traffic required to keep the routing table entries up to date. For example, as we will see, a Chord node may easily have 50 neighbors. Assuming it sends each neighbor a minimum-length UDP message of 28 bytes every 1 second to verify liveness, this results in 1.4 KB/sec of traffic. That rate may already be significant if our goal is for the DHT to be very low-overhead infrastructure—and using 10 basic virtual servers per node, the traffic incrases to an undesirable 14 KB/sec. Traffic can be reduced by pinging less frequently but this proportionally increases the time to detect a failure.

3.2.3 Basic Virtual Server Selection (Basic-VSS) Scheme

As discussed in the introduction, the virtual server technique can be used to balance load not only in a homogeneous system, but also in a heterogeneous system [27]. However, the precise way in which the technique is applied to highly heterogeneous systems has not been specified. In the reminder of this section we present a simple strategy that we later adapt in Υ_0.

Let $\alpha = \alpha(n)$ be **the number of virtual servers per unit capacity**. When $\alpha = 1$, nodes of average capacity have a single virtual server, but the maximum share is $\Theta(\log n)$. When $\alpha = \Theta(\log n)$, we can achieve $\Theta(1)$ maximum share but the degree of a node increases by a factor $\alpha = \Theta(\log n)$.

The main issue we address in this section is how to handle low capacity nodes. Since the total system capacity is n, there are roughly αn virtual servers. Thus, the expected fraction of ID space associated with a single virtual server is $1/(\alpha n)$. There is a tradeoff in the choice of α: if α is small, very low capacity nodes will be overloaded even if they maintain only a single virtual server. On the other hand, a large α leads to high degree, as even nodes of average capacity must maintain α virtual servers and the associated overlay connections for each. In particular, to ensure that the nodes of minimum capacity c_{min}

1. $\tilde{n}, \tilde{c}_v \leftarrow$ estimates of n and c_v, respectively
2. $m \leftarrow$ if $\tilde{c}_v < \gamma_d$ then 0 else $\lfloor 0.5 + \tilde{c}_v \alpha(\tilde{n}) \rfloor$
3. Choose m virtual servers with IDs $r_1, \ldots, r_m$ where each r_i is uniform-random $\in [0, 1)$
4. Reselect IDs as above when $\tilde{c}_v$ changes by a factor $\geq \gamma_u$ or $\tilde{n}$ changes by a factor ≥ 2

Figure 3.1: The Basic Virtual Server Selection Scheme (Basic-VSS), run at each node v.

Parameter	Description
$\gamma_c, \gamma_n \geq 1$	Bound the maximum factor error (w.h.p.) in each node's estimate of its capacity and of n, respectively; see Section 3.2.1.
$\gamma_d < 1$	Capacity threshold below which a node is discarded.
$\gamma_u > 1$	Each node updates its IDs when its estimate of its capacity changes by a factor γ_u. (must have $\gamma_u > \gamma_c$ to avoid instability)
$\alpha(n)$	Number of virtual servers per unit capacity (Section 3.2.3).

Figure 3.2: Parameters of both Basic-VSS and Y_0's LC-VSS.

have close to their fair share c_{min}/n of the ID space, we must have $1/(\alpha n) = O(c_{min}/n)$, *i.e.*, $\alpha = \Omega(1/c_{min})$. But since c_{min} may be arbitrarily small, α may be arbitrarily large, implying very high overhead. Moreover, c_{min} may be unstable and hard to estimate.

We avoid this tradeoff by simply discarding nodes whose capacity is lower than some **discard threshold** γ_d. If $c_v < \gamma_d$, then node v does not instantiate any virtual server, and does not participate in the standard DHT routing protocol. The remaining nodes v with capacity $c_v \geq \gamma_d$ pick $c_v \cdot \alpha$ virtual servers. We call this algorithm the **Basic Virtual Server Selection (Basic-VSS) Scheme** and show the pseudocode in Figure 3.1. Figure 3.2 shows the main parameters of the algorithm.

A natural concern with this algorithm is that it might discard too much capacity, overburdening the remaining nodes. But it is easy to see that in the worst case, at most a fraction γ_d of the total capacity is discarded—ignoring estimation error and lazy update to avoid instability and excessive load movement as (normalized) capacity changes. Removing those simplifications, we have:

Claim 1 *In the Basic-VSS scheme, the fraction of the total capacity remaining in the ring is at least* $1 - \gamma_c \gamma_u \gamma_d$ *w.h.p.*

Proof: See Appendix B. ■

An optimization deserving of further study is to fix α and find the γ_d which minimizes the maximum share. However, we expect that $\gamma_d = \frac{1}{2}$ will be acceptable for most applications. With this choice, the capacity of any node remaining in the ring is at least half the average capacity, so we do not need to significantly increase α to handle the low-capacity nodes. Although in the worst case 50% of the total capacity is discarded, in our simulations of Section 3.5, less than 20% is discarded in a range of power law capacity distributions and less than 10% in a real-world distribution. If the discarded capacity were excessive in some distribution, we could use a smaller γ_d at the cost of increasing α to maintain the same load balance.

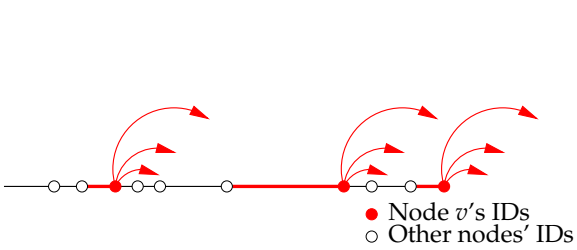
Furthermore, if we cannot afford to discard *any* capacity, we can use discarded nodes for data storage (but not routing) by having each discarded node pick a "parent" in the ring which would assign it data to store.

Discarded nodes may still perform lookup operations in the DHT although they are not part of the ring structure and do not route messages. We connect discarded nodes to the system through k links to nodes owning the IDs $r + \frac{1}{k}, \ldots, r + \frac{k}{k}$, where $r \in [0, 1)$ is chosen randomly. In our simulations of Section 3.5, we use $k = \lceil 3c_v \log_2 n \rceil$ for node v since with $\alpha = 1$, Chord nodes in the ring have roughly $3c_v \log_2 n$ outlinks as well ($\approx \log n$ fingers and $\approx 2 \log n$ successors for each of the $\approx c_v$ virtual servers).

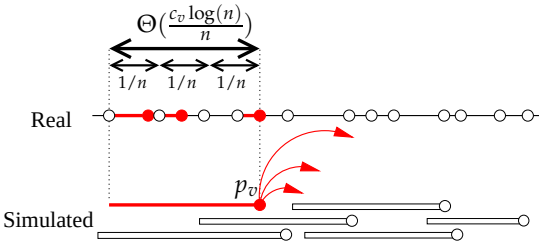
This raises another natural question: whether the congestion on the nodes in the ring will increase due to lookup operations. However, Claim 1 implies that the increase cannot be too great. Indeed, in Section 3.5.1, we will see a slight drop in congestion when moving from homogeneous to heterogeneous capacities.

3.3 The Y_0 DHT

In this section we present the Y_0 DHT, which is composed of a **Low Cost Virtual Server Selection (LC-VSS) Scheme** and simple changes to Chord's successor lists, finger tables, and routing. Later, in Section 3.4.3, we present formal guarantees for LC-VSS applied to any overlay topology, not just Chord.



(a) **Basic-VSS** gives a node of capacity c_v ownership of $\Theta(c_v \log n)$ disjoint segments of the ID space. The DHT must maintain a set of overlay links for each.



(b) Y_0's **LC-VSS** scheme results in $\Theta(c_v \log n)$ disjoint segments clustered in a $\Theta(\frac{c_v \log n}{n})$ fraction of the ID space. When Y_0 builds the overlay, it *simulates* ownership of one contiguous interval. The nodes' simulated intervals overlap.

Figure 3.3: ID selection illustrated.

3.3.1 Low Cost Virtual Server Selection (LC-VSS) Scheme

The LC-VSS scheme which selects IDs for Y_0's virtual servers is shown in the illustration of Figure 3.3 and in pseudocode in Figure 3.4. As in the Basic-VSS scheme of Section 3.2.3, nodes of capacity less than γ_d are discarded and each remaining node v chooses $\Theta(c_v \alpha)$ IDs, where $\alpha = \Theta(\log n)$. However, these IDs are not selected randomly from the entire ID space. Instead, we pick a random starting point p_v and then select one random ID within each of $\Theta(\log n)$ consecutive intervals of size $\Theta(1/n)$. When n or c_v changes by a constant factor, the ID locations are updated. The algorithm inherits the parameters shown in Figure 3.2.

It has been proposed to compute a node's ID as a hash of its IP address, which makes it more difficult for a node to hijack arbitrary regions of the ID space [114]. To support this, we could easily replace the randomness in the description of the algorithm with such hashes.

3.3.2 Successor Lists

In Chord, each virtual server keeps links to its $2 \log_2 n$ successors in the ring [114]. The purpose is fault tolerance: when each node fails with probability $\frac{1}{2}$, a remaining virtual server will still have a link to its immediate successor with probability $1 - \left(\frac{1}{2}\right)^{2 \log_2 n} = 1 - O(1/n^2)$, so *all* remaining virtual servers will have their successor links with probability

Initialization:

1. $p_v \leftarrow$ random ID $\in [0,1)$
2. $\tilde{n}, \tilde{c}_v \leftarrow$ estimates of n and c_v

ID selection:

1. $spacing \leftarrow 2^{-\lfloor 0.5+\log_2 \tilde{n} \rfloor}$ (*i.e.*, roughly $1/\tilde{n}$)
2. $start \leftarrow \lfloor p_v/spacing \rfloor \cdot spacing$
3. $m \leftarrow$ if $\tilde{c}_v < \gamma_d$ then 0 else $\lfloor 0.5 + \tilde{c}_v \alpha(\tilde{n}) \rfloor$
4. Choose m IDs at $start - (i + r_i) \cdot spacing$ for each $i \in \{0, \ldots, m-1\}$ and each r_i chosen uniform-randomly $\in [0,1)$

Periodic update:

1. Obtain new estimates $\tilde{c}'_v, \tilde{n}'$
2. If $\tilde{c}'_v$ is at least a factor γ_u from $\tilde{c}_v$, set $\tilde{c}_v \leftarrow \tilde{c}'_v$
3. If $spacing \notin [\frac{1}{2\tilde{n}'}, \frac{2}{\tilde{n}'}]$, set $\tilde{n} \leftarrow \tilde{n}'$
4. If either $\tilde{n}$ or $\tilde{c}_v$ changed, reselect IDs as above

Figure 3.4: Y_0's LC-VSS scheme run at each node v.

$\geq 1 - O(1/n)$. Thus, the ring structure is preserved w.h.p. To obtain the same bound in Y_0, each node must keep links to the $2 \log n$ *distinct* nodes succeeding each of its virtual servers.

In more detail, Chord constructs its successor list as follows. Each virtual server vs belonging to node v obtains its successor list from its immediate successor $succ(vs)$ and adds $succ(vs)$ to the list. If the list is of length $> 2 \log n$, it drops the clockwise-farthest entry. In Y_0, we drop any virtual server belonging to v before considering dropping the clockwise-farthest entry. Beginning with an empty list at each node, periodic repetition of this algorithm builds a list of $2 \log_2 n$ distinct successors.

Will a node's $\Theta(c_v \log n)$ virtual servers involve $\Theta(c_v \log^2 n)$ successor links, as in Basic-VSS? We show in Section 3.4.3 that since Y_0's IDs are clustered, these $\Theta(c_v \log^2 n)$ *logical* successors fortunately involve only $\Theta(c_v \log n)$ *distinct* nodes, so only $\Theta(c_v \log n)$ network connections need to be maintained. For routing purposes Y_0 remembers all the

logical IDs, so memory use does increase by a logarithmic factor. But this is not likely to be detrimental: even when $n = 2^{30}$, a node with $\log n$ virtual servers would maintain $2\log^2 n = 1800$ logical successors, which at a generous 1 KB each uses less than 2 MB of RAM.

3.3.3 Finger Tables

For each virtual server with ID x, Chord keeps a **finger table** of links to the nodes owning the "target IDs" $(x + b^{-i}) \mod 1$ for $i = 1, 2, \ldots$. Typically $b = 2$.

In Y_0, for the purposes of overlay construction, each node v **simulates** ownership of a contiguous interval I_v of size $\Theta(\frac{c_v \log n}{n})$ enclosing its virtual servers, building one set of overlay links as if it actually owned all of I_v. This is depicted in Figure 3.3b. Specifically, v simply keeps one finger table emanating from the clockwise end of I_v (see Section 3.3.1). An important property is that although a node's virtual server IDs may change, p_v is fixed. Thus, the finger table changes only when a target ID changes ownership.

This construction allows us to choose a better overlay topology than Chord in the heterogeneous case. The size of I_v scales with the node's capacity, but this does not affect the size of the finger table. Since we allow each node a number of outlinks proportional to its capacity, we have some unused capacity on high-capacity nodes. We use this to reduce route length by choosing a "denser" set of overlay links on high-capacity nodes: node v chooses fingers at integer powers of b^{1/c_v}, where b is a parameter which we take to be 2 by default as in Chord. This results in $O(\log_{b^{1/c_v}} n) = O(c_v \log_b n)$ fingers w.h.p. Note that, unlike the rest of Y_0, this technique does not generalize to arbitrary overlay topologies.

3.3.4 Routing

To find the owner of some ID x, Chord routes greedily on the finger table, at each step selecting the neighbor whose ID is clockwise-closest to x. In Y_0, we employ the same algorithm except that the successor list is also considered in finding the closest neighbor, rather than just the finger table.

The result of using this algorithm in Y_0 will be that within $O(\log n)$ steps, due to the presence of the finger table, the message is routed to a node v for which $x \in I_v$. At this point, v may not truly own x, and we have no fingers which are closer to the true destination. However, due to the clustering of v's IDs, the real owner is not far from one

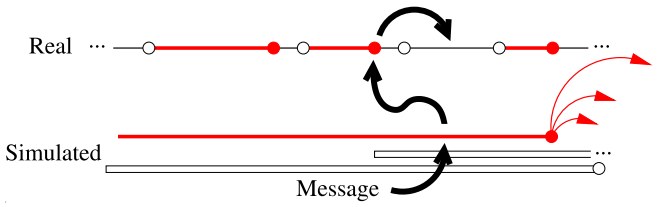


Figure 3.5: Routing in Y_0. A close-up of part of the ID space is shown.

of v's virtual servers. Specifically, they are separated by $O(\log n)$ nodes along the ring, so the true owner will be reached using successor lists within $\Theta(1)$ additional hops w.h.p., as shown in Figure 3.5. We will prove these results in Section 3.4.4.

3.4 Analysis

In this section we analyze the performance of Y_0 with respect to the metrics described in Section 3.2.2. The results apply Y_0's techniques to any ring-based DHT, e.g [62, 78, 87, 104, 114]. We prove the following:

- Y_0 can achieve a near-optimal maximum share of $1 + \varepsilon$ for any $\varepsilon > 0$ (Section 3.4.1).
- As long as the number of nodes n and the average capacity remain relatively stable, the amount of load movement that Y_0 incurs to maintain a good balance is close to optimal (Section 3.4.2).
- Compared to the case of a single virtual server, Y_0 with $\alpha = \Theta(\log n)$ increases the number of distinct links that a node maintains by at most a constant factor, while providing flexibility in neighbor selection for any topology (Section 3.4.3).
- Compared to the case of a single virtual server, Y_0 with $\alpha = \Theta(\log n)$ increases route length by at most an additive constant (Section 3.4.4).

We defer all proofs to Appendix B.

3.4.1 Load Balance Bounds

Our first theorem says that the maximum share can be made arbitrarily close to 1, even in the heterogeneous case. Throughout the analysis we assume that all nodes have performed their periodic update step since the last event in the system.

Theorem 2 *For any $\varepsilon > 0$, if $\alpha \geq \frac{8\gamma_n\gamma_c\gamma_u}{(1-\gamma_c\gamma_u\gamma_d)\gamma_d\varepsilon^2} \cdot \ln n$, then the maximum share of a node is at most $\frac{(\gamma_c\gamma_u)^2}{(1-\varepsilon)^2(1-\gamma_c\gamma_u\gamma_d)} + o(1)$ w.h.p.*

Proof idea: When $\alpha = \Theta(\log n)$, we have $\Theta(\log n)$ IDs in each region of the ID space of size $\Theta(1/n)$ w.h.p. regardless of the capacity distribution. With this balanced distribution of IDs, the share associated with each ID is approximately a geometric random variable. Applying a Chernoff bound to their sum yields the result. ■

Despite the frightening plethora of constants in the bound, our simulations of Section 3.5 show that we get a maximum share of less than 3.6 with a modest $\alpha = 2\log_2 n$ in various capacity distributions. Furthermore, most of the complexity in the above guarantee is due to varying node capacities. Hard-coding $\tilde{c}_v = 1$ into LC-VSS and a straightforward modification of our analysis yields the following bound:

Theorem 3 *For any $\varepsilon > 0$, if node capacities are homogeneous and $\alpha \geq \frac{8\gamma_n}{\varepsilon^2} \ln n$, the maximum share of a node is at most $(1-\varepsilon)^{-2} + o(1)$ w.h.p.*

3.4.2 Load Movement Bounds

To maintain load balance as nodes join and leave, nodes occasionally update their IDs. Each move of a virtual server is equivalent to a *leave* followed by a *join* under a new ID. Thus, load balancing effectively increases the churn rate of the system, which involves costly object movement and overlay link changes. In this section we bound this overhead in terms of the amount of churn in the population of nodes in the system.

Given a sequence of node join and leave events, we say that the **underlying churn** is the sum over all events of the fraction change in *system capacity* due to the event. Specifically, if the system currently has total capacity C and a node of capacity c joins, the underlying churn increases by $c/(C+c)$; if the same node subsequently leaves, churn increases by the same amount again. Similarly, the **effective churn** is the sum over all events of the fraction change in *ID space ownership* due to the event, which depends on the policy of the partitioning scheme.

Definition 1 *The* **churn ratio** *for a sequence of events is the DHT's expected effective churn divided by its underlying churn over those events.*

This is equivalent to a metric used in [15].

We are interested the churn ratio after t events for which the underlying churn is $\Omega(t)$ (to avoid degenerate cases such as zero-capacity nodes joining and leaving forever). We also assume the system always has positive capacity. Finally, for simplicity we assume LC-VSS's estimation error bounds γ_c, γ_n hold with probability 1.

Since every part of the ID space has to be assigned to some node, it is easy to see that the churn ratio is ≥ 1 in the worst case for any scheme—that is, the effective churn is at least the underlying churn. Basic-VSS in a homogeneous environment with $\alpha = 1$ (*i.e.*, the standard Chord protocol) achieves this lower bound, since in expectation the change in ID space due to a join or leave is exactly the change in total system capacity, and once joined, a node never changes its ID.

Our bound on LC-VSS's churn ratio will apply to event sequences during which the total number of nodes and average capacity don't vary greatly (but the underlying churn rate may be arbitrarily high).

Definition 2 *The system is* (β, δ)**-stable** *over a sequence of events when the minimum and maximum values of n differ by less than a factor β, and the minimum and maximum values of the average capacity differ by less than a factor δ.*

Theorem 4 *If the system is* $(\frac{2}{\gamma_n}, \frac{\gamma_u}{\gamma_c})$*-stable, the churn ratio of LC-VSS is* $\frac{1}{1-\gamma_c\gamma_u\gamma_d} + o(1)$.

This reduces to $1 + o(1)$ in the homogeneous case. Thus, during periods wherein n and the average capacity are relatively stable—which is likely the common case—the overhead of LC-VSS's load balancing operations is small. The result follows from the fact that in this case very few nodes need to reselect IDs.

3.4.3 Overlay Construction and Degree Bounds

In this section we describe how to use LC-VSS with any overlay topology without significantly increasing outdegree, while providing some flexibility in neighbor selection, even if the original topology had none.

Sequential Neighbors. We deal first with what [54] terms **sequential neighbors**. In ring-based DHTs, each node v keeps links to nodes whose IDs are close to each of v's IDs — either the $k = \Theta(\log n)$ successors in the ring [114] or the k successors and k predecessors [104]. As discussed in Section 3.3.2, in Y_0, each node keeps links to the k *distinct* nodes closest to each of its IDs.

In our overlay construction we assume $k = \Theta(\log n)$. Since $\alpha = \Theta(\log n)$, this implies that nodes have $\Theta(c_v \log^2 n)$ *logical* sequential neighbors. The following theorem shows that due to the clustering of IDs, the number of *distinct* neighbors is low.

Theorem 5 *Each Y_0 node has $\Theta(c_v\alpha) = \Theta(c_v \log n)$ distinct sequential neighbors w.h.p.*

Long-Distance Neighbors. There is a significant amount of variation in the structure of "long-distance" overlay links in DHTs. To analyze the construction of arbitrary topologies on top of Y_0's partitioning of the ID space, we extend and formalize Naor and Wieder's Continuous-Discrete Approach [87]. We model the overlay topology in continuous form, as a **neighborhood function** E which maps a contiguous interval $(p_1, p_2] \subseteq [0, 1)$ to a subset of $[0, 1)$. E specifies that the owner of $(p_1, p_2]$ should be connected to the region $E(p_1, p_2)$ of the ID space. For example, Chord's finger table would be specified as follows, with all values modulo 1:

$$E(p_1, p_2) = \left\{ p_2 + \frac{1}{2}, \quad p_2 + \frac{1}{4}, \quad p_2 + \frac{1}{8}, \quad \ldots \right\}.$$

Chord does not depend on p_1, but general topologies may. The Distance Halving overlay of [87] has

$$E(p_1, p_2) = \left(\frac{p_1}{2}, \quad \frac{p_2}{2} \right] \cup \left(p_1 + \frac{1}{2}, \quad p_2 + \frac{1}{2} \right].$$

But in a real system, rather than connecting portions of the ID space with other portions of the ID space, we connect physical nodes with other physical nodes. We translate the continuous graph E to a graph on nodes thusly:

Definition 3 *A* **discretization** *of E is a* **simulated interval** $I_v \subseteq [0, 1)$ *for each node v and a graph G on the nodes in the system such that*

$$\forall v \; \forall p \in E(I_v) \quad (\exists (v, w) \in G \quad p \in I_w).$$

That is, if I_v is connected to a point $p \in E(I_v)$ in the continuous graph, then v is connected to some node $w \in G$ simulating ownership of p.

Thus, a discretization guarantees that each edge in the continuous graph can be simulated by the discrete graph. The set I_v defines what part of the continuous graph node v simulates. Note that the I_vs may overlap.

The simplest example is what we call the **Standard Discretization**. For each v we simply let I_v be the subset of the ID space which v owns. The edges of G are the minimal set which satisfies the above definition: v has a link to each node which owns part of $E(I_v)$. This edge set is unique because ownership of the ID space is a partitioning: for each $\ell \in E(I_v)$, there is exactly one node w with $\ell \in I_w$. This discretization is equivalent to the operation of Chord and the Distance Halving DHT.

To adapt to our multiple-virtual server setting with low degree, we use the following **Shared Discretization**. For each v we let I_v be the smallest contiguous interval which contains the (disjoint) set of IDs that v owns as well as p_v (recall the definition of p_v from Figure 3.4). Since the I_vs overlap, we have a fair amount of choice in selecting the edge set in a way that satisfies Definition 3. We use a simple greedy algorithm for each node v. Initially, label all of $E(I_v)$ uncovered. Iterate by picking the uncovered point p which is the first from p_v in the clockwise direction. Contact p's owner, get its successor list, its furthest successor's successor list, etc. until we find a node w for which I_w covers all of $[p, p + \Theta(\frac{\log n}{n})] \cap E(I_v)$. Add an edge from v to w. Terminate when all of $E(I_v)$ is covered.

The above procedure clearly satisfies Definition 3, assuming eventual termination and success at finding the ws. The number of overlay links it creates depends on E. Let $f(n)$ be a nondecreasing function such that if $|I| \leq \frac{1}{n}$, then $E(I)$ can be covered by $\leq f(n)$ segments of length $\frac{1}{n}$. Taking $f(n)$ to be as small as possible intuitively provides a lower bound on node degree: even in a homogeneous system with a perfect partitioning of the ID space, the Standard Discretization must give some nodes $\geq f(n)$ outlinks. The following theorem says that in LC-VSS, nodes' outdegrees are inflated by at most a constant factor.

Theorem 6 *Using LC-VSS with $\alpha = \Theta(\log n)$, the Shared Discretization algorithm terminates successfully, and each node v has at most $O(c_v f(n))$ outlinks w.h.p. for any capacity distribution.*

Flexibility in neighbor selection. The ability to adapt the DHT topology dynamically, rather than having the topology be a deterministic function of node IDs, has several benefits. The DHT can perform proximity-aware neighbor selection, which can significantly reduce the latency of DHT lookups [54], and can select neighbors based on their stability, which can improve reliability (as we will see in Chapter 4). Whether the overlay topology accommodates this flexibility is an important difference between different topologies; although Chord has a great deal of flexibility, some DHT topologies do not. [54].

One benefit of our Shared Discretization is that it provides flexibility in the choice of each of a node's neighbors, *even if there was no such choice in the original topology*. More specifically, as noted in the proof of Theorem 6 (see Appendix B), there will be $\Theta(\log n)$ choices for each neighbor. Intuitively, this occurs because the simulated intervals overlap at a depth of $\Theta(\log n)$ (under the safe assumption that node capacities are at most $O(n/\log n)$). Even this fairly limited choice is likely to provide a significant benefit. Simulations suggest that for proximity-aware neighbor selection, 16 choices for each neighbor provide nearly all the benefit that an unbounded number of choices would provide [28].

3.4.4 Route Length

To complete our adaptation of the overlay, we show that we can simulate the overlay's routing function while increasing route length by at most an additive constant.

The continuous overlay is accompanied by a **routing function** $r(I, t)$ which, given a current location $I \subseteq [0, 1)$ and a target destination t, returns an ID in $E(I)$ representing the next hop. At each step, we apply the routing function to the current location I and move to a neighbor w for which $r(I, t) \in I_w$. Since $r(I, t) \in E(I)$, by definition, any discretization must have an edge to some such w. We set $I \leftarrow I_w$ and iterate. If after $f(m)$ such steps the distance from some $\ell \in I$ to t in the ID space is $\leq 1/m$ for any starting location and any t, we say that E has **path length** $f(m)$.

Thus, in $f(m)$ hops, any discretization can arrive at a node v such that some ID in I_v is at distance $\leq 1/m$ from the destination t. Completion of the discrete routing then depends on the relation between I_v and what v actually owns in the ID space, which depends on the discretization.

In the Standard Discretization, I_v is exactly the ID space owned by v, so we are at a node which owns a part of the ID space at distance $1/m$ from our destination. Thus, we can pick m large enough that $1/m$ is less than the minimum distance between two nodes' IDs. With such a choice, the continuous phase must finish at the successor or predecessor of the owner of t, or the owner itself, at which point we can reach the destination in ≤ 1 hop. For example, in Chord with a single virtual server per node, nodes are separated by distance $\geq 1/n^3$ w.h.p. (Claim 8 in Appendix B) so route lengths are $\leq 1 + f(n^3) = 1 + 3\log_2 n$ w.h.p. We can improve the bound by using the successor list, as follows. A standard balls-and-bins result shows that Chord picks at most $O(\log n)$ IDs in each interval $1/n$ of the ID

space w.h.p. Thus, we can route in $\log_2 n$ hops to within distance $1/n$ of t, and continue with greedy routing on the successor list of length $\Theta(\log n)$ to route through the remaining distance in $O(1)$ hops.

In the Shared Discretization, unlike the Standard Discretization, v does not own all of I_v. However, due to LC-VSS's placement of IDs at intervals of $O(1/n)$ in the ID space, v will always have a virtual server whose ID is at distance $O(1/n)$ from t. A Chernoff bound shows that when $\alpha = O(\log n)$, the owner of t is $O(\log n)$ successors away, and we route to our destination in $O(1)$ hops using the successor lists which we keep for each chosen ID. Thus, we arrive at the following result.

Theorem 7 *Using LC-VSS and the Shared Discretization of any overlay topology with path length $f(n)$, any message can be routed in $f(n) + O(1)$ hops w.h.p.*

3.5 Simulation

We compare Y_0 and Chord using a simple simulator which constructs the ID space partitioning and overlay topology of the two DHTs in a static setting. To provide heterogeneity-aware load balance in Chord, we use the Basic-VSS scheme of Section 3.2.3.

We assume that each node is given its capacity and n; *i.e.*, there is no estimation error. We use $\gamma_d = \frac{1}{2}$. Each data point represents an average over 15 independent trials. We show 95% confidence intervals. We show results for n equal to powers of 2. For intermediate n, there is some variation in the load balance/degree tradeoff space for Y_0 because its spacing between virtual servers is rounded to the nearest power of 2. Specifically, simulations show maximum share increases by up to $\approx 10\%$ when degree decreases up to 10%, or maximum share decreases up to 10% while degree increases up to 25%.

We study three types of node capacity distributions: (1) homogeneous, (2) power law with various exponents, and (3) samples from the uplink bottleneck bandwidths of actual Gnutella hosts measured by Saroiu et al [107], which we call the **SGG distribution**. Thus, for this simulation, we consider the capacity of a node to be the bandwidth of its connection to the Internet. We have discarded points in their raw data set which show bandwidth higher than 1 Gbps, since the authors of [107] believe those to be artifacts of faulty measurement and discard them in their study as well [105].

Our simulation results are as follows.

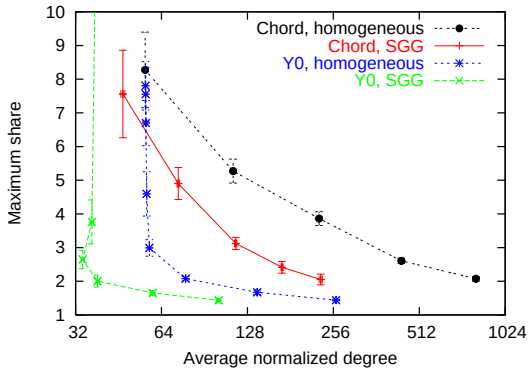


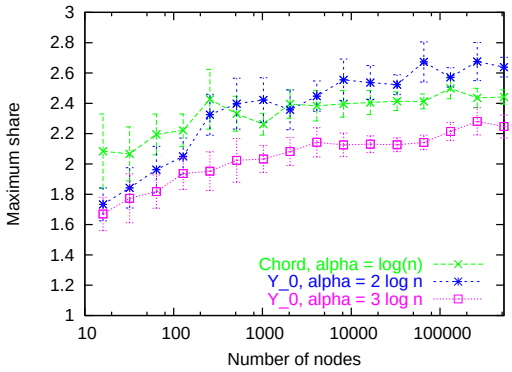
Figure 3.6: Tradeoff between maximum share and average normalized degree, achieved through varying α, for $n = 2048$. For Chord, $\alpha \in \{1,2,4,8,16\}$, and for Y_0, $\alpha \in \{1,2,4,\ldots,128\}$.

- Y_0 achieves a maximum share of less than 3.6 with $\alpha = 2\log n$, where α is the number of virtual servers per unit capacity. This is roughly as well-balanced as Chord with $\alpha = \log n$ (Section 3.5.1).
- The degree of nodes in Y_0 with $\alpha = 2\log n$ is as low as Chord with $\alpha = 1$ and even lower in some heterogeneous distributions (Section 3.5.2).
- In both DHTs, route lengths decrease as heterogeneity increases, but Y_0 has a clear asymptotic advantage (Section 3.5.3).

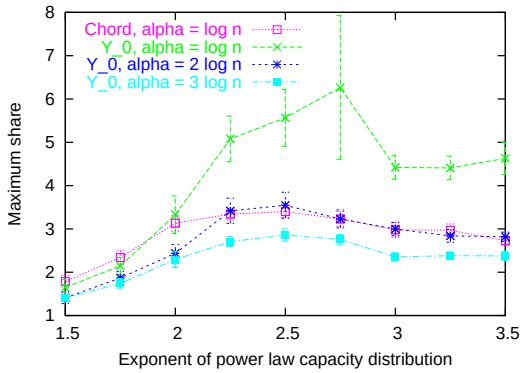
3.5.1 Load Balance

Figure 3.6 summarizes the tradeoff between quality of load balance and degree. We obtain various points on the tradeoff curve for each protocol by selecting different α. Y_0 provides a substantially better set of achievable points. Both algorithms perform significantly better in the SGG distribution, although we will see (Figures 3.7b and 3.9b) that not all capacity distributions are better than the homogeneous case.

Figure 3.7a shows that with $\alpha = 2\log n$, the maximum share of any node in Y_0 is less than 2.7 in a homogeneous environment — nearly as good as Chord with $\alpha = \log n$ but (as we will see) at much lower cost. A larger α provides a slightly better balance. Figure 3.7b shows that in a range of power law capacity distributions with varying exponent, the balance can be slightly worse or better than in the homogeneous case, at all times



(a) Maximum share vs. n with homogeneous capacities. Chord with $\alpha = 1$ (not shown) increases to a maximum share of ≈ 13.7.



(b) Maximum share vs. capacity distribution in a 16,384-node system.

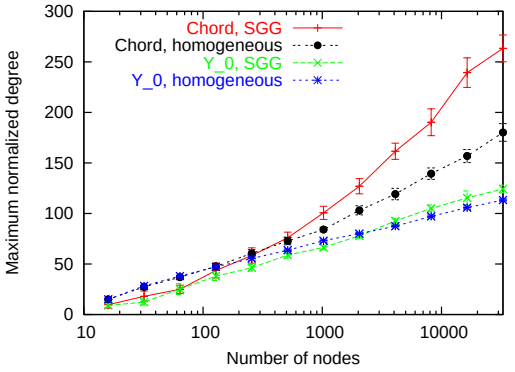
Figure 3.7: Maximum share

staying below 3.6.

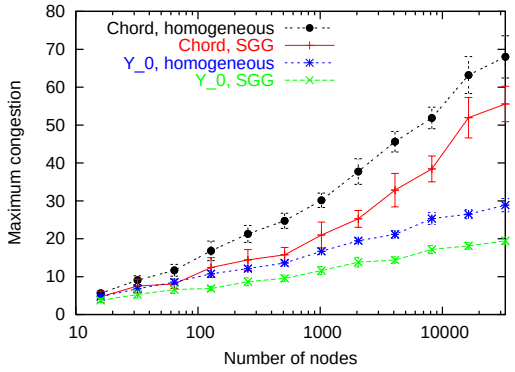
Once the ID space is balanced, balance of overlay connections and routing load follow closely. We measure maximum degree as defined in Section 3.2.2. To measure balance of routing load, we have each node route a message to a random ID. The **congestion** at a node is the number of messages that flow through it, divided by its capacity. Both metrics are $\Theta(\log^2 n)$ in Chord with $\alpha = 1$: essentially, $\Theta(n \log n)$ messages and $\Theta(n \log n)$ fingers are distributed uniformly in the ID space, and some nodes own a fraction $\Theta(\frac{\log n}{n})$ of the space. Y_0's constant-factor load balance thus implies $\Theta(\log n)$ maximum degree and maximum congestion. This is illustrated in Figure 3.8. Note that the reduced congestion in the heterogeneous case results from reduced route length, which we cover in Section 3.5.3.

3.5.2 Normalized Degree

In this section we evaluate the effectiveness of our technique at maintaining low average normalized degree, defined in Section 3.2.2. We average over all nodes that were not discarded. Figure 3.9a shows this metric as a function of α with homogeneous capacities and $n = 2048$. We see essentially no increase in degree in Y_0 until $\alpha > 2 \log n \approx 22$. This is because when $\alpha \leq 2 \log n$, the additional links associated with the virtual servers are to essentially the same set of nodes with which we are already linked due to the $2 \log n$

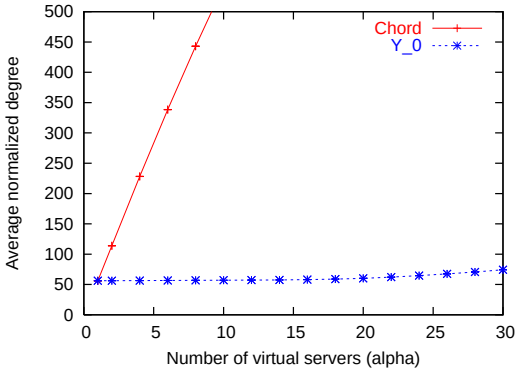


(a) Maximum normalized degree vs. n, with $\alpha = 1$ in Chord and $\alpha = 2\log n$ in Y_0.

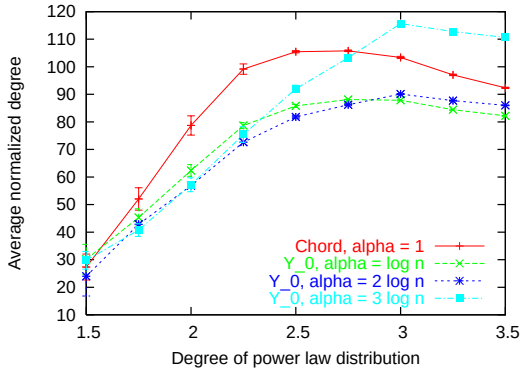


(b) Maximum congestion vs. n, with $\alpha = 1$ in Chord and $\alpha = 2\log n$ in Y_0.

Figure 3.8: Load balance of overlay links and routing load

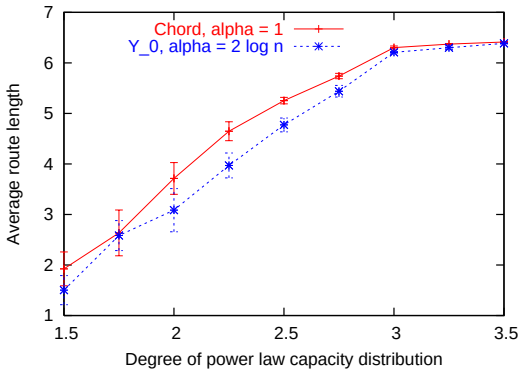


(a) Average normalized degree vs. number of virtual servers in a homogeneous 2048-node system.

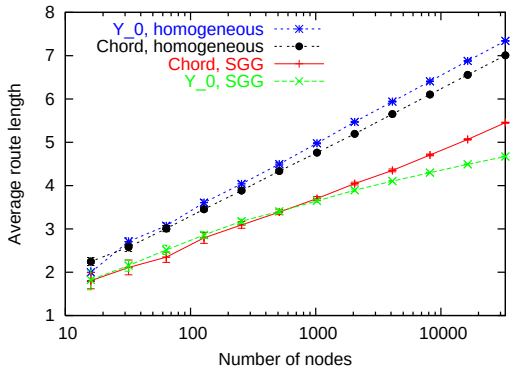


(b) Average normalized degree vs. capacity distribution in a 16,384-node system.

Figure 3.9: Average normalized degree



(a) Route length vs. capacity distribution in a 16,384-node system.



(b) Route length vs. n. In Chord, $\alpha = 1$ and in Y_0, $\alpha = 2 \log n$.

Figure 3.10: Route length

incoming successor links. Even when $\alpha > 2 \log n$, Theorems 5 and 6 imply that Y_0's degree is $O(\alpha)$ rather than $O(\alpha \log n)$ as in Chord.

Due to the normalization by capacity in our degree metric, Y_0's improved load balance gives it a lower average normalized degree than Chord in the heterogeneous case. This is depicted in Figure 3.9b.

3.5.3 Route Length

To measure route length, we have each node route to a random ID, and average over the n resulting hop counts. Figure 3.10a shows decreasing route lengths in the power law distribution as the power law exponent decreases, with slightly greater benefit in Y_0 than in Chord. Note that large power law exponents approach a homogeneous system.

Figure 3.10b examines the asymptotics of route length. In a homogeneous system, Y_0 has very slightly higher route length than Chord, but as we showed in Theorem 7 this is only an additive constant. In the SGG distribution, both DHTs see significantly decreased route length. Chord's benefit is primarily due to the fact that there are fewer nodes in the ring: roughly 75% of the nodes in this distribution had capacity less than $\gamma_d = \frac{1}{2}$ and were discarded. This can only decrease route length by an additive constant, but it is significant in small- to medium-sized systems. At $n = 32,768$ nodes, Chord and Y_0 have 22% and 33% smaller mean route length, respectively, than Chord in the homogeneous case.

In Y_0, our technique of increasing the number of fingers at high-capacity nodes provides a constant-factor decrease in route length (note the lower slope). A least-squares fit of our data for networks with 1024 or more nodes yields the following estimated asymptotic route lengths (*i.e.*, we ignore additive terms):

DHT	Capacities	Route length	Normalized
Chord	Homogeneous	$0.450 \log_2 n$	100%
	SGG	$0.348 \log_2 n$	77%
Y_0	Homogeneous	$0.472 \log_2 n$	105%
	SGG	$0.203 \log_2 n$	45%

3.6 Related Work

ID space balance for homogeneous DHTs. The simplest way to balance load, under the uniform load assumption, is to obtain a $O(1)$ maximum share. The simplest way to do that is for each node to maintain $\Theta(\log n)$ virtual servers [63, 114]. Since this increases node degree by a factor of $\Theta(\log n)$, multiple proposals followed which give each node just one ID, but need to select and update that ID intelligently. All the proposals of this type are similar in that they consider $\Theta(\log n)$ locations for a node and select the one which gives the best load balance. They differ in which locations they check and when.

In Naor and Weider [87], a node checks $\Theta(\log n)$ random IDs when joining, and joins at the one which gives it the largest share. They show that this produces a maximum share of 2 if there are no node deletions. Handling node deletions requires an additional protocol, not precisely specified or analyzed in [87], whereby nodes are divided into groups of $\Theta(\log n)$ nodes and periodically reposition themselves within each group. In Adler et al. [2], a joining node contacts a random node v already in the DHT, and splits in half the largest interval owned by one of v's $O(\log n)$ overlay neighbors. This results in an $O(1)$ maximum share. A simple deletion protocol was given but not analyzed.

Manku's algorithm [77] has a joining node pick a random node v and split in half the largest interval owned by one of the $\Theta(\log n)$ nodes adjacent to v in the ID space. This achieves a maximum share of 2 while moving at most a single extra node's ID for each node arrival or departure. It extends to balancing within a factor $1 + \varepsilon$ but moves $\Theta(1/\varepsilon)$ IDs. In contrast, the results of Section 3.4.2 imply that Y_0 moves *no* extra IDs, even while achieving a maximum share of $1 + \varepsilon$, as long as n and the average capacity are relatively stable. The algorithm is more complicated than Y_0 and requires assignment

of IDs to arbitrary locations in the ID space, so we cannot use the security mechanism of requiring a node's ID to be one of several hashes of its IP address [27].

In one simple algorithm of Karger and Ruhl [65], each node has a fixed set of $O(\log n)$ possible IDs (so the security technique can be employed) and periodically reselects among them. This has maximum share $2+\varepsilon$, but it requires reassignment of $O(\log \log n)$ IDs per arrival or departure in expectation. Bienkowski et al [12] later gave a similar algorithm which reduces the number of reassignments to a constant, but they show only $O(1)$ maximum share. A second algorithm of [65] adapts to uneven distributions of objects in the ID space (which Y_0 and the previously mentioned algorithms cannot), but requires unrestricted ID selection and a special overlay topology if the object distribution is very skewed, and its maximum share is 64.

ID space balance for heterogeneous DHTs. Comparatively few schemes handle node heterogeneity. Dabek et al. [27] suggested that each physical node have a number of virtual servers proportional to its capacity, which we have developed into Basic-VSS in Section 3.2.3. Surana et al [44, 118] balance load dynamically by transferring virtual servers between physical nodes. This approach can handle node heterogeneity and load which is not distributed uniformly in the ID space, and was shown through simulation to produce a very good balance. But it is more complicated than Y_0, cannot employ the aforementioned security mechanism, cannot take advantage of heterogeneity to reduce route length as much as Y_0, and has higher degree due to its use of multiple virtual servers per node.

In a DHT-related work, Brinkmann et al. [15] develop two schemes which divide an ID space fairly among a set of nodes of heterogeneous capacities, providing efficient ID lookup and node join and leave algorithms. Their SHARE strategy is very similar to our LC-VSS: in both, each node selects an interval of the ID space of size $\Theta(\frac{\log n}{n})$, and ownership of a segment is "shared" among the nodes whose intervals overlap that segment. However, there are several differences. First, they assume a centralized system with no overlay network. As a consequence, their technique is required only to handle the nodes of very low capacity; if they were willing to discard low-capacity nodes as we do, the trivial Basic-VSS scheme of Section 3.2.3 would be acceptable. In contrast, we cluster a node's IDs in order to share overlay links. Moreover, the way in which the ID space sharing is performed in [15] is more complicated than in our scheme; notably, nodes need $\Theta(\log^2 n)$ IDs, rather than $\Theta(\log n)$.

Load balance by object reassignment. The above strategies balance load by changing the assignment of IDs to nodes. Another approach is redirection: store a pointer from an object's ID to the arbitrary node currently storing it. This can balance the load of storing and serving data, but not load due to routing or maintenance of the overlay — and if objects are small, routing dominates the bandwidth and latency of storing and finding an object. It also requires maintenance of the pointers and adds one hop to each lookup.

Karger and Ruhl [64] can handle heterogeneous capacities and obtain a constant-factor load balance. Each node periodically contacts another, and they exchange objects if one node's load is significantly more than the other's. But their bound on movement cost depends on the ratio of the maximum and minimum node capacities. Byers et al. [17, 18] use a "power of two choices" approach, hashing an object to $d \geq 2$ IDs and storing it on the corresponding node with least load, which results in a maximum load of $\log \log n / \log d + O(1)$. Swart [120] places object replicas on a lightly-loaded subset of nodes in Chord's successor list. Neither [17], [18], nor [120] provide results for the heterogeneous case.

Exploiting heterogeneity in P2P systems. Ratnasamy et al. [96] posed the question of whether heterogeneity could be leveraged to improve routing performance in DHTs. Nearly all the DHTs which have followed up on this suggestion use a two-level hierarchy, dividing nodes into a set of high-capacity "superpeers" and low-capacity peers. Mizrak et al. [86] build a clique of $\sqrt{n}$ superpeers, each with $\sqrt{n}$ low-capacity peers attached. This produces a graph of constant diameter but requires that superpeers have polynomially many neighbors. Its scalability, particularly with respect to maintenance traffic, was not demonstrated for a wide range of capacity distributions. Zhao et al. [133] and Garces-Erice et al. [42] organize peers into groups based on network locality. Each group chooses a superpeer based on reliability, bandwidth, or latency, and the superpeers across all groups form a DHT. We argue that these two-level techniques cannot adapt to or take full advantage of arbitrary node capacity distributions, because peers within each of the two classes are treated equally. In fact these schemes are complementary to ours: peers at each level of the hierarchy are likely to have nonuniform capacities, so our techniques could be applied to the DHTs used at each level to further improve performance.

To the best of our knowledge, the only heterogeneity-aware DHT designs not employing a two-level hierarchy, other than Y_0, are SmartBoa [56] and Xu et al. [130]. In SmartBoa, nodes are divided into up to 128 levels based on capacity, and a node at level

k maintains roughly $n/2^k$ neighbors. A heterogeneity-aware multicast algorithm allows efficient dissemination of node join or leave events, so nodes need not continually ping their neighbors, enabling much larger routing tables and thus lower route length than traditional DHTs. However, SmartBoa does not adapt the ID space partitioning (*i.e.*, object storage load) to nodes' capacities, nor was it evaluated experimentally or theoretically. Xu et al [130] adapt the DHT topology to fit the underlying network connectivity, which may be heterogeneous, thus obtaining low stretch. But the ID space is again not adapted to node capacity.

In the world of unstructured P2P systems, Freenet's Next Generation Routing [25] employs heuristics to route messages to nodes with observed faster responses. Chawathe et al [23] propose an unstructured Gnutella-like system which adapts topology, flow control, and data replication to nodes' capacities, and found that their system performed significantly better in a heterogeneous environment. The techniques used in these systems, and the services they provide, are quite different than those of Y_0.

3.7 Conclusion

This chapter proposed a scheme to assign IDs to virtual servers, called Low Cost Virtual Server Selection (LC-VSS), that yields a simple DHT protocol, called Y_0, for which node degree does not increase significantly with the number of virtual servers. Y_0 adapts to heterogeneous node capacities, can achieve an arbitrarily good load balance, moves little load, and can compute a node's IDs as $O(\log n)$ hashes of its IP address for security purposes. The techniques behind Y_0 generalize to arbitrary overlay topologies while providing some flexibility in neighbor selection, even if the underlying topology did not. We demonstrated the effectiveness of our techniques through simulation, showing a maximum share less than 3.6 in a range of capacity distributions with no greater degree than in Chord with a single virtual server.

Y_0 has several drawbacks. It uses $\Theta(\log^2 n)$ memory per node, but we expect this will be acceptable in nearly all circumstances. If a particularly good balance is desired, the number of links needed to maintain the ring structure of the DHT increases by a constant factor. Node join and leave operations will be slowed somewhat by the fact that a node owns $\Theta(\log n)$ virtual servers, although the overlay links need to be constructed only once. Perhaps most significantly, Y_0 requires good estimates of n and the average capacity, and a

good balance is guaranteed only under the uniform load assumption.

We also illustrated the potential of taking heterogeneity into account, with route lengths in our DHT in a real-world capacity distribution less than half those of a homogeneous distribution due to our adaptation of the Chord overlay topology to a heterogeneous environment. An interesting open problem is to study the effect of heterogeneity in other DHT topologies, such as de Bruijn graphs [62, 87].

Chapter 4

Minimizing Churn in Distributed Systems

This chapter is the second to develop techniques for taking advantage of heterogeneity. Unlike Chapter 3, which dealt with heterogeneity in *capacity* like bandwidth or processor speed, we now address heterogeneity in *reliability*: distributed systems typically have widely varying time-to-failure across their components, and across time for one particular component.

A pervasive requirement of distributed systems is to handle churn: change in the set of participating nodes due to joins, graceful leaves, and failures. A high churn rate can increase costs or decrease service quality. This chapter studies how to reduce churn by selecting which subset of a set of available nodes to use.

First, we provide a comparison of the performance of a range of different node selection strategies in five real-world traces. One of our key findings is that the simple strategy of picking a uniform-random replacement whenever a node fails performs surprisingly well, a result which is consistent across the wide range of traces we examine. We explain this effect through analysis in a stochastic model, showing that random replacement performs better as failure patterns become more heterogeneous.

Second, we show that a class of strategies, which we call "Preference List" strategies, arise commonly as a result of optimizing for a metric other than churn, and produce high churn relative to more randomized strategies under realistic node failure patterns. Using this insight, we demonstrate and explain differences in performance for designs that

incorporate varying degrees of randomization. We give examples from a variety of protocols, including anycast, overlay multicast, and distributed hash tables. In many cases, due to the widespread heterogeneity of failure patterns, simply adding some randomization can go a long way towards reducing churn.

4.1 Introduction

Almost every distributed system has to deal with churn: change in the set of participating nodes due to joins, graceful leaves, and failures. There is a price to churn which may manifest itself as dropped messages, data inconsistency, increased user-experienced latency, or increased bandwidth use [75,100]. Even in peer-to-peer systems which were designed from the outset to handle churn, these costs limit the scenarios in which the system is deployable [13]. And even in a reasonably stable managed infrastructure like PlanetLab [10], there can be a significant rate of effective node failure due to nodes becoming extremely slow suddenly and unpredictably [99].

In this chapter, we study how to reduce the churn rate by intelligently selecting nodes. Specifically, we consider a scenario in which we wish to use k nodes out of $n \geq k$ available. How should we select which k to use in order to minimize churn among the chosen nodes over time? This question arises in many cases, such as the following:

- Running a service on PlanetLab in which $n = 500$ nodes are available and we would like $k \approx 20$ to run the service in order to have sufficient capacity to serve requests.
- Selecting a reliable pool of $k \approx 1000$ super-peers from among $n \approx 100,000$ end-hosts participating in a peer-to-peer system.
- Choosing k nodes to be nearest the root of an overlay multicast tree, where failures are most costly.
- In a storage system of n nodes, choosing k nodes on which to place replicas of a file.

To better understand the impact of node selection on churn, we study a set of strategies that we believe are both relevant in practice, and provide a good coverage of the design space.

At the high level, we classify the selection strategies along two axes: (1) whether they use information about nodes to attempt to predict which nodes will be stable, and (2)

whether they replace a failed node with a new one. We refer to strategies that base their selection on individual node characteristics (*e.g.*, past uptime or availability) as **predictive** strategies, and ones that ignore such information as **agnostic**. On the second axis, we use the term **fixed** for strategies that never replace a failed node from the original selected set, and **replacement** for strategies that replace a node as soon as it fails, if another is available.

Predictive fixed strategies are often used in the deployment of services on PlanetLab, where typically developers pick a set of machines with acceptable past availability, and then run their system exclusively on those machines for days or months. *Predictive replacement* strategies appear in many protocols that try to dynamically minimize churn. The most common heuristic is to select the nodes which have the longest current uptime [42, 72, 110].

Agnostic strategies can frequently describe systems which do not explicitly try to minimize churn. The simplest form of *Agnostic Replacement* strategy is **Random Replacement (RR)**: replace a failed node with a uniform-random available node. Another important form of agnostic replacement strategy is what we call a **Preference List** *(PL)* strategy, which arises as a result of optimizing for a metric other than churn: rank the nodes according to some preference order, and pick the top k available nodes. Note that we use the term PL specifically in the case that the preference order is *not* directly related to churn (*e.g.*, latency), and is essentially static. Such PL strategies turn out to describe many systems well. One example of a PL strategy is anycast, where one client aims to select the closest available server(s).

Results

Basic evaluation of strategies. The first part of the chapter performs an extensive evaluation of churn resulting from a number of node selection strategies in five real-world traces. Among our conclusions is that replacement strategies yield a 1.3-5$\times$ reduction in churn over the best fixed strategy in the longer traces, intuitively because of their ability to dynamically adapt. This indicates that for some systems, dynamic node reselection strategies may be worthwhile even if they are more difficult to implement.

A more surprising finding is that there is a significant difference in churn among agnostic strategies. One might expect that selecting nodes using a metric unrelated to churn should perform similar to RR, since neither strategy uses node-specific stability in-

formation. However, it turns out that while PL strategies perform poorly, RR is quite good, *typically within a factor of less than* 2 *of the best predictive strategy.*

To explain the low churn achieved by RR, we analyze it in a stochastic model. While with an exponential session time distribution, RR is no better than Preference Lists, RR's churn rate decreases as the distributions become more heterogeneous—and session time distributions tend to be heterogeneous in realistic scenarios.

Applications to systems design. In the second part of this chapter, we explore systems in which different designs or parameter choices "accidentally" induce a PL or RR-like strategy. Consider constructing a multicast tree as follows: each node, upon arrival or when one of its ancestors in the tree fails, queries m random nodes in the system, and connects to the node through which it has the lowest latency to the root. Clearly, increasing m better adapts the tree to the underlying topology, but it also has the nonobvious result that *the tree can suffer from more churn as* m *increases*, as node selection moves from being like RR to being like a PL strategy.

Of course, there will always be a tradeoff between churn and other metrics. What we aim to illuminate is the nonobvious way in which that tradeoff arises. Although this is a simple phenomenon at heart, to the best of our knowledge it has not been studied in the context of distributed systems. This framework can explain previously observed performance differences in new ways, and provide guidance for systems design.

Contributions

In summary, the main contributions of this chapter are as follows:

- We provide a quantitative guide to the churn resulting from various node selection strategies in real-world traces.
- We demonstrate and analytically characterize the performance of Random Replacement, showing that it is better than Preference List strategies and in many cases reasonably close to the best strategy. Its simplicity and acceptable performance may make RR an appropriate choice for certain systems.
- Using the difference between RR and PL, we demonstrate and explain performance differences in existing designs for the topology of DHT overlay networks, replica placement in DHTs, anycast server selection, and overlay multicast tree construction.

In many protocols, simply adding some randomization is an easy way to reduce churn.

This chapter proceeds as follows. Section 4.2 evaluates churn under various selection strategies. In Section 4.3, we give intuition for and analysis of RR and PL strategies. Section 4.4 explores how the difference between RR and PL affects system design. We discuss why one would intentionally use RR in Section 4.5 and related work in Section 4.6.1, and conclude in Section 4.7.

4.2 Churn Simulations

The goal of this section is to understand the basic effects of various selection strategies in a wide variety of systems and node availability environments. To this end, we use a simple model of churn which will serve as a useful rule of thumb for metrics of interest in real systems. We show one such metric later in this section—the fraction of failed route operations in a simulation of the Chord DHT [114]—and we will see others in more depth in Section 4.4.

In Section 4.2.1 we give our model of churn. We list the node selection strategies in Section 4.2.2 and the traces of node availability in Section 4.2.3. Section 4.2.4 presents our simulation methodology. Our results appear in Section 4.2.5.

4.2.1 Model

In this section we define churn essentially as the rate of turnover of nodes in the system. Intuitively, this is proportional to the bandwidth used to maintain data in a load-balanced storage system.

System model. At any time, each of n nodes in the system is either *up* or *down*, and nodes that are up are either *in use* or *available*. Nodes fail and recover according to some unknown process. We call a contiguous period of being up a *session* of a node. At any time, the node selector may choose to add or remove a node from use, transitioning it from *available* to *in use* or back. There is a target number of nodes to be in use, $k = \alpha n$ for some $0 < \alpha \leq 1$, which the replacement strategies we consider will match exactly unless there are fewer than k nodes up. The fixed strategies will pick some static set of k nodes, so they will have fewer than k in use whenever any picked node is down.

Definition of churn. Given a sequence of changes in the set of in-use nodes, let U_i be the set of in-use nodes after the ith change, with U_0 the initial set. Then churn is the sum over each event of the *fraction of the system that has changed state* in that event, normalized by run time T:

$$C = \frac{1}{T} \cdot \sum_{\text{events } i} \frac{|U_{i-1} \ominus U_i|}{\max\{|U_{i-1}|, |U_i|\}},$$

where $\ominus$ is the symmetric set difference. We count a failure, and the selector's response to that failure, as separate events. So in a run of length T, if we begin with k nodes in use, two nodes fail simultaneously, and the selector responds by adding two available nodes, churn is $\frac{1}{T}\left(\frac{2}{k} + \frac{2}{k}\right)$. If each of the k in-use nodes fails, one by one with no reselections, churn is $\frac{1}{T}\left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}\right) \approx \frac{1}{T} \ln k$.

An important assumption in this definition is that a node which fails and then recovers is of no more use to us than a fresh node. This is reasonable for systems with state that is short-lived relative to the typical period of node downtime, such as in overlay multicast or *i*3 [113]. We study the case of storage systems, which have long-term state, in Section 4.4.4.

4.2.2 Selection Strategies

Predictive Fixed strategies. When deploying a service on a reasonably static infrastructure such as PlanetLab, one could observe nodes for some time before running the system, and then use any of the following heuristics for selecting a "good" fixed set of nodes to use for the lifetime of the system, whenever they are up:

- *Fixed Decent*: Discard the 50% of nodes that were up least during the observation period. Pick k random remaining nodes. (If $k > \frac{n}{2}$, then pick all the remaining nodes and $k - \frac{n}{2}$ random discarded nodes.)
- *Fixed Most Available*: Pick the k nodes that spent the most time up.
- *Fixed Longest Lived*: Pick the k nodes which had greatest average session time.

It would be natural to try picking the k nodes that result in minimal churn during the observation period, but unfortunately this problem is NP-complete (see Appendix C.4).

The complexity arises from the property that the cost of a failure depends on the number of nodes in use at the time.

Agnostic Fixed strategies. We look at only a single strategy in this class, which will turn out to be interesting because its performance is similar to Preference List strategies:

- *Fixed Random*: Pick k uniform-random nodes.

Predictive Replacement strategies. The following strategies select a random initial set of k nodes, and pick a replacement only after an in-use node fails. They differ in which replacement they choose:

- *Max Expectation:* Select the node with greatest expected remaining uptime, conditioned on its current uptime. Estimate this by examining the node's historical session times.
- *Longest Uptime:* Select the node with longest current uptime. This is the same as Max Expectation when the underlying session time distribution has decreasing failure rate.
- *Optimal:* Select the node with longest time until next failure. This requires future knowledge, but provides a useful comparison. It is the optimal replacement strategy (see Appendix C.3).

Agnostic Replacement strategies.

- *Random Replacement (RR)*: Pick k random initial nodes. After one fails, replace it with a uniform-random available node.
- *Passive Preference List:* Given a ranking of the nodes, after an in-use node fails, replace it with the most preferable available node.
- *Active Preference List:* Given a ranking of the nodes, after an in-use node fails, replace it with the most preferable available node. When a node becomes available that's preferable to one we're using, switch to it, discarding the least preferable in-use node.

In this section, we will assume a randomly ordered preference list chosen and fixed at the beginning of each trial.

4.2.3 Traces

The traces we use are summarized in Table 4.1 and described here.

Synthetic traces: We use session times with PDF $f(x) = ab^a/(x+b)^{a+1}$ with exponent $a = 1.5$ and b fixed so that the distribution has mean 30 minutes unless otherwise stated. This is a standard Pareto distribution, shifted b units (without the shift, a node would be guaranteed to be up for at least b minutes). Between each session we use exponentially-distributed downtimes with mean 2 minutes.

PlanetLab All Pairs Ping [115]: this data set consists of pings sent every 15 minutes between all pairs of 200-400 PlanetLab nodes from January, 2004, to June, 2005. We consider a node to be up in one 15-minute interval when at least half of the pings sent to it in that interval succeeded. In a number of periods, all or nearly all PlanetLab nodes were down, most likely due to planned system upgrades or measurement errors. To exclude these cases, we "cleaned" the trace as follows: for each period of downtime at a particular node, we remove that period (i.e. we consider the node up during that interval) when the average number of nodes up during that period is less than half the average number of nodes up over all time. We obtained similar results without the cleaning procedure.

Web Sites [8]: This trace is based on HTTP requests sent from a single machine at Carnegie Mellon to 129 web sites every 10 minutes from September, 2001, to April, 2002. Since there is only a single source, network connectivity problems near the source result in periods when nearly all nodes are unreachable. We attempt to remove such effects using the same heuristic with which we cleaned the PlanetLab data.

Microsoft PCs [14]: 51,662 desktop PCs within Microsoft Corporation were pinged every hour for 35 days beginning July 6, 1999.

Skype superpeers [52]: A set of 4000 nodes participating in the Skype superpeer network were sent an application-level ping every 30 minutes for about 25 days beginning September 12, 2005. As in the web sites trace, there are a number of short periods when many nodes appear to fail, due to network problems near the measurement site.

Gnutella peers [106]: Each of a set of 17,125 IP addresses participating in the Gnutella peer-to-peer file sharing network was sent a TCP connection request every 7 minutes for about 60 hours in May, 2001. A host was marked as up when it responded with a SYN/ACK within 20 seconds, indicating that the Gnutella application was running. The majority of those hosts were usually down (see Table 4.1).

Trace	Length (days)	Mean # nodes up	Median node's mean session time
PlanetLab	527	303	3.9 days
Web Sites	210	113	29 hours
Microsoft PCs	35	41970	5.8 days
Skype	25	710	11.5 hours
Gnutella	2.5	1846	1.8 hours

Table 4.1: The real-world traces used in this chapter. The last column says that 50% of PlanetLab nodes had a mean time to failure of $\geq$ 3.9 days.

4.2.4 Simulation Setup

We compute churn in an event-based simulator which processes transitions in state (*down*, *available*, and *in use*) for each node. We allow the selection algorithm to react immediately after each change in node state. This is a reasonable simplification for applications which react within about 7 minutes, since the time between pings used to produce the traces is at least this much.

We also feed the sequence of events (transitions to or from the *in use* state) into a simple simulator of the Chord protocol included with the *i*3 [113] codebase. Events are node joins and failures and datagrams being sent and received. Datagram delivery is exponentially distributed with mean 50 ms between all node pairs with no loss (unless the recipient fails while the datagram is in flight). Once per simulated second we request that two random DHT nodes v_1, v_2 each route a message to the owner of a single random key k. The trial has *failed* unless both messages arrive at the same destination. Failure due to message loss was about an order of magnitude more common than failure due to inconsistency (the messages being delivered to two different nodes).

In all cases, we split each trace in half, train the fixed strategies on the first half, simulate the strategies on the whole trace, and report statistics on the second half only. All plots use at least 10 trials and show 95% confidence intervals unless otherwise stated. For the traces with more than 1000 nodes, we sample 1000 random nodes in each trial.

In the real-world traces, the parameter k does not directly control system size for the fixed strategies, since some nodes have extended downtimes. To provide a fairer comparison, we plot performance as a function of the *average number of nodes in use over time*, controlled behind the scenes by varying k. Replacement strategies have an advantage

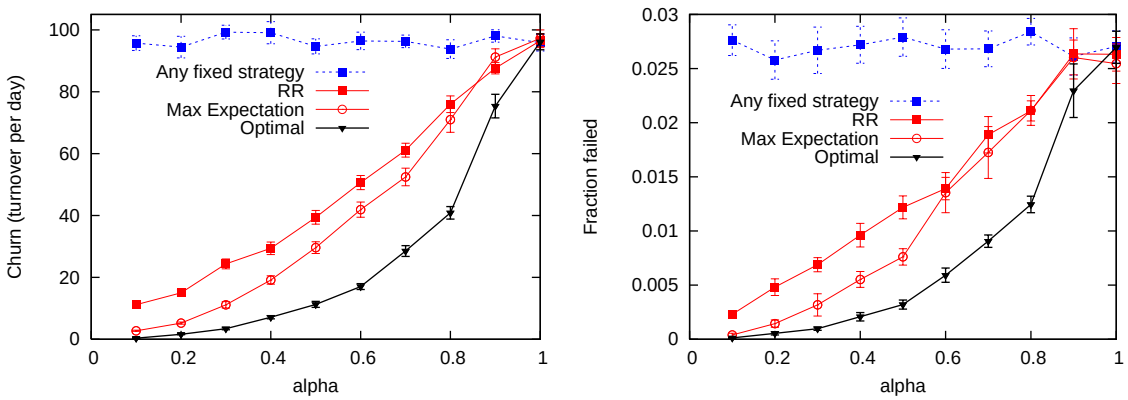


Figure 4.1: Churn (left) and fraction of requests failed in Chord (right) for varying α, with fixed $k = 50$ nodes in use and the synthetic Pareto lifetimes.

that this metric doesn't capture: the number of nodes in use is exactly k as long as $\geq k$ nodes are up.

4.2.5 Results

The results of this section are shown in Figures 4.1-4.5. Note that for clarity in the plots, we have shown the Preference List strategies separately (Figure 4.5).

Some basic properties

Figure 4.1 shows churn in the synthetic Pareto session times as a function of α with fixed k, so that $n = k/\alpha$ varies. Here all fixed strategies are equivalent: since all nodes have the same mean session time, it is not possible to pick out a set of nodes that is consistently good. We can also see that Random Replacement is close to Max Expectation when α is large. As one would expect, performance is best when $\alpha \ll 1$. In this case, Max Expectation does much better than RR intuitively because it finds the few nodes with very long time to failure.

By comparing the two plots in Figure 4.1, we can see that churn is proportional to the fraction of requests failed in Chord. This demonstrates that our churn metric can predict performance in at least one real system; we will study other systems in Section 4.4.

Figure 4.2 shows a difference in performance for the PlanetLab trace, rather than the synthetic Pareto trace of Figure 4.1. In Figure 4.2, we vary k (the number of nodes in

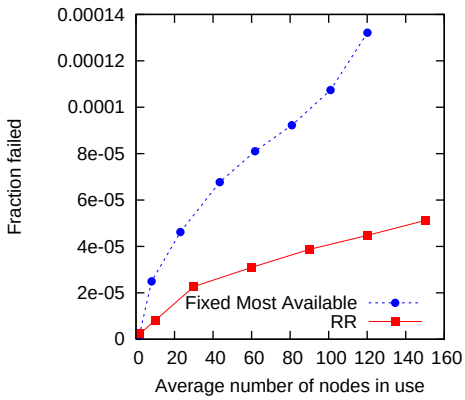


Figure 4.2: Chord in the PlanetLab trace (one trial per data point).

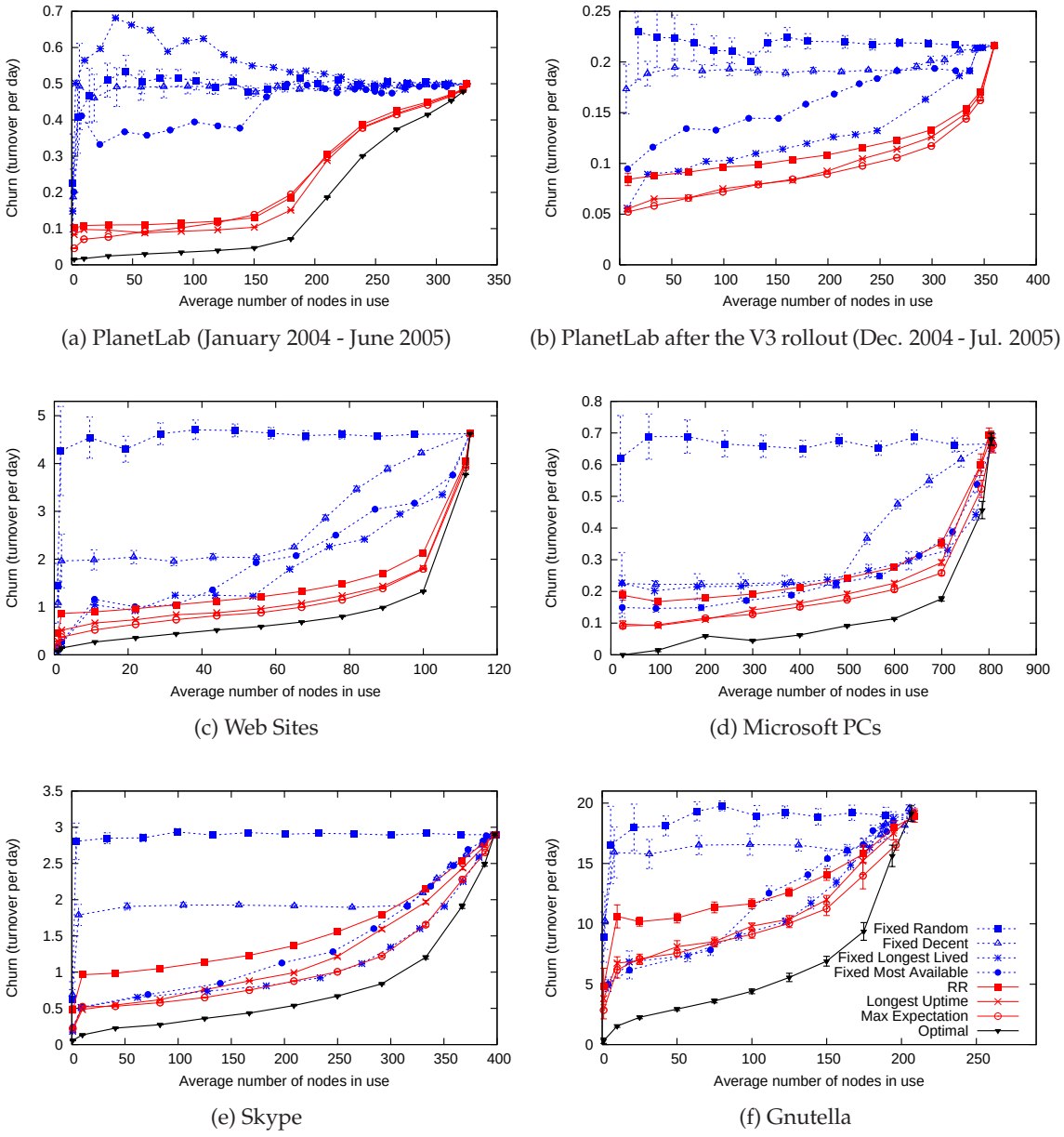
use) rather than n, which results in more failures as k grows since route lengths increase as $O(\log k)$. Not shown is that RR results in 3.3% lower mean message latency in Chord in the PlanetLab trace. We will see how churn affects other systems in Section 4.4.

Benefit of Replacement over Fixed strategies

Figure 4.3 shows that in the two peer-to-peer traces, the best fixed strategies match the performance of the best replacement strategies, perhaps since these traces are shorter than the others (Table 4.1). In any case, fixed strategies are less applicable in a peer-to-peer setting due to the dynamic population.

In the other traces, the best replacement strategies offer a 1.3-5$\times$ improvement over the best fixed strategy, depending on k and the trace. This suggests that dynamically selecting nodes for a long-running distributed application would be worthwhile when churn has a sufficient impact on cost or service quality.

In the PlanetLab trace, the fixed strategies are particularly poor. This is primarily due to a period of uncharacteristically high churn from late October until early December, 2004, coinciding with the PlanetLab V3 rollout. During this period, fixed strategies had an order of magnitude higher churn than at other periods, while the replacement strategies increased by only about 50%. While this is impressive on the part of the replacement strategies, the rollout period may not be representative of PlanetLab as a whole. Restricting the simulation to the 6-month period after the rollout (Figure 4.3(b)), the smart fixed strategies offer some benefit, and there is less separation between strategies in general. However, all



(a) PlanetLab (January 2004 - June 2005)

(b) PlanetLab after the V3 rollout (Dec. 2004 - Jul. 2005)

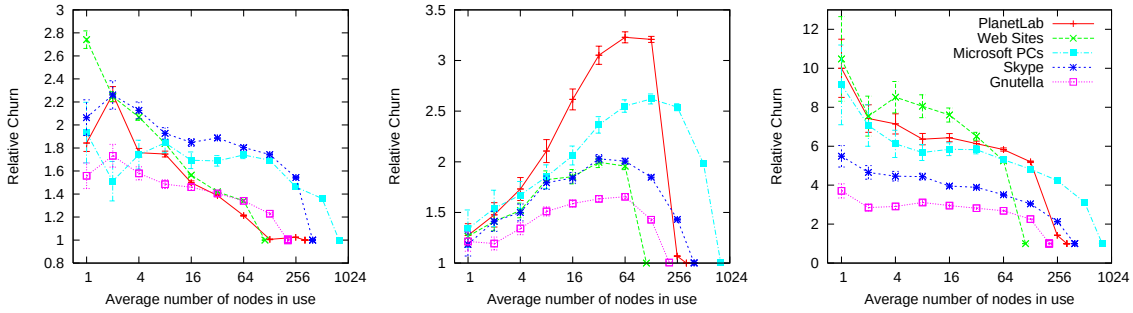
(c) Web Sites

(d) Microsoft PCs

(e) Skype

(f) Gnutella

Figure 4.3: Churn with varying average number of nodes in use traces. The key at lower right applies to all six plots.



(a) RR divided by Max Expectation (b) Passive PL divided by RR (c) Active PL divided by RR

Figure 4.4: Churn of Random Replacement relative to other strategies. The key at right applies to all three plots.

of the replacement strategies are still more effective than the best fixed strategy.

Agnostic strategies

Figure 4.4(a) shows the churn of Random Replacement divided by the churn of Max Expectation, the overall best strategy (other than Optimal, which requires knowledge of the future). As in the synthetic distributions, RR's relative performance is worse for small k, but is usually within a factor of 2 of Max Expectation.

Figure 4.5 illustrates the general behavior of the Preference List strategies via the PlanetLab trace. Active PL is similar to, and worse than, Fixed Random. Intuitively, this is because both strategies pay for every failure that occurs on a fixed set of k nodes. Additionally, according to our definition of churn, Active PL pays to add preferred nodes as soon as they recover. Passive PL becomes more similar to Fixed Random as k increases. While it doesn't pay for every failure on the top k nodes, it is usually using those nodes and pays for most of the failures.

Figures 4.4(b) and (c) show churn under the Passive and Active PL strategies, respectively, divided by the churn of RR. RR is generally 1.2-3$\times$ better than Passive and 2.5-10$\times$ better than Active PL.

In the next section, we give more precise intuition for—and analysis of—the differing performance of RR and Preference List strategies. In Section 4.4 we will show how that difference affects system design.

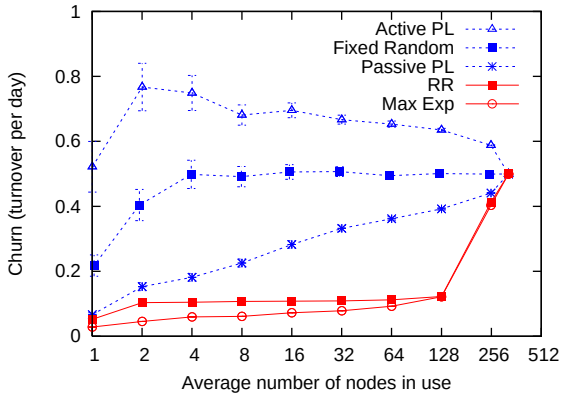


Figure 4.5: Preference List strategies in the PlanetLab trace. Note the log-scale x axis.

4.3 Analysis

Why does picking a random replacement for each failed node produce much lower churn than using a fixed random set of nodes, or the top k nodes on a preference list? We answer that question within a stochastic model defined in Section 4.3.1. We give intuition for why Preference List strategies are as bad as Fixed Random in Section 4.3.2, and why RR does better in Section 4.3.3.

Our main analytical results are in Section 4.3.4. We derive RR's expected churn rate, show that its churn decreases as the session time distributions become more heterogeneous, and show that if all nodes have equal mean session time, RR has no worse than twice the churn of any fixed or Preference List strategy. However, if there are very few nodes with high mean session time, RR can be much worse.

4.3.1 Stochastic Model

We use the following renewal process. For each node v_i, there is a distribution of session times with given PDF f_i and mean μ_i. *To simplify the exposition, we will assume all nodes have equal mean μ unless otherwise specified.* At time 0 all nodes are up. Each node draws a session time ℓ_1 from its distribution independently of all other nodes, fails at time ℓ_1, recovers instantaneously, draws another session time ℓ_2, fails at time $\ell_1 + \ell_2$, and so on until the end of the run at some given time T. We will be interested in the expected churn as $T \to \infty$. (If instantaneous recovery seems unrealistic, we note that the analysis of RR is identical in the model that each node has only a single session, and the total number of

nodes is held constant by introducing a fresh node after each failure.)

4.3.2 Fixed and Preference List Strategies

Fixed strategies are very easy to analyze in this model. Since nodes recover instantaneously, our definition of churn reduces to $\frac{2}{kT}$ times the number of failures ($\frac{1}{k}$ for each failure and $\frac{1}{k}$ for each recovery, normalized by time T). As $T \to \infty$, the number of failures on any node approaches its expected value T/μ, so the total number of failures on the k selected nodes approaches $\frac{Tk}{\mu}$. Thus all fixed strategies result in expected churn $\frac{2}{kT} \cdot \frac{Tk}{\mu} = 2/\mu$.

Now consider Passive PL and suppose S is the set of k most preferred nodes. Like fixed strategies, each failure of some node $v \in S$ causes us to pay $\frac{2}{kT}$ for the failure and replacement. Since recovery is instantaneous, the next time *some other* node fails, v must be its replacement (at any time there will be at most one node in S not in use). As k grows, the rate of failures of in-use nodes grows, so we switch back to v more and more quickly. In particular, the probability that we switch back to v before its next failure approaches 1. Thus, for large k, Passive PL pays for nearly every failure on $\{v_1, \ldots, v_k\}$ and its churn approaches $2/\mu$ also.

Finally, consider Active PL. Like the fixed strategies, it pays for every failure and recovery on its k most preferred nodes, for churn $2/\mu$. But in addition, it pays to switch to a replacement node while one of its most preferred nodes has failed, yielding total churn $3/\mu$.

In summary, fixed strategies, Passive PL, and Active PL have in common the property that they are always or nearly always using a fixed set of k nodes, and hence have failure rate similar to the mean failure rate of those nodes.

4.3.3 Intuition for Random Replacement

RR's good performance is an example of the classic *inspection paradox*. When RR picks a node v_i after a failure, the replacement's time to failure (TTF) is *not* simply drawn from the session time distribution f_i. Rather, RR is (roughly) selecting the *current* session of a random node. (As we will see in the next section, there is some mathematical complexity hidden in the word "roughly".) This is biased towards longer sessions since a node spends longer in a long session than in a short one.

Alternately, consider some node in the system. As it proceeds through a session, the probability that it has been picked by RR increases, simply because there have been more times that it was considered as a potential replacement. Thus, nodes with longer uptimes are more likely to have been picked. And for realistic distributions, nodes with longer uptimes are less likely to fail soon.

But RR does very badly when stable nodes are rare. Suppose $k = 1$ and all nodes have exponential session times, one with mean $r \gg 1$ and $n - 1$ with mean 1. When RR selects a node, its expected time to failure is $\frac{1}{n}(r) + \frac{n-1}{n}(1) \approx 1$ when $n \gg r$, so its churn is 2. But the best fixed strategy has mean TTF r and churn $2/r$.

A rigorous and general analysis of RR takes some more work and is the subject of the next section.

4.3.4 Analysis of Random Replacement

We now derive RR's churn rate in terms of the session time distributions and α, assuming large n and T but not assuming equal means (Theorem 8), and show that the analysis matches simulations even for $n = 20$ (Figure 4.6). From this we show that the churn of RR decreases as the distributions become more heterogeneous (Corollary 8). We will define this rigorously, but as an example, the Pareto distribution becomes more heterogeneous as the exponent parameter a decreases [7]. Finally, we show that for any session distributions that have equal mean, RR has at most twice the expected churn of any fixed or Preference List strategy (Corollary 9).

To simplify the analysis, we assume nodes belong to an arbitrarily large constant number d of groups of n/d nodes, such that the nodes within each group i have the same session time distribution f_i. Additionally, our analysis assumes that the session time distributions have the property that the system converges to a steady state, in the following sense.

Definition 4 *In a run of length T, let random variable L_i be the length of a uniform-random session of node i, and let R_i be the number of reselections (of any nodes) during that session. The session time distributions $f_1, \ldots, f_d$ are* **convergent** *if they have finite mean and variance and there exists a constant $c > 0$ such that*

$$\Pr[(1-\varepsilon)cL_i \leq R_i \leq (1+\varepsilon)cL_i] \geq 1-\varepsilon$$

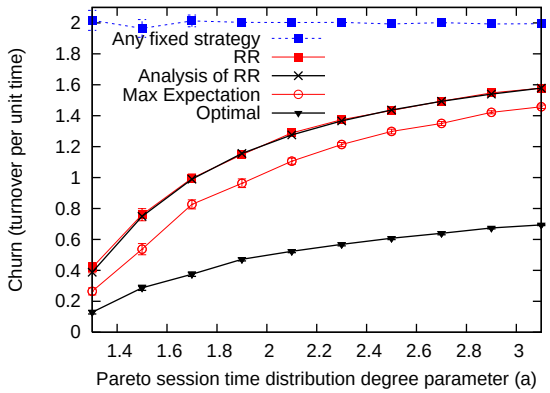


Figure 4.6: Simulation and analysis of churn with varying session time distribution, $n = 20$, and $\alpha = \frac{1}{2}$.

$\forall i, \forall \varepsilon > 0, \alpha \in (0,1)$, *and sufficiently large n and T.*

In other words, perhaps after some period of time, there is a roughly constant rate of c reselections per unit time. This property is trivially true in the (uninteresting) case that all nodes have exponentially distributed session times with common mean. We conjecture that in fact it is true quite generally. Our main analytical result is the following.

Theorem 8 *Let C be the churn in a trial of length T using Random Replacement. If the node session time distributions (f_i) are convergent and $\alpha \in (0,1)$, then as $n, T \to \infty$, $E[C]$ is given by the unique solution to*

$$E[C] = \frac{2}{\alpha d} \sum_{i=1}^{d} \frac{1}{\mu_i} \left(1 - E\left[\exp\left\{-\frac{\alpha}{2(1-\alpha)} E[C] \cdot L_i\right\}\right]\right),$$

where random variable L_i has PDF f_i.

Proof: See Appendix C.1. ■

Figure 4.6 shows agreement of this analysis with a simulation for $n = 20$ and Pareto-distributed session times with PDF $f(x) = ab^a/(x+b)^{a+1}$, as in Figure 4.1. We vary a and pick b so that $\mu = 1$. Even though the analysis assumes large n, it differs from the simulation by $\leq 1.5\%$ for $a \geq 1.5$. As a approached 1, convergence time in the simulation became impractical. For $a \leq 1$, $f(x)$ has infinite variance and does not satisfy the conditions of Theorem 8.

We next characterize the churn of RR in terms of heterogeneous or "skewed" the session time distributions are, in the sense of the Lorenz partial order. Note that this is a generalization to a probabilistic setting of the majorization partial order which we used in Chapter 2.

Definition 5 *Given two random variables $X, X' \geq 0$ with CDFs F and F', respectively, we say $X' \succeq X$ ("X' is more heterogeneous than X") when $E[X'] = E[X] < \infty$, the PDFs of X and X' exist, and for all $y \in [0,1]$,*

$$E\left[X' \mid X' \geq x'\right] \quad \geq \quad E\left[X \mid X \geq x\right],$$

where $x' = F'^{-1}(y)$ and $x = F^{-1}(y)$.

Note that x' and x are the yth percentile values of X' and X, so intuitively this definition compares the tails of the two distributions. The Lorenz partial order is consistent with variance, in the sense that $X' \succeq X$ implies $\mathrm{var}(X') \geq \mathrm{var}(X)$.

Our first corollary states that RR's expected churn decreases as the session time distributions become more heterogeneous. In the framework of Chapter 2 extended to the probabilistic setting, this corollary states that the price of heterogeneity of RR is 1.

Corollary 8 *Let C and C' be the expected churn of RR as given by Theorem 8 under session time distributions (f_i) and (f'_i), respectively, and fixed α. If $f'_i \succeq f_i$ for all $i \in \{1, \ldots, d\}$, then $E[C'] \leq E[C]$.*

Thus, for fixed mean session times, the least heterogeneous distribution—when session times are deterministically equal to their mean—is the worst case for RR. In the special case that all mean session times are equal (but may have different distributions), we have the following:

Corollary 9 *If the session time distributions are convergent and have equal mean, RR's expected churn is at most twice the expected churn of any fixed or Preference List strategy.*

The proofs appear in Appendix C.2.

4.4 Applications

We have seen that Random Replacement consistently outperforms Preference List strategies (Section 4.2.5) by taking advantage of heterogeneous session time distributions

(Section 4.3). In this section, we study how these two classes of strategies come up in real systems.

We begin in Section 4.4.1 with a simple example, anycast server selection, in which there are natural analogies for strategies on the spectrum between RR and PL, and doing *less* work (in terms of optimizing latency) decreases churn. We also study how quickly RR converges to its steady-state churn rate.

In Section 4.4.2 we discuss how two classes of proposed DHT topologies behave like Active PL and RR, and show that randomizing the Chord topology decreases the fraction of failed lookups by 29% in the Gnutella trace.

In Section 4.4.3, we show how strategies similar to RR and PL occur in overlay multicast tree construction. Our results provide further insight into an initially surprising effect observed by [110], that a random parent selection algorithm was better than a certain longest-uptime heuristic.

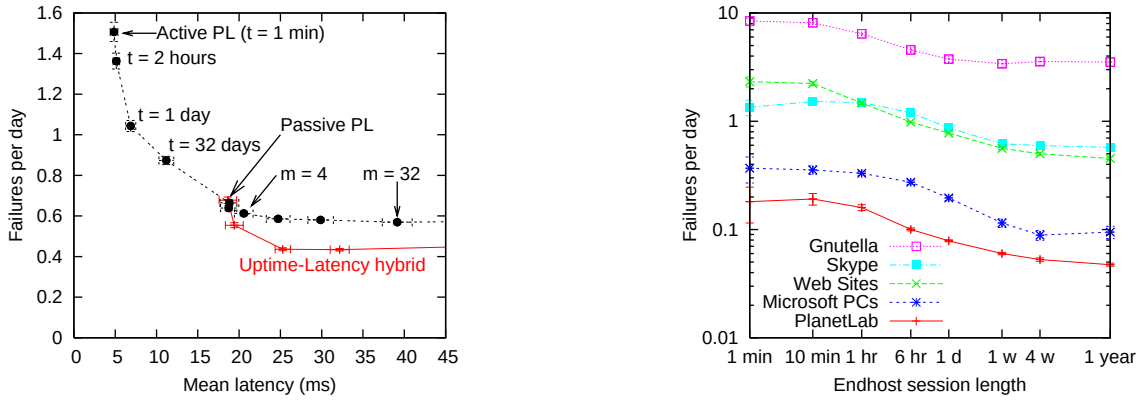
Finally, Section 4.4.4 explores two strategies for placing replicas in DHTs. Although a difference in their associated maintenance bandwidth had been previously observed, we show that part of the performance difference is due to behavior close to RR in one, and PL in the other.

4.4.1 Anycast

To give a simple instantiation of preference list and RR strategies, consider an endhost which desires to communicate with any of a set of n acceptable servers. The endhost begins by connecting to a random server. Whenever its current server fails, it obtains a list of the m servers to which it has lowest latency, perhaps by utilizing an anycast service such as [9, 40, 129], and connects to a random one of these m. Additionally, the endhost periodically probes for a closer server that may have newly joined, switching to such a server after some random delay in $[0, t]$ following the join.

We simulated the resulting number of failures in a simple simulator with events at the level of node joins and failures, as in Section 4.2. We do not count a switch as a failure. Latencies were obtained from a synthetic edge network delay space generator of Zhang et al. [132], which is modeled on measurements of latency between DNS servers. The large availability traces were sampled down to 2000 nodes.

Figure 4.7(a) depicts the tradeoff between server failure rate and latency that re-



(a) Stability-latency tradeoff in Skype trace

(b) Convergence of RR

Figure 4.7: Anycast simulation results.

sults from various choices of the parameters m and t in the Skype trace. The upper-left point has $t = 1$ minute and $m = 1$, and corresponds to an Active PL strategy. As $t \to \infty$, we move to a Passive PL strategy, and failure rate decreases by roughly 56% (46-72% in the other traces). Increasing m results in an RR strategy and decreases failure rate by a further 13% (13-21% in the other traces). This latter decrease is modest since we are only selecting one node at a time (compare with Figure 4.4(b) with $k = 1$ nodes in use). However, this may be useful if, for example, a mean latency of 40 ms were acceptable to the application in question. We also simulated a hybrid strategy which used $t = \infty$ and selected the replacement server which minimized $w \cdot latency - (1 - w) \cdot uptime$. As w decreases from 1 to 0, the strategy moves from Passive PL to Longest Uptime. In the Skype trace, this additional uptime information reduces failure rate by about 24% below the randomized strategy with $m = 32$.

Of course, the right point in the tradeoff space depends on the particular application, but these results show that we should expect stability to suffer as latency is better optimized, and conversely that doing a little *less* work is an easy way to reduce the failure rate.

So far we have assumed an endhost which continually selected a server over the entire trace. Suppose now that the endhost arrives at a random time, uses RR server selection, and departs after a given session length ℓ. Figure 4.7(b) shows that when ℓ is small,

the endhost experiences the mean server failure rate, as in Active PL. Intuitively, the endhost departs before it makes full use of the session of the server it selected. The failure rate converges as ℓ approaches the mean session length of a RR-selected server, decreasing by 2.3×-5.1× depending on the trace.

As an example, some Skype peers which are behind NATs select superpeers through which to relay voice calls. Since 90% of relayed Skype calls last less than 36 minutes [52], if the peers select relays randomly, these calls would see roughly the mean superpeer failure rate (one failure every 16 hours). However, one could imagine designing the superpeer network to maintain a set of randomly selected "super-superpeers" through which interruption-sensitive voice calls are routed when possible. Such a design would result in a failure rate similar to that of a persistent endhost session (one failure every 42 hours).

4.4.2 DHT Neighbor Selection

In a distributed hash table, each node v is assigned an identifier $id(v)$ in the DHT's keyspace. Ownership of the keys is partitioned among the nodes. Each node in a DHT maintains links to certain other nodes as a function of the IDs of the nodes. Generally these come in two types: *sequential* neighbors, such as the successor list in Chord: each node v maintains links to about $\log n$ nodes whose IDs are closest to v's. These are used to maintain consistency of the partitioning of the keyspace among nodes. Second, nodes have *long-distance* neighbors, such as the finger table in Chord, to provide short routes between any pair of nodes. We will compare two different ways of selecting long-distance neighbors.

Deterministic and randomized topologies

In the first class of topologies, used in Chord [114], CAN [95], and others [62, 87], each node v maintains links to the owners of certain other IDs which are a deterministic function of v's ID. For example, Chord's keyspace is $\{0, \ldots, N-1\}$, where $N = 2^{160}$, and node v maintains links called *fingers* to the owners of $id(v) + 2^i \pmod N$ for each $i \in \{0, \ldots, (\log_2 N) - 1\}$. This results in links to $\Theta(\log n)$ distinct nodes, where n is the number of nodes in the system. Each node periodically performs lookup operations to find the current owner of the appropriate key for each of its fingers, updating its links as ownership changes due to node arrivals and departures. In Chord, a key x is owned by

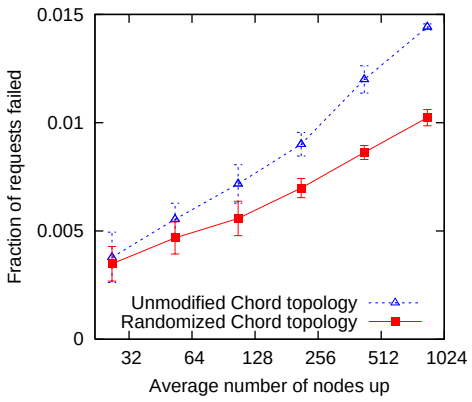


Figure 4.8: DHT neighbor selection simulation in Gnutella trace.

the node whose ID most closely follows x in the (modular) keyspace. Thus the choice of each finger i for a node v can be described as an Active Preference List strategy with $k = 1$ nodes in use, where the preference ordering ranks a node w according to the distance from $id(v) + 2^i$ to $id(w)$.

In the second class of topologies, links are chosen randomly. Symphony [78] was the first design to explicitly choose random neighbors, but some other topologies have enough underlying flexibility [54] that trivial modifications of the original design allow them to choose from many potential long-distance neighbors. For example, a natural way to randomize Chord[1] is to select the ith finger as the owner of a random key in $\{id(v) + 2^i, \ldots, id(v) + 2^{i+1}\}$. When that link fails, we can choose a new random neighbor in the same range. Unsurprisingly, this strategy is essentially RR.

Results

We simulated these two variants of Chord using the simulator and methodology described in Section 4.2.4. In each trial we sampled n random nodes from the Gnutella trace and simulated a run of Chord over those n nodes, with deterministic and random neighbor selection. Since most of the nodes are usually down, we plot results as a function of $\tilde{n}$, the average number of nodes up. Figure 4.8 shows that with $\tilde{n} \approx 850$, the randomized topology has 29% fewer failed requests due to the lower finger failure rate. The randomized topology also had negligibly longer routes (7.6% longer for $\tilde{n} \approx 27$ but decreasing to

[1]This topology was studied in [79] in the context of route length.

just 0.8% longer for $\tilde{n} \approx 850$; and in fact, additional techniques can reduce route length in the randomized Chord topology below that of the deterministic topology [79]).

Implications

The key conclusion we wish to highlight here is that *randomized topologies are inherently more stable than deterministic ones, even without explicitly picking neighbors based on their expected stability.* This fact may be useful in designing systems, as well as teasing out subtle properties of systems design.

For example, Leonard et al. [73] analyzed the resilience of several P2P systems including Chord in a stochastic model, deriving the expected time until a node is disconnected from the network. The analysis assumed that the selection of a neighbor at any point in time is independent of its age—essentially an RR strategy. This might initially seem reasonable since finger selection is based on pseudorandom node IDs. However, we have now seen that deterministic DHTs in fact follow a PL strategy with a pseudorandom preference list. Since time-until-disconnection depends superlinearly on finger failure rate, the assumption of [73] would result in a significant overestimate of the resilience of the standard Chord protocol and the other deterministic DHTs.

We note that other work has dealt with flexible, non-deterministic topologies in DHTs. One advantage of these topologies is the ability to use proximity neighbor selection to reduce latency [54]—which, depending on the implementation, may result in a latency-based PL strategy. In the work most similar to this section, Ledlie et al. [72] used Longest Uptime for finger selection. In the same Gnutella trace, their simulations showed a 42% reduction in maintenance bandwidth when compared to a proximity-optimizing neighbor selection strategy, albeit at the cost of increasing latency by 50%.

4.4.3 Multicast

In this section we simulate how preference list strategies can affect the stability of overlay multicast trees.

Simulation setup

We closely follow the simulation scenario of Sripanidkulchai et al. [110]. We deal with a single-source multicast tree, whose root is always present without failure. When

a node v joins, it contacts m random suitable nodes. A node is *suitable* for v when it is connected to the tree and has available bandwidth for another child. The node then picks one of those m nodes as its parent in the tree, according to one of several strategies we will describe momentarily. Whenever a node fails, each of its descendants experiences an *interruption* in the hypothetical multicast stream, and repeats the join procedure. Thus a failure near the root may disrupt the structure of a large subtree.

We use three strategies for selecting the parent among the m suitable nodes: (1) the node with *Longest Uptime*; (2) the node at *Minimum Depth* from the root; or (3) the node which would result in the *Minimum Latency* along v's path through the tree to the root. The first two strategies with $m = 100$ were also simulated in [110], in traces which had peak sizes of $1,000$-$80,000$ nodes up.

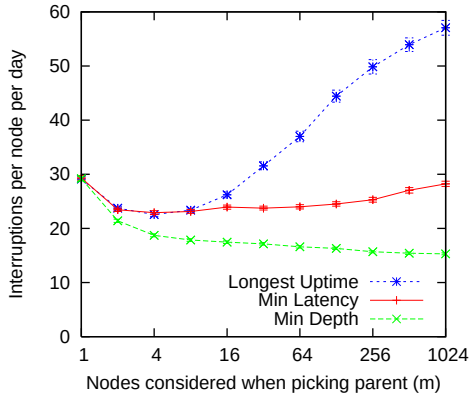
Unfortunately, we could not test under the traces and node bandwidth bounds used in [110] since their data is not publicly available. Instead, we use the traces of Section 4.2.3, latencies from Zhang et al. [132] as in Section 4.4.1, and uniform node capacities: each node accepts at most $d = 4$ children unless otherwise stated. In [110], after a node fails, its descendants which are contributing more resources are allowed to rejoin before freeriders. Since we use homogeneous capacities, we have nodes rejoin in random order. Finally, a minor difference is that we have a node query m suitable parents and pick the best, rather than querying m nodes, filtering out the unsuitable ones, and picking the best.

We report the total number of interruptions. Additionally, we periodically sample the mean node depth (number of hops from each node to the root) and mean latency through the tree to the root. We take the mean of these metrics over all samples within each trial, and then over all trials.

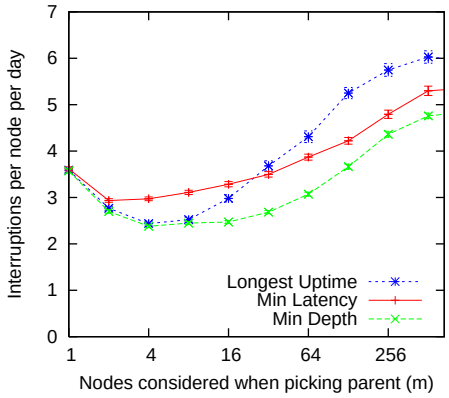
Results

We begin by discussing the Min Latency strategy. We will then confirm and offer additional interpretation of two results of [110] regarding the Min Depth and Longest Uptime strategies.

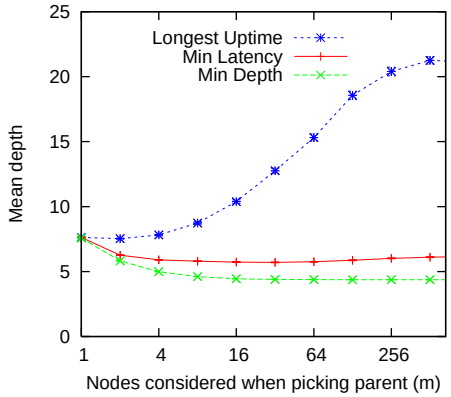
Figures 4.9(a) and 4.9(b) show that optimizing latency both helps and hurts the number of interruptions. The case $m = 1$ is random parent selection. As we begin increasing m, latency to the root decreases (Figure 4.9(d)) but there is a side effect of reducing tree height (Figure 4.9(c)), which reduces the interruption rate (22% in Gnutella, 19% in Skype)



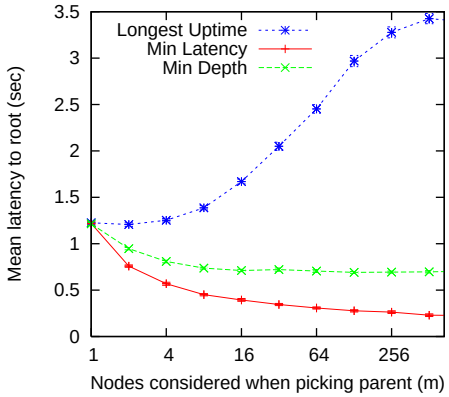
(a) Interruption rate in Gnutella trace



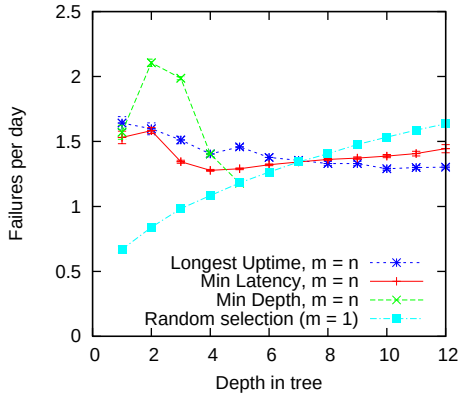
(b) Interruption rate in Skype trace



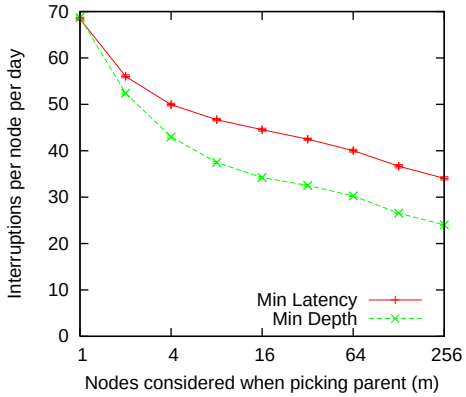
(c) Mean node depth in Skype trace



(d) Mean latency in Skype trace



(e) Node failure rate by depth in Skype trace



(f) Interruption rate in a bad case for RR

Figure 4.9: Multicast simulation results.

because there are fewer opportunities for failure along a node's path to the root. But for $m \geq 4$ the mean node depth is essentially constant and the trees become less stable, with interruptions increasing 22% in Gnutella and 86% in Skype.

The interior structure of the trees reveals the proximate cause of this instability. Figure 4.9(e) shows that smaller m actually results in more stable nodes closest to the root, where failures affect the most descendants, while $m = n$ does a poorer job of getting the best nodes near the root.

We claim that the ultimate cause of this increase in failure rate for the Min Latency strategy is due to the RR/PL effect. The case is not as clear as in the previous examples: even with $m = n$ the trees produced are not deterministic since the nodes re-join in random order after an ancestor fails. However, consider the $\leq d$ children of some node v in the tree. After one of the children fails, eventually a new child will join. With $m = n$ the new child is likely to be a nearby node, while with $m = 1$ the new child is selected more like RR. Then with $m = 1$ we should expect the children of v to be more stable, and hence v's grandchildren will experience fewer interruptions.

To test this hypothesis, if the nodes had session time distributions in which RR performed *worse* than PL strategies, performance should *improve* as $m \to n$. By Corollary 8, such an (unrealistic) bad case is when session times are essentially constant, e.g. uniform in $[9, 11]$. Figure 4.9(f) shows that in this case, interruption rate is indeed a monotonically decreasing function of m.

We now compare our results to two results of Sripanidkulchai et al [110]. First, in tests using a fixed m, they found that Min Depth best optimized stability among the strategies they tested, which is true in most cases we tested (e.g. all of Figure 4.9(a)). Interestingly, we find that even Min Depth can benefit from some randomization as well, with less than half the interruption rate at $m = 4$ than $m = n$ in Skype. This effect also appeared in the Microsoft PCs trace and to a lesser extent in PlanetLab, but not in Gnutella or Web Sites.

Second, Sripanidkulchai et al. [110] found it surprising that the Longest Uptime parent selection performed more poorly than random selection ($m = 1$) in many cases, and they determined the cause was that it built much taller trees. We obtained similar results in Figure 4.9(a,b) for sufficiently large m. However, we also find that using Longest Uptime, the nodes near the root are less stable than in the $m = 1$ case (Figure 4.9(e)).

In fact, neither Min Depth nor LU optimizes exactly the right metric. Min Depth

ignores the stability of nodes on the path from the parent to the root, and LU ignores the length of that path and the stability of all ancestors except the parent. Thus, given the results of this chapter, it should not be surprising that random selection (which, rather than being agnostic, does a decent job optimizing for the *right* metric) can be better than the other heuristics.

4.4.4 DHT Replica Placement

In this section we compare two common strategies, *Root Set* and *Random,* for managing file replicas in distributed hash table-based storage systems. The metric we study is the rate at which new replicas are created, which directly affects maintenance bandwidth.

Replica management strategies

In DHT-based storage systems, nodes are assigned identifiers (IDs) in a keyspace. Each stored file or object o is also assigned a key $key(o)$. The node whose identifier most closely follows $key(o)$ serves as the object's coordinator or *root* $r(o)$. For redundancy, some number k of replicas of o are stored on some set of nodes.

The *Root Set* strategy for placing those replicas is used in slightly varying forms by DHash [27], PAST [103], Bamboo [100], and Total Recall [11], among others: put replicas on the k nodes whose IDs most closely follow $key(o)$, or the "root set". Specifically, when a node in the root set fails, we add the next closest to the set, causing one replication; when a node joins with an ID that places it in the root set, a replica of o is sent to it. In both cases a file transfer is not necessary if the node in question already has the file. This occurs when a node returns after a *transient* failure, such as a network outage, which does not affect files stored on disk.

The *Random* strategy is used by Pond [98], Total Recall [11], and Weatherspoon et al [127]. The root $r(o)$ stores a directory of all available replicas of o, which may be on any node in the system. (The directory is assumed to be small relative to the size of an object replica, so the cost of replicating it — with, for example, the Root Set strategy — is negligible.) Random has two parameters, k and $f \in (0, 1]$. Repair is only initiated when the number of available replicas falls below $\lceil f \cdot k \rceil$, at which point new replicas are created until k are available. This "lazy replication" provides a buffer so the system reacts to transient failures less frequently.

Simulation setup

It has been previously observed [11, 127] that Random significantly outperforms Root Set, and this has been attributed to a number of disadvantages of Root Set. Root Set might "forget" about replicas that end up outside the root set; it replicates when nodes arrive, rather than only in response to failures; and in some implementations it lacks the lazy replication threshold f.

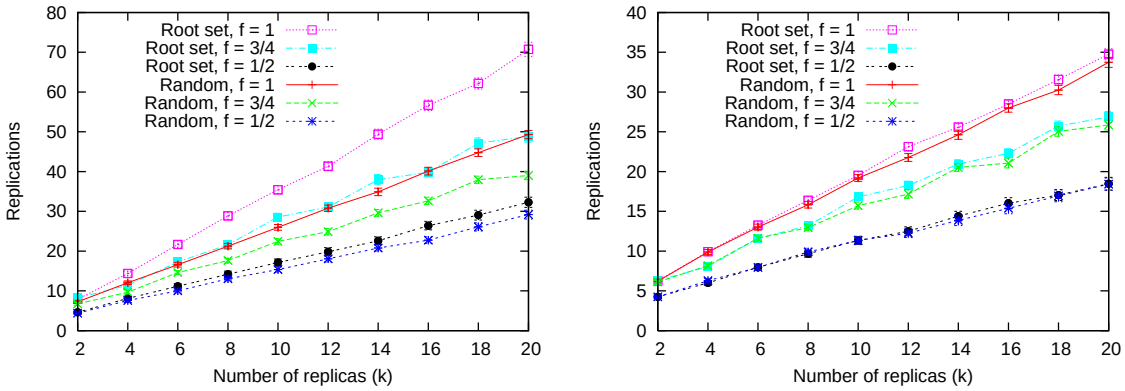
Our goal is to quantify the impact of another difference: the choice of node on which to place a replica once it is created. To compare the strategies on equal footing, we modify the Root Set strategy so that it monitors all replicas in the system, does not replicate in response to node joins, and uses the lazy replication threshold f. The remaining difference is that Root Set places each new replica on the first node available node in the root set (i.e., Passive PL) while Random follows RR node selection.

As before we use a simulator with events at the level of node joins, node failures, and file replications. Since our traces do not include information about data loss associated with failures, we assume no data loss, which provides a lower bound on the permanent failure rate. We assume files are written to the system at the beginning of each trial. We measure the mean number of replications used to maintain each file after the initial write.

Results

Figure 4.10 compares the two strategies with $f \in \{1, \frac{3}{4}, \frac{1}{2}\}$ in two representative traces, PlanetLab and Gnutella, for $k \in \{2, \ldots, 20\}$. (Note that each "replica" may be an erasure-coded fragment of the file, so $k = 20$ is reasonable; in fact, Pond uses $k = 32$.) At $f = 1$ and $k = 20$ in Gnutella, Random requires 30% fewer replications than Root Set, and in fact Random with $f = 1$ is as good as Root Set with $f = \frac{3}{4}$. However, this difference diminishes as f is decreased, and the strategies differ little in PlanetLab.

Several limitations of the traces likely underestimate the long-term benefit of Random over Root Set. Once transient failures are largely masked, the strategies compete at the level of permanent failures. However, none of the traces is long relative to the permanent failure rate. For example, among Gnutella nodes that were up at some point in the first half of the trace, only 33% were absent in the second half, and that fraction was smaller in the other traces. Additionally, we have underestimated the permanent failure rate by assuming no data loss. As a consequence, it is likely that Random has not yet converged



(a) Maintenance cost in Gnutella trace

(b) Maintenance cost in PlanetLab trace

Figure 4.10: Replica placement simulation results.

in these simulations (see Section 4.4.1).

Tati and Voelker [121] observed an effect of the Random strategy: nodes with higher average availability will be selected to receive objects more frequently, and will also likely have higher average availability in the future. This effect is closely related to RR (compare with the intuition in Section 4.3.3) and undoubtedly contributes to the difference between strategies that we have observed. Separating these effects, as well as obtaining better data on permanent failures, remains an interesting area of future research.

4.5 Discussion

When would one use Random Replacement?

As we have seen in Section 4.4, RR appears in a variety of real systems, in sometimes subtle ways. Our results are thus useful in better describing the performance of those systems.

However, if a system designer were intentionally implementing node selection to minimize churn, the results of Section 4.2 show that Longest Uptime offers somewhat better performance. Is there any case in which one would intentionally pick RR?

There are several cases in which RR would be easier to implement and may offer a better tradeoff between churn and system complexity. For example, when failures are due to the network, it may be hard for a node v to determine when it has "failed" and thus report its uptime. If v notices a dropped connection to some other node w, this may be due

to the departure of w or a problem on the network path between v and w.

Even when it is easy to determine the uptime of a node, there may be incentive for nodes to lie about their uptime to obtain better service, such as faster file transfer in a P2P file distribution system or placement closer to the root in an overlay multicast tree. In such cases, RR would be more robust to misbehavior than LU.

Finally, if we are dealing with a protocol that has already been standardized, there may be no support for querying a node's uptime. A client could potentially implement RR node selection—to pick DNS servers, for example—without support from the protocol and without active probing, and still obtain reasonable stability.

What about load balance?

In all effective node selection strategies, including RR, stable nodes are used more on average. What performance can we expect when the most stable nodes are sought after simultaneously by multiple agents, such as peers in a P2P system or users in a shared infrastructure like PlanetLab?

The parameter α, the fraction of nodes needed, gives a way to analyze the total churn experienced by all users: we can take α to be the utilization of the distributed system as a whole. However, our results do not address fairness between users, which we leave to future work.

4.6 Related Work

4.6.1 Page Replacement

In the special case of instantaneous recovery times, there is a precise correspondence between our model of churn (Section 4.2.1) and page replacement in a two-level memory system: each page is a machine; the pages that are *not* in cache are the set of in-use machines; and a page access corresponds to a node failure and instantaneous recovery. Churn is thus twice the number of page faults. What we call Longest Uptime is then exactly the pervasive Least Recently Used (LRU) policy, and Random Replacement is known by the same name. It is well known that in practice, LRU typically results in fewer page faults than RR, although RR performs better in the special case that memory references cycle through an array which does not fit in cache. RR has been implemented in a limited

number of systems, such as the i860 processor [101].

There has been a substantial amount of work on analysis of page replacement algorithms including LRU and RR; see e.g. [29,38,39] and the discussion in [38]. Stochastic analysis of page replacement algorithms has generally been limited to the "independent reference model" in which one page P_t is accessed in each timestep t, where the (P_t) are i.i.d. This corresponds to the special case of our model in which node session times are exponentially distributed (with possibly unequal means). Thus a major difference is that our model analyzed in Section 4.3 is not limited to memoryless session times.

4.6.2 Heuristics for Distributed Systems

Longest Uptime is a common heuristic which has been studied in contexts including DHT neighbor selection [72], selecting superpeers [42], and selecting parents in an overlay multicast tree [110]. The Accordion DHT [74] selects neighbors by computing the conditional probability that a node is currently up given when it was last contacted and how long it was up before that, assuming session times fit a Pareto distribution with learned parameters. Mickens [85] used sophisticated statistical techniques to predict future node uptime, and experimented with placing file replicas in Chord on successors with greatest predicted time to live.

Leonard et al. [73] analyzed the resilience of DHTs in a stochastic model, under the assumption that selection of a neighbor at any point in time is independent of its age. This is essentially the RR strategy for very small α (see also discussion of [73] in Section 4.4.2).

4.7 Conclusion

This chapter provided a guide to performance of a range of node selection strategies in real-world traces. We highlighted and explained analytically the good performance of Random Replacement relative to smart predictive strategies, and relative to Preference List strategies. Through the difference in churn between RR and PL strategies, we have explained the performance implications of a variety of existing distributed systems designs. These results also show that some dynamic randomization is an easy way to reduce churn in many protocols.

Chapter 5

Stabilizing Internet Routing

Route instability is an important contributor to data plane unreliability on the Internet, and also incurs load on the control plane of routers. In this chapter, we study how route selection schemes can avoid these changes in routes. Similar to Chapter 4, our techniques function by taking advantage of heterogeneity in the pattern of events. However, rather than considering primarily stability, we now focus on the fundamental tradeoffs that arise from balancing stability with other objectives.

Specifically, we characterize the tradeoffs between *interruption rate*, our measure of stability; *availability* of routes; and *deviation* from the network operator's preferred routes. We develop algorithms to lower bound the feasible points in the tradeoff spaces between these three cost metrics. We also propose a new approach, *Stable Route Selection* (SRS), which uses flexibility in route selection to improve stability without sacrificing availability, and with a controlled amount of deviation.

Our large-scale simulation results show that SRS can significantly improve stability while deviating only a small amount from preferred routes. We implement our protocol in a software router, Quagga, and confirm in cluster deployment that SRS's gains in route stability translate to improved reliability in the data plane. Finally, we evaluate SRS under direct feeds of route update traffic from Internet routers. In this case, we observe much less improvement, but SRS can still improve stability when multiple disjoint paths are available.

5.1 Introduction

A number of studies [36,70,124] point to stability as a key problem for the Border Gateway Protocol (BGP), the interdomain routing protocol which knits the fabric of today's Internet. Network failures, policy changes, and the BGP convergence process itself can generate huge numbers of routing updates, causing problems in both the data and control planes.

In the data plane, it is well known that end-to-end path quality is degraded by BGP route updates (see [126] and references therein). According to a recent study, the majority of packet loss bursts are caused by inter-domain route convergence problems such as transient forwarding loops, rather than by congestion [124]. These problems are increasingly important as the Internet is becoming an ubiquitous platform for voice and video applications. A measurement of VoIP calls between clients on PlanetLab showed that almost half of problems in these calls were highly correlated with BGP updates, and BGP events were estimated to cause 90% of VoIP call drops [68]. Internet games such as Counter-Strike have similar demands for interactivity, commonly sending periodic delay-sensitive bursts of packets [37].

In the control plane, a storm of route updates can overload routers. While current average-case updates do not incur significant CPU utilization [3], there is concern that a burst of updates from multiple neighbors [35] or future update loads resulting from rapidly increasing routing table size [58] could lead to processing and convergence delays [35, 76] or further instability [35], or could necessitate more expensive routers [76]. These problems led the Internet Architecture Board Workshop on Routing and Addressing to recently identify update churn as one of the challenges for future scalability of the routing system [84].

The main mechanism for improving stability in BGP is route flap damping [122], which filters routes that have a short-term update rate above some threshold. Unfortunately this seemingly simple strategy is fraught with problems. In 2002, Mao et al. [80] demonstrated that flap damping creates pathological conditions that slow convergence. Flap damping also worsens *availability*—the fraction of time that a router has a route to a destination—by occasionally shutting off *all* available routes. The operator community has become aware of these problems, with the RIPE Route Working Group advising in 2006 that "the application of flap damping in ISP networks is NOT recommended. ... With

current vendor implementations, BGP flap damping is harmful to the reachability of prefixes across the Internet." [108] Other approaches to improving stability require protocol modifications [22, 80] or address narrow cases [4, 41, 51].

Thus, despite the fact that the problem was recognized more than a decade ago, there is still no compelling mechanism for stabilizing BGP routes, and key questions remain unanswered. For example, notwithstanding the fact that it can delay convergence, does flap damping provide an overall improvement in stability or not? Within the framework of the BGP route selection process, how much is it possible to improve stability, and at what cost?

With concerns about the scalability and reliability of the routing system prompting a renewed interest in stability, we believe it is time for a more principled method to stabilizing Internet routing. Our high-level method is as follows:

1. We identify *general approaches* to stabilizing path-vector routing protocols such as BGP, and their *inherent tradeoffs* with other objectives.

2. We characterize how well each approach can perform by sandwiching the set of feasible points in the tradeoff spaces between upper and lower bounds.

This evaluation makes possible a more informed choice of a method to combat instability.

Characterizing the tradeoff spaces

This chapter studies three general approaches to stabilizing routing: (1) reducing route convergence overhead; (2) avoiding churn by preferring stable routes; and (3) avoiding churn by occasionally shutting off all routes between a particular source and destination. The latter two approaches imply tradeoffs with *availability* and *deviation* from preferred routes, respectively.

Our characterization of what can be accomplished with each of these approaches proceeds in two parts. First, we give algorithms which lower-bound the performance of *theoretically optimal strategies*, which allow us to constrain which points in the tradeoff spaces are achievable, for any given network topology and pattern of failures. To obtain numerical results, we apply these algorithms to a measured topology of around 20,000

autonomous systems (ASes) with inferred customer/provider/peer relationships and one year of link failure data from Route Views [102].

Second, we evaluate the performance of *implementable strategies* including flap damping and new *Stable Route Selection* (SRS) strategies that we develop. SRS has the goal of *safely* stabilizing routing. Unlike flap damping, SRS does not reduce availability; unlike BGP's MRAI timer and Path Exploration Damping [59], SRS does not cause inconsistency by delaying updates. Instead, SRS uses flexibility in route selection to prefer more stable paths. Our evaluation of these strategies uses simulations of the BGP protocol in the same environment as our lower bounds, and experiments using a cluster-based deployment of software routers.

Results

Our lower bound techniques provide the following key impossibility results within our simulation environment:

- *Reducing convergence overhead* can only decrease instability by 5-20% in our simulated environment. This conclusion is robust to the presence or absence of flap damping and the degree of heterogeneity of message propagation delay. However, convergence overhead can be higher due to withdrawals by the origin AS and if AS policies do not conform to standard customer-provider-peer relationships.
- By *allowing deviation* from the operator's desired paths but preserving optimal availability, standard BGP's stability cannot be improved by more than $8.1\times$.
- By also *trading off availability*, stability might be improved by an additional $2-3\times$ to a total of $\approx 20\times$ with some downtime, but bigger improvements are not possible without substantial downtime.

In addition to these lower bounds, we evaluate the performance of implementable strategies, with the following main conclusions:

- Flap damping does improve stability, but at the significant cost of about two "nines" of lost availability with Cisco default parameters.
- Our SRS strategies preserve the high availability of BGP without flap damping, while obtaining up to $5\times$ better stability—slightly better than BGP *with* flap damping, and

coming within 1.6× of the theoretical optimum. Alternately, SRS can provide a 2× improvement over standard BGP while deviating from preferred routes for less than 8 minutes per day, on average. Experiments with a software router deployment show that SRS's benefits translate to improved data plane reliability.

We also evaluate the version of SRS which ensures limited deviation under direct feeds of route update traffic from Internet routers recorded by Route Views [102]. In this case, we find a only a 12% reduction in interruption rate compared with BGP. One reason is that this evaluation is for deployment at a single router, rather than at all routers in the network. But we also provide evidence that the smallness of this improvement is due in part to the fact that for most prefixes, all available routes share the same final AS-to-AS link. For prefixes with multiple disjoint paths, we find an a 24% reduction in interruption rate.

The rest of this chapter proceeds as follows. In Section 5.2, we define our model of BGP, metrics, and classification of approaches to stabilizing BGP. Section 5.3 describes our algorithms for obtaining lower bounds in the tradeoff spaces, and Section 5.4 describes the upper bounds: in particular, our proposed SRS strategies. Our simulation and experimental results appear in Sections 5.5 and 5.6. Our results with Route Views feeds appear in Section 5.7. We discuss related work in Section 5.8 and conclude in Section 5.9.

5.2 Preliminaries

This section introduces a simple model of BGP (Sec. 5.2.1) and the main metrics that we study (Sec. 5.2.2). We then set the stage for the rest of the chapter by classifying approaches to stabilizing BGP and their inherent tradeoffs (Sec. 5.2.3).

5.2.1 Model of BGP

In this section we describe a simplified model of BGP which forms the basis of our analytical results.

At a high level, the operation of a BGP router is simple: for each destination (an IP prefix), it learns advertised routes from its neighbors; it selects one neighbor's route to use, according to some route selection policy; and according to some export policy, it may

subsequently advertise this selected route, as well as its own local destinations, to other neighbors.

Our model adopts those general rules: We are given some fixed destination D. A route is specified as a sequence of nodes along a path to D. Each router R at any given time has *selected* either one route to D, or no route; and may be *advertising* this selected route its neighbors.

Additionally, our model includes two important constraints. First, we assume that message propagation and routing decisions take a negligible amount of time. This condition, which we refer to as **batching**, effectively partitions time into epochs during which all routes are fixed, punctuated by instantaneous "batches" of convergence events. Typically, these events are triggered by link state changes in the underlying topology, but they may also occur if routers decide to change their selected paths after some non-negligible period of time, as in flap damping and some versions of our SRS strategy. Batching enables us to obtain bounds on the optimal policies. Although batching does affect the system, we will observe similar performance with and without batching in our simulations of Section 5.5.

The second condition we impose is **path consistency**. We say paths are consistent at a given moment if for each router v_1 that has currently selected some path $v_1, \ldots, v_k$, the following are true: (1) all links (v_i, v_{i+1}) are up, (2) v_2 selected the path $v_2, \ldots, v_k$ and is advertising this path to v_1, and (3) the ultimate node v_k is the destination D. We require that path consistency holds at any time, except during instantaneous convergence batches. Modulo timing differences, any classic path vector routing protocol, BGP included, attempts to satisfy path consistency. However, in Section 5.2.3 we discuss several strategies which result in inconsistency.

Subject to the conditions of path consistency and batching, routers may follow any route selection and export policies. Our numerical results will use a range of route selection strategies and export policies based on inferred real-world business relationships.

5.2.2 Metrics

In this section we define our three main metrics: interruption rate, availability, and deviation.

An **interruption** is the event that the path selected by some AS changes or is

withdrawn entirely (i.e., a transition to the disconnected state). We do not count recovery events as an interruption. We use interruption rate, which measures stability, as a proxy for data plane performance and control plane CPU utilization due to its computational and analytical tractability. Our experimental results in Section 5.6 will correlate interruption rate with packet loss.

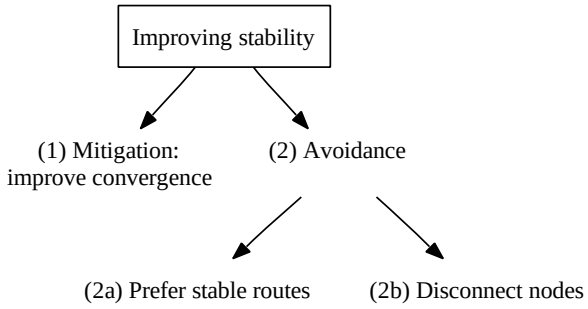
We define **availability** for a particular source-destination pair as the fraction of time that the source has a route to the destination. We will typically study the mean availability over all source-destination pairs.

Finally, **deviation** compares a sequence S of selected paths against a *benchmark sequence* of paths S^*, and is defined as the fraction of time that the route in S "matches" the route in S^*. Deviation is intended to capture how closely a particular route selection strategy (S) follows the network operators' preferred paths (S^*). Since measurements show that ASes' routing preferences are based on next-hops for 98% of IP prefixes [125], we say that two routes from a particular source "match" when their next-hops are equal. Our simulations will take S^* as the paths selected by the standard BGP decision process; thus, our numerical results measure how much various strategies differ from the status quo. As with availability, we study the mean deviation over all source-destination pairs.

5.2.3 Approaches to Stabilizing BGP

Within our model, it is possible to give a complete classification of approaches to reducing interruption rate relative to standard BGP.

At a high level, we have a simple choice: pick the same sequence of paths as standard BGP—except during the instantaneous convergence events—or pick paths which differ. These two cases respectively imply that we must either (1) *mitigate the impact* of topology or policy changes by improving the reconvergence process, or (2) *avoid* reconvergence events altogether. Within the avoidance approach, there are two pure approaches: (2a) select paths that fail less often, and (2b) select *no path*, disconnecting the source from the destination.



These three approaches require qualitatively different sacrifices ranging from free to severe. Approach (1) is the most attractive because it improves stability without compromising other objectives. The two remaining approaches directly imply tradeoffs: (2a) results in nonzero deviation, and (2b) is an extreme approach which sacrifices availability. In the limit, a network where all nodes are disconnected has no interruptions, but also has zero availability!

Note that (2b) is not equivalent to RFD's strategy of shutting off (damping) unstable routes. Damping a route causes BGP to select another route, as long as an alternate un-damped route is available. Thus, RFD mixes approaches (2a) and (2b).

Our characterization of the tradeoff spaces places limits on what can and can't be accomplished with these three approaches. We begin the characterization in Sec. 5.3 with our lower bounds algorithms. Our upper bounds, i.e., implementable strategies, include our new SRS strategies described in Sec. 5.4 in addition to Standard BGP and RFD. The numerical results of these lower and upper bounds appear in Sec. 5.5.

Strategies outside this classification. Note that policies which allow path inconsistency (defined in Sec. 5.2.1) are implementable in today's BGP but fall outside our model. Such strategies are beyond the scope of this work, and may be useful; however, we note that as a result of their path inconsistency, they must be handled with care, since there is the possibility that routing loops or disconnectivity can persist for non-negligible periods of time.

To the extent that a 30-second time period is considered non-negligible, an example of a path-inconsistent strategy is the commonly-used Minimum Route Advertisement Interval (MRAI) timer. The MRAI rate-limits update messages to each neighbor of a router to one per 30 sec. Recently, Huston [59] proposed delaying updates for just longer than an MRAI interval, 35 sec, when they match a pattern likely to indicate BGP path exploration.

Consecutive matches could delay updates for minutes or more.

5.3 Lower Bounds

In this section we describe how we numerically compute bounds on the maximum possible improvement that can be obtained from the above approaches to improving stability in BGP. In Section 5.5, we will apply these lower-bounds procedures to the measured topologies and traces used in our simulator, allowing us to compare how close our proposed strategies are to the best possible policies.

The procedures take as input an AS-level topology annotated with customer-provider-peer business relationships between ISPs, which they honor; a trace of AS adjacency ("link") failures; and a sequence of preferred route selections over time for each source/destination pair.

Given this input, we will find the following:

- **Approach (1): Convergence.** We find the minimum number of interruptions required to adhere to the given preferred routes *almost always*, i.e., at all times except for negligible periods of time during convergence events (formally, a set of times of zero measure).

- **Approaches (1)+(2a): Stability-Deviation tradeoff.** We compute a set of *undominated points* (x_i, y_i) such that for each i, in the given topology and traces, it is impossible to select routes that achieve both a mean interruption rate of $< x_i$ and a mean deviation of $< y_i$, while preserving the highest possible availability. Means are over all sources for a given set of destinations (which will be a random set when we generate results in Section 5.5).

- **Approaches (1)+(2a)+(2b): Stability-Availability tradeoff.** Similarly, we compute a set of undominated points (x_i, y_i) such that for each i, it is impossible to select routes that achieve *both* a mean interruption rate of $< x_i$ *and* a mean unavailability of $< y_i$, with no constraint on deviation.

The first item, bounding convergence overhead, appears in Section 5.3.1 and is straightforward. In contrast, it is nontrivial to obtain good lower bounds in the trade-

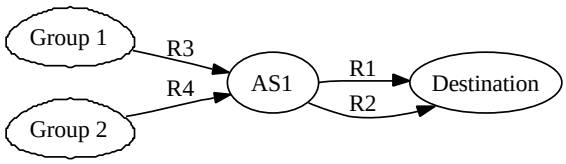
off spaces. We explain why (Section 5.3.2) before describing the procedure (Section 5.3.3) which is similar for the two tradeoff spaces.

5.3.1 Convergence

Batching, described in Section 5.2.1, allows us to easily lower-bound the minimum number of interruptions needed to match the preferred routes almost always, i.e., with no convergence overhead. Given a fixed setting of each router's converged state before and after each batch, there must be at least 1 interruption for each batch in which the route changed or was withdrawn, and at least 0 if the path stayed the same. We simply sum these over all batches to produce the desired lower bound for each source-destination pair.

5.3.2 Hardness of Minimizing Interruptions

Provably providing a good bound in the tradeoff spaces is nontrivial in large part because of *dependencies between nodes* when routing to some particular destination. We illustrate this with an example for one particular point in the tradeoff spaces: that of minimizing interruption rate under the constraint of maximum availability. Consider the following topology:



Groups 1 and 2 consist of n and m ASes, respectively, which depend on routing through $AS1$ to the destination. Suppose the routes $R1$ and $R3$ are available during the time interval $[0, 10]$, while $R2$ and $R4$ are available during $[5, 15]$. Sometime during $[5, 10]$, $AS1$ must switch from $R1$ to $R2$. But this affects Groups 1 and 2: if $AS1$ switches at time 5, it triggers an interruption at each of the n nodes in Group 1; if it switches at time 10, it interrupts the m nodes in Group 2. The optimal routes for $AS1$, in terms of the cost to the network as a whole, therefore hinge upon whether $n < m$. We thus have nonlocal dependencies between nodes' route selections.

In fact, it is possible to show that the problem of minimizing the number of interruptions is NP-complete, using a gadget similar to the above as one step of a reduction

from SAT. The proof appears in Appendix D.

5.3.3 Tradeoff Procedure

To avoid the dependency problem we allow each node to independently select its own route, thus possibly picking a route which was not chosen by the downstream ASes along the path. This relaxation of path consistency will allow us to compute optimal tradeoff values for each (source, destination) pair separately; we then assemble these pieces together to produce an undominated point in the tradeoff spaces.

We first describe two important subroutines which we then use in our final procedure.

Optimal Path Sequence (OPS) subroutine

This core "inner loop" used by our tradeoff calculations computes the optimal sequence of path selections for a single node over time. Specifically, OPS is given a set of route options. Each option is of the form (r, t_1, t_2, c), meaning route r is available during $[t_1, t_2]$ and incurs cost c per unit time that it is selected. The cost of causing an interruption (see Sec. 5.2.2) is fixed at 1. OPS computes the minimum cost sequence of route selections over time for the given options.

This computation is equivalent to a shortest paths problem on an appropriately constructed abstract graph whose nodes and edges represent route choices and legal transitions between them, respectively. Our implementation uses a somewhat more efficient dynamic programming algorithm which iterates through time, storing the least-cost way to reach each currently available path choice; we omit the details.

Most Stable Paths subroutine

This subroutine computes a set of potential routes that will later be fed into the OPS subroutine. Specifically, it calculates, for each source s, destination d, and time t, *the available $s \rightarrow d$ path which will be available starting at t and continuing farthest into the future.* Given knowledge of the future availability of any given route, this can be computed *en masse* for all sources and a particular destination using a BGP-like algorithm whose path selection prefers routes that will be available longer.

Computing the future availability of any route involves examining future link failure times, which are easily obtained from the trace provided to the lower bounds procedure. However, it also involves a complication: the future failure time of a route is dependent on future changes in the business class of the route selected by each AS on the path. For example, if a downstream AS switches from a customer route to a peer route, it will no longer export the route to other peers or providers. We calculate business class switch times by running the route selection simulation twice, recording the business class switch times on the first trial, and using them to compute paths' future failure times in the second. It can be shown, along the lines of the proof of convergence in [41], that the business class switch times will be identical in both trials.

Putting it all together

We now describe how we compute a set of undominated points (i.e, a Pareto set) in the tradeoff spaces, utilizing the above subroutines. We begin with the stability-availability tradeoff.

Let $R = \{r_{sd}\}$ be a sequence of route selections for each source-destination pair (s, d), and let $intr(\cdot)$ and $down(\cdot)$ represent the number of interruptions and the amount of downtime, respectively, in R. Our goal is to produce R's for which the point

$$(intr(R),\ \ down(R))$$

is undominated. To do this, we we begin with the well-known weighted sum method of multiobjective optimization, as follows. We introduce a parameter λ which intuitively sets the cost of being disconnected per unit time. We next describe how to produce a single undominated point given λ; we vary λ to produce multiple points.

A straightforward application of the weighted sum method would then find the optimal feasible value R^* of R which minimized $intr(R) + \lambda \cdot down(R)$. However, as we showed in Sec. 5.3.2, computing R^* is hard. We instead optimize each source-destination pair separately. Specifically, for each source s and destination d we find (using a procedure to be described below) the valid route assignment ℓ^*_{sd} which minimizes

$$intr(\ell^*_{sd}) + \lambda \cdot down(\ell^*_{sd}). \tag{5.1}$$

We then construct the network-wide route assignment $L^* = \{\ell^*_{sd}\}$ and finally return the undominated (though potentially infeasible) point $p = (intr(L^*), down(L^*))$. We must

show that p is in fact not dominated by any feasible point. To see why this is true, assume for the sake of contradiction that there existed some feasible route assignment $X = \{x_{sd}\}$ for which both $intr(X) < intr(L^*)$ and $down(X) < down(L^*)$. This implies

$$intr(X) + \lambda \cdot down(X) < intr(L^*) + \lambda \cdot down(L^*)$$

and hence that for some source-destination pair (s, d),

$$intr(x_{sd}) + \lambda \cdot down(x_{sd}) < intr(\ell^*_{sd}) + \lambda \cdot down(\ell^*_{sd}).$$

But then ℓ^*_{sd} must not have minimized (5.1)—a contradiction. Hence, the point p returned by the procedure is undominated.

We have thus reduced the problem to that of minimizing (5.1) for an individual source-destination pair (s, d). We note that whenever a path is selected at time t, an optimal choice is the path which will be available for longest time into the future beginning at time t (see Fact 5 in Appendix C.3). Thus, the Most Stable Paths subroutine provides (a superset of) the set of routes which might be involved in the optimal sequence, ℓ^*_{sd}. We add to this set a persistently available "null route" with cost λ per unit time, and feed all these choices to the OPS subroutine, whose output must be an optimal set of route selections, ℓ^*_{sd}.

That concludes our procedure for lower-bounding the stability-availability tradeoff. Our bound in the stability-deviation space is quite similar. The main difference is that since route preferences are based on next-hops (see Sec. 5.2.2), the set of routes in ℓ^*_{sd} might include the route *through any neighbor* which will be available longest into the future. We modified the Most Stable Paths subroutine to obtain these routes. We label them with costs per unit time—zero while they are preferred, or λ otherwise—and then feed them into OPS to produce ℓ^*_{sd}.

5.4 Stable Route Selection

We next describe our proposed class of Stable Route Selection strategies. SRS avoids instability by using flexibility in route selection to select more stable paths (approach (2a) in the classification of Section 5.2.3). SRS offers a tunable tradeoff between stability and amount of deviation from preferred paths. In this section, we describe where SRS fits in the context of the BGP decision process (Section 5.4.1), and then how our SRS strategy selects paths (Section 5.4.2).

5.4.1 Fitting SRS into BGP

BGP affords a high degree of flexibility through the use of a *decision process* [111], which allows operators to customize route selection to conform to goals such as traffic engineering or economic relationships. The BGP decision process consists of the sequence of steps shown in Table 5.1, which select a route based on *attributes* contained in the BGP route announcements. The output of each step is a *set* of routes that are *equally good* according to that and every previous step. By adding, modifying, or filtering attributes in update messages, operators can control the specific route selected to reach a particular destination.

Step	Action
1.	Highest local preference
2.	Lowest AS path length
3.	Lowest origin type
4.	Lowest MED
5.	eBGP over iBGP-learned
6.	Lowest IGP cost
7.	Lowest router ID

Table 5.1: BGP Decision Process

We insert the SRS heuristic as an additional step between Steps 1 and 2 of the BGP decision process. (Alternately, SRS could be implemented by appropriately modifying route attributes using an import filter before routes reach the decision process.) SRS selects the best route based on a combination of Steps 2-7 and a heuristic for predicting route stability. We present the details of SRS in the next section.

An alternate design would have placed SRS before the first step, like flap damping. We chose to place SRS after the Local Preference step to ensure that the highest-level routing preferences, such as preferring customer routes over provider routes, are always maintained, even during SRS's delay period (see below). This has at least two benefits. First, it provides a useful guarantee to operators. Second, we note that it is possible for a violation of the Local Preference step to reduce availability for other ASes. In particular, ASes typically have neighbors who are *providers*, *customers*, or *peers*; a route whose next-hop is a provider or peer is exported only to customers [41]. If an AS were to select a provider route over a customer route, it would block the export of the route to other providers and peers, potentially disconnecting those neighbors from the destination. Al-

though this case may be uncommon, it is our primary concern to ensure high availability.

Despite the restrictions imposed on SRS by being subordinate to Local Preferences, in our simulations, sufficient flexibility remains to significantly improve stability.

5.4.2 The SRS Heuristic

The SRS heuristic is inserted after Step 2 of the BGP decision process (Table 5.1). The heuristic has one main parameter, a *delay* δ. We will write $SRS(\delta)$ to indicate the value; SRS with the parameter omitted refers to $SRS(\infty)$. SRS uses a procedure, $pref(r_1, r_2)$, that implements Steps 2-7 in Table 5.1. $pref(r_1, r_2)$ returns "first" if r_1 is more preferred according to Steps 2-7, "second" if r_2 is more preferred, or "equal" if they are equally preferred. SRS then decides which of two routes r_1, r_2 should be selected as follows:

1. If r_1 has been up for time $\geq \delta$ and $pref(r_1, r_2) =$ "first", then select r_1.
2. Otherwise, if r_2 has been up for time $\geq \delta$ and $pref(r_1, r_2) =$ "second", then select r_2.
3. Otherwise, if one of r_1 and r_2 is currently selected, keep that route.
4. Otherwise, if one of r_1 and r_2 has lower AS path length, select that route.
5. Otherwise, select the route that has been up for the longest time.

The single winning route is selected by iterating this pairwise comparison over all available routes.

The intuition behind this choice of steps is as follows. Steps 1 and 2 select preferred routes, as long as they are not recent advertisements. This step assumes that recently advertised routes are more likely to be withdrawn soon, and provides a tradeoff in the parameter δ. $SRS(0)$ is equivalent to the decision procedure $pref(\cdot, \cdot)$, while $SRS(\infty)$ gives no consideration to preferred routes, reserving maximum flexibility for stability.

The strategy of sticking with the current choice (Step 3) and then using a "longest uptime" strategy if that choice fails (Step 5) has been used in many contexts from page replacement to peer-to-peer systems, and is a good heuristic for stability since past behavior is frequently correlated with future behavior (see Chapter 4). In simulations, we found that inserting the shortest-path step (Step 4) often somewhat improved stability while simultaneously providing a significant reduction in mean path length.

5.5 Analytical and Simulation Evaluation

In this section we describe results from a simulation-based evaluation, as well as the results of our lower bounds algorithms applied to the same environment as the simulator. We first describe the structure and setup of our simulator in Section 5.5.1 and present results in Section 5.5.2.

5.5.1 Methodology

Data sets. We infer the inter-domain AS-level topology by culling AS adjacencies from Route Views [102] feeds. Since policies on the Internet are not widely disclosed, we leverage [116] to infer and assign local preferences associated with business relationships as done in [70, 80, 117], characterizing links as either provider-customer or peer-peer. We assume ASes distribute routes according to the common-case import and export policies as discussed in [116]: ASes prefer customer over peer and peer over provider routes, and don't advertise routes from peers/providers to other peers/providers.

To infer the pattern of failures, we record the appearances and departures of links from the Route Views feeds. Specifically, we consider a link to be available at time t if some route which uses the link is currently advertised to a Route Views peer at time t. In this manner, we infer a trace of link state changes from Route Views from January 1, 2006, to December 30, 2006, which we replay against our simulator.

Simulator. To evaluate the performance of various route selection strategies, we use an event-driven BGP simulator extended from the simulator used in [33]. The simulator's events are at the level of link state changes and BGP update messages. Each AS is represented by a single node running a BGP instance, as in some past studies [21, 117]. Inter-AS packet propagation delay is selected randomly for each packet, uniformly distributed between 5 and 15 ms.

The simulator runs a simplified version of the BGP protocol described in RFC 1771 [97]. Since our simulator models each AS as a single router, Steps 3-6 of the decision process (see Table 5.1) are not executed. For Step 7, we assign each AS a uniform random router ID.

We implement batching (see Sec. 5.2.1) in the simulator to compare with our lower bounds. We do this in such a way that the BGP update messages are processed *in the same*

order that they would have been with link delays and MRAI timers turned on and with subsequent topology changes delayed until after the convergence process completed.

Each plot incorporates measurements of at least 100 trials. In each trial we select a single random destination to which all nodes route over a random month of our year-long data. We gather measurements only after the first 24 hours of simulated time, to eliminate initial convergence effects. Since some data is missing from our topology causing a minority of nodes to be always disconnected, when collecting measurements we ignore source-destination pairs whose availability in the Standard BGP strategy (without flap damping) is < 0.99. Other than excluding ASes which were usually disconnected, this did not substantially affect our results.

Route selection strategies. Our simulations will compare the basic BGP decision process, which we call *Standard BGP*, with SRS and with Route Flap Damping (RFD) as in RFC 2439 [122].

RFD maintains a numeric penalty value $p_{P,N}$ associated with every (prefix P, neighbor N) pair. Upon receipt of an advertisement or withdrawal, the router increases $p_{P,N}$. When $p_{P,N}$ increases beyond a *cut-off threshold*, the route is excluded from consideration when selecting routes. The penalty decays exponentially, and the route is reconsidered for use when its value falls below a *reuse* threshold. In our tests, unless otherwise stated, the strategy "RFD" refers to flap damping with Cisco's default parameters, which increase $p_{P,N}$ by 500 after attribute changes and by 1000 for withdrawals, and specify a reuse threshold of 750, a cut-off threshold of 2000, and a decay half-life of 15 minutes [80]. We also test with flap damping parameters used by Juniper [80], SprintLink [109], and three sets of parameters recommended by RIPE [90].

5.5.2 Results

This section presents the results of our simulation and our lower-bounds analysis applied to the environment described above. Our main conclusions are as follows:

- Batching, which allows us to obtain bounds on the optimal policies, preserves the qualitative performance of various strategies (Section 5.5.2).
- Improvements to convergence cannot obtain a large improvement in our environment. This conclusion is surprisingly robust under various message propagation

delay distributions, but convergence overhead can be larger due to policy misconfiguration and withdrawals by origin ASes (Section 5.5.2).

- $SRS(\infty)$ can obtain a dramatic improvement in stability compared with Standard BGP and greater than that of RFD, without sacrificing availability and closely approaching our lower bound (Section 5.5.2).
- By adjusting the delay parameter appropriately, SRS can reduce interruption rate by 68% while deviating from preferred paths less than 0.6% of the time (Section 5.5.2).
- SRS only slightly increases mean path length (Section 5.5.2), and stability-aware routing can obtain significant improvements in stability even under limited deployment scenarios (Section 5.5.2).

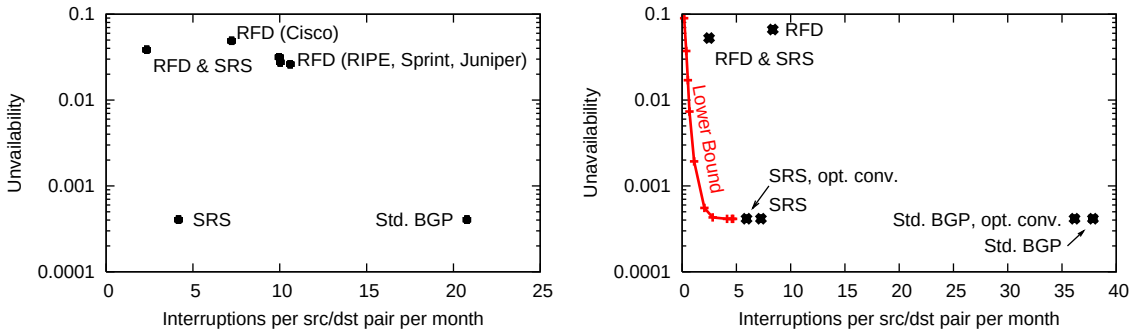
Effect of batching

Figure 5.1 shows performance of various strategies both without and with batching. Recall from Section 5.2.1 that batching makes link delays and MRAI timers negligible, allowing us to lower-bound the interruption rate of optimal policies. We see a substantially similar relationship between the strategies with batching on and off, which suggests that batching is a reasonable approximation under which to compare strategies. The main difference is inflated interruption rates, which can be explained by the fact that in Fig. 5.1(a) some link state changes may be effectively skipped as a result of link delays and MRAI timers, while in the batched environment the system finishes reconverging after every topological change.

Convergence overhead

Figure 5.1(b) shows the interruption rate of Standard BGP and SRS along with their hypothetical counterparts with optimal convergence—which transition from the initial to the final path in each batch without any path exploration process. This shows that in our environment, convergence has only a minor contribution to interruption rate. But on what aspects of the environment does this conclusion depend? We investigated how origin events, policy misbehavior, and heterogeneity of link delays affect convergence overhead.

Origin events. ISPs may withdraw and announce prefixes either intentionally or due to configuration error, triggering long sequences of path hunting which are triggered rela-



(a) Batching off

(b) Batching on

Figure 5.1: Performance of various strategies in the stability-availability space, with batching off and on. SRS's delay parameter is fixed at ∞. Unless otherwise specified RFD refers to flap damping with Cisco standard parameters.

tively rarely by our Route Views traces of link state changes. Here, we consider the effect of an origin AS announcing or withdrawing a single prefix. To do this, we constructed a trace consisting of announcements/withdrawals separated by long periods of time. This allowed us to measure the convergence overhead for each change in isolation, and allowed us to compare against the BGP Beacons measurement study by Mao et al. [81]. Like [81], we found that prefix advertisements generate roughly 3-5$\times$ fewer updates than withdrawals. For example, with our default configuration, we found routing advertisements incurred an overhead of 1.14 while withdrawals incurred an overhead of 6.00 on average. In addition, we found that failing a single link adjacent to the destination incurred an average overhead of 3.90, while repairing the single link incurred an average overhead of 1.03.

Policy misconfiguration. Misconfigured ISPs may introduce oscillations and instability which can lengthen convergence periods. We randomly selected a small fraction of ASes to "misbehave" by selecting routes based on a uniform-random preference ordering among next-hops, rather than our default configuration based on the Gao-Rexford policies [41]. We found that having a fraction of ASes misbehave in this manner can increase convergence overhead; for example, convergence overhead is 2.68 with 2% of ASes misbehaving, or 2.70 with 5% misbehaving.

Link delay. Increasing the delay of links, or the heterogeneity of delays across links, has

been associated with worsening of routing convergence times [69]. To measure this, we varied the distribution of delays of inter-AS links in our simulator. We assigned each link a delay sampled from a Pareto distribution with shape parameter α, which controls the variance of the distribution, and varying mean. Results are shown in Table 5.2. Like previous work, we found that increasing link delays increases convergence *time*. However, varying the variance of links, and varying the mean delay of links, only changed convergence *overhead* slightly. Like previous work [50], we also found that enabling MRAI increases convergence time, but reduces control overhead.

α	**Convergence overhead**	
	mean = 4 ms	mean = 100 ms
2.000001	1.154	1.159
2.0001	1.166	1.151
2.01	1.168	1.167
2.1	1.167	1.165
2.4	1.163	1.165
4	1.165	1.166
15	1.169	1.170
2000	1.172	1.172

Table 5.2: Effect of varying link delay.

Stability-availability tradeoff

Figure 5.1 shows performance of various strategies in the stability-availability space. Comparing the points in Figure 5.1(a), where batching is disabled, Standard BGP maintains a high availability of 99.98%, but suffers from a high rate of 20.8 interruptions per month. Route Flap Damping (RFD) with Cisco's default parameters reduces the mean rate of interruptions by a factor of 2.9, but sacrifices two "nines" of availability, and the other parameter values used by Sprint and Juniper and recommended by RIPE have substantially similar performance. One might expect the tradeoff between availability and stability to be fundamental. However, by preferring more stable paths, SRS is able to achieve the high availability of standard BGP with even fewer interruptions than RFD. Although we do not advocate the use of RFD, we note that using RFD and SRS in conjunction results in an additional 3.1-fold decrease in interruption rate over RFD and slightly

improves availability over RFD alone. This is to be expected, since by picking more stable paths, RFD is triggered less often.

Figure 5.2 explores the pattern of interruptions in more detail with a complementary CDF over all measured end-to-end paths. That is, the y axis shows the fraction of (source, destination) pairs that have *at least* the interruption rate on the x axis. Standard BGP's long tail shows what other studies [36] have observed: a small number of Internet routes suffer from high instability. Both SRS and RFD drastically reduce the size of this tail. SRS is able to achieve roughly the same benefit in the tail as RFD without incurring RFD's reduction in availability. Interestingly, and unlike RFD, SRS improves the stability for the upper part of the curve, i.e., for routes that have only moderate instability. Finally, when we combine SRS with RFD, we note that the instability of the most unstable routes is reduced by about an order of magnitude compared with RFD in isolation.

Figure 5.1(b) also compares performance with lower bounds on the optimal policies. SRS performs surprisingly close to optimal among strategies which achieve the highest availability, with an interruption rate just 55% higher than our lower bound, which uses knowledge of the future. Factoring out SRS's convergence overhead, it would be just 27% worse than optimal. These results indicate that the SRS heuristic does a very effective job of predicting the relative stability of paths in this environment. It also shows that BGP's stability cannot be improved by more than about 8.1$\times$ without sacrificing availability, given our assumptions such as the preservation of common business relationship-based routing policies and path consistency (see Sec. 5.2.1).

Finally, the "Lower Bound" curve in Figure 5.1(b) demonstrates limits on how much improvement can be gained by occasionally disconnecting nodes. This lower bound admits the possibility that stability can be improved to about 2 interruptions per month with small availability loss, but any further improvements come at the cost of significantly more downtime. For example, reaching 1 interruption per month requires reducing availability from 99.96% to below 99.8%, *i.e.*, over 4$\times$ as much downtime.

Stability-deviation tradeoff

The above results, which set SRS's delay parameter to ∞, assume SRS is permitted to use a large amount of flexibility in the choice of paths. In this section, we explore the tradeoff between stability and the fraction of time that routes deviate from the next-hops

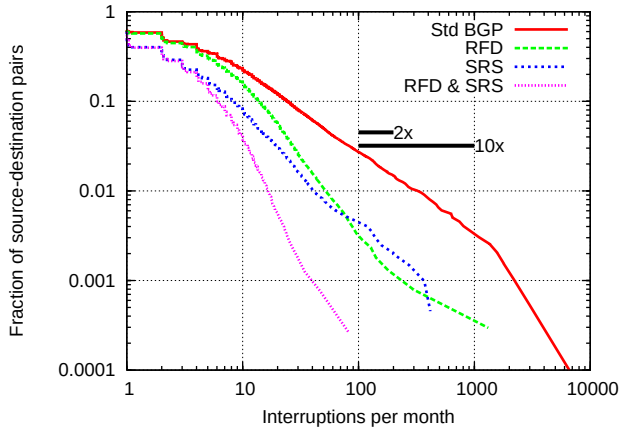


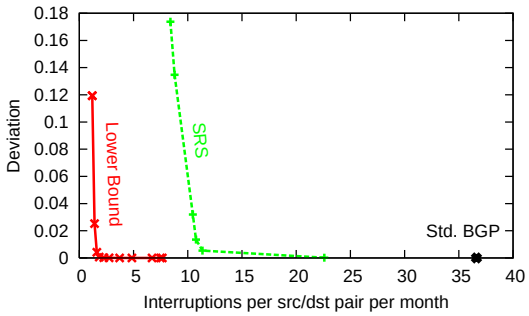
Figure 5.2: Complementary CDF of interruptions per month over all measured source-destination pairs. SRS's delay parameter is fixed at ∞.

chosen by Standard BGP.

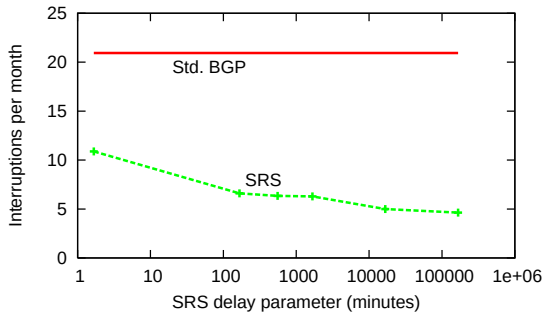
Figure 5.3(a) shows the stability-deviation tradeoff that results from varying SRS's delay parameter, in the batched environment. With a delay of $\delta = 100$ sec, SRS cuts interruption rate by about 38% compared with Standard BGP and has a mean deviation of 0.021%, with 99.5% of ASes having deviation $< 1\%$. At the knee of the tradeoff curve, with $\delta = 167$ minutes, interruption rate is 69% lower than Standard BGP; mean deviation is 0.54% and 86% of ASes have deviation $< 1\%$. Figure 5.3(b) shows that even with batching off (MRAI and link delays on), SRS with $\delta = 167$ minutes still reduces interruption rate by 68%. These results are promising: many ISPs base bandwidth utilization payments on the 95th percentile of the traffic load for each month [89], so a deviation of less than one percent may be acceptable.

We also note there may be cases where it is useful to move beyond the knee to obtain greater stability. For example, if an ISP has the ability to route multiple classes of traffic along different routes, it would be possible to send the most stability-sensitive flows (*e.g.*, real-time voice traffic) along SRS paths, and other flows along whichever paths minimize maximum link utilization. This would allow the ISP to achieve critical traffic engineering objectives while still providing greater stability where it matters.

Figure 5.3(a) also plots our lower bound to the optimal achievable points in the stability-deviation space (without sacrificing availability). In contrast with the stability-



(a) The stability-deviation tradeoff space with *batching on*.



(b) SRS's interruption rate as a function of its delay parameter, with *batching off*.

Figure 5.3: Deviation and interruption rate of SRS(δ) for various δ. SRS's delay parameter δ varies from 100 sec to effectively ∞ (a value longer than the one-month trace).

availability lower bound, this is quite far from the upper bound, SRS. We believe the true optimal is actually much closer to the upper bound. Our basic lower-bounds technique in both tradeoff spaces relies on computing the optimal sequence of paths for each node independently, allowing nodes to take inconsistent paths. In particular, a node can adhere to its preferred next hop, while choosing the remainder of the path to be maximally stable. But this has implications on the amount of deviation of the nodes along the path, which our algorithm does not take into account. Providing a substantially better lower bound would likely be possible by using an alternate definition of deviation which requires that the entire path matches the preferred path, rather than just the next hop.

Path length

The previous section dealt with adherence to general routing objectives, in the form of next-hop preferences. In this section, we study one objective which is not reducible to next-hops: path length.

Figure 5.4 shows the CDF of mean path length over source-destination pairs. SRS has a mean path length of 4.14 AS-level hops, or just 4.2% greater than Standard BGP's 3.97 hops. We note that BGP itself empirically incurs an AS-level path inflation of 49.8% due to policies [19]. Hence an additional inflation of 5% (resulting in a combined inflation of roughly 56.4%) may be tolerable to some ISPs, while others may tune SRS to reduce this

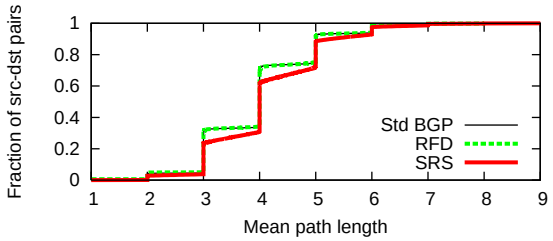


Figure 5.4: CDF of mean path length over all measured source-destination pairs.

inflation at the expense of a somewhat lower stability.

This low inflation may be expected since part of the SRS heuristic prefers shorter paths (Sec. 5.4). In addition, path lengths in this environment are constrained by the hierarchical nature of the AS graph when business relationships are satisfied: we found that a hypothetical strategy which always preferred *longest* paths would have mean path length just 32% longer than Standard BGP.

Partial Deployment

Since Internet path selection is a collaborative process, SRS's benefit to one autonomous system would depend on whether other ASes have also deployed it. But for reasons of deployment incentive, it would be helpful if a single AS can unilaterally deploy SRS and achieve at least some improvement.

The following evaluation of partial deployment scenarios uses a Random Replacement strategy (following Chapter 4), rather than SRS. Specifically, instead of Steps 4 and 5 of the SRS heuristic described in Section 5.4.2, we select a random available route from the set of routes most preferred by the previous steps. These results use RR only for historical reasons, and the performance is very similar to SRS with delay $\delta = \infty$.

Figure 5.5 shows the performance of this RR strategy with the extent of deployment ranging from 1% to 100% of ASes. The set of ASes running RR is selected randomly in each trial, and the others use standard BGP. We measure the interruption rate on those nodes running RR separately from those not running RR. The leftmost points in the plot show that the first ASes deploying RR would see a significant drop in their own interruption rate of roughly 1.8×. As more ASes deploy stability-optimized route selection, the interruption rates decrease further, so that when all ASes implement RR, we achieve a 5×

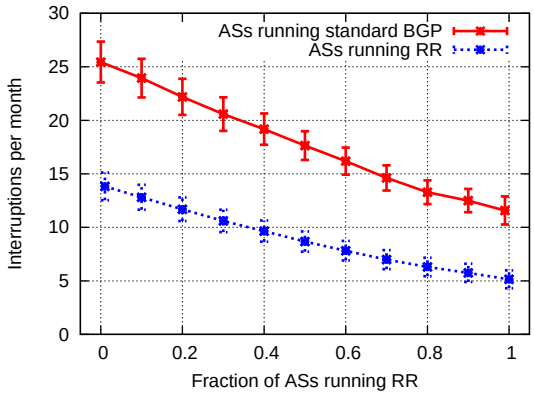


Figure 5.5: Interruption rate under partial deployment of a stability-aware routing strategy, with delay parameter $\delta = \infty$ and batching off.

reduction in the interruption rates (similar to the improvement with a full deployment of SRS(∞) as shown in previous plots). This is due to the distributed nature of BGP path selection: each AS that prefers more stable routes effectively stabilizes routes on behalf of other ASes that route through it (including those running standard BGP!).

The factor of 1.8$\times$ improvement, however, is for the average AS; certain ASes obtain a somewhat bigger improvement. In particular, we found that individual Tier-1 ASes (as classified by [116]), which have more flexibility in path choice, had a median reduction in interruption rate of 2.7$\times$, with 4 of the 28 Tier-1s seeing more than a 4$\times$ improvement.

5.6 Experimental Results

Our experimental results use a network of software routers. Although we have scaled it only to tens of nodes, this deployment allows us to (1) measure performance in a realistic implementation, and (2) validate our simulations by correlating them with observed data plane performance. (We note that another major deficiency of our simulator is that the patterns of failure may not match real-world route update patterns; that deficiency is addressed in Section 5.7.)

5.6.1 Methodology

Our experimental evaluation is based on the BGP implementation in the Quagga software router [94]. We modified Quagga in two main ways: we implemented new route selection policies in the decision process, and we built a custom forwarding plane to enable more flexible experimentation. We then evaluated the resulting software router by deploying it on a cluster and emulating failures. These steps are described in more detail below, and the overall arrangement of components is depicted in Figure 5.6.

Route selection policies. We altered Quagga's decision process to (optionally) run SRS with delay $\delta = \infty$ or $\delta = 10$ min. We also tested several strategies already implemented in Quagga: what we have referred to as Standard BGP; Quagga's default strategy, which employs a longest-uptime step directly before the final router-ID step of the decision process; and Route Flap Damping, again with Cisco-default parameters.

Forwarding plane. We built a custom data plane to enable more complete instrumentation and to conveniently run on a shared cluster. In effect, this data plane allows us to use Quagga as a router for an overlay network. As depicted in Fig. 5.6, we instrumented Quagga so that it sends route updates, in the form of (destination, next hop) pairs, to our forwarding plane, rather than to the kernel. The forwarding plane generates, receives, and forwards UDP packets to remote forwarding plane processes, as directed by Quagga's route updates. Data packet generation occurs every 5 seconds, with 10% random jitter, for every (source, destination) pair. That is, the time between packets for a (source, destination) pair is uniform random in $[4.5, 5.5]$.

Emulating failures. We emulate link failures by leveraging Linux's iptables facility, with which we can block or unblock UDP and TCP traffic between pairs of software routers at appointed times. This allows us to emulate a trace of link failures and recoveries over our chosen network topology.

We configured our software routers using two small-scale network topologies. First, we employ IS-IS link-state updates and topology traces from the Abilene backbone network [1]. Although our primary target is interdomain rather intradomain routing, this gives us a realistic environment of scale appropriate for our testbed. The topology contains 11 nodes and 14 edges. We use a portion of this trace from 10 August 2004 to 13 Oct 2004. We "compressed" this trace by reducing to 2 minutes every interval of greater than

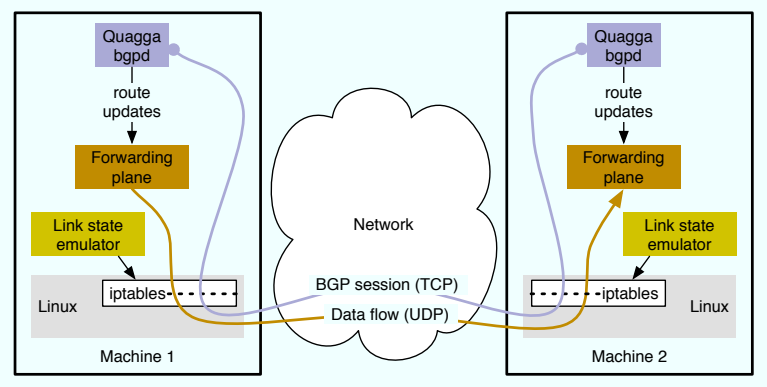


Figure 5.6: A diagram of our software router, showing two nodes with a flow of data from Machine 1 to Machine 2.

2 minutes in which no events occurred. This reduced the length of the one-month trace to 7.3 hours. The compression step preserves the ordering of events, while allowing us to run tests in a shorter time period and to stress-test the route selection policies in a more challenging environment. This does change the pattern of failures and can have an effect on the performance of the strategies we test. However, the main goal of this section is not to test a realistic pattern of failures, but rather to observe the effects of moving from a simulation environment with only a control plane, to an implementation with a data plane.

The second environment we use is a synthetic Erdös-Rényi $G(n, m)$ random graph, i.e., n nodes with m edges connected uniformly at random. We used $n = 25$ and $m = 50$, so that the average node degree is 4. We generated a bimodal pattern of failures among the m edges: a random set of 40 are stable, never failing; the other 10 are unstable, with a heavy-tailed uptime distribution of mean 2 minutes, and constant 1-minute downtimes. The trace lasts 2 hours.

We exclude results near the beginning and end of the trace to avoid measuring startup and shutdown effects. We show results from single trials, but we have found repeated trials produce very similar results.

5.6.2 Results

In this section we show that SRS can substantially improve data plane performance over other strategies; interestingly, a delay of $\delta = 10$ minutes performs better than

$\delta = \infty$, potentially due to faster convergence as a result of slightly shorter paths. We then correlate our simulator's interruption rate metric with packet loss in the software router.

Software router performance

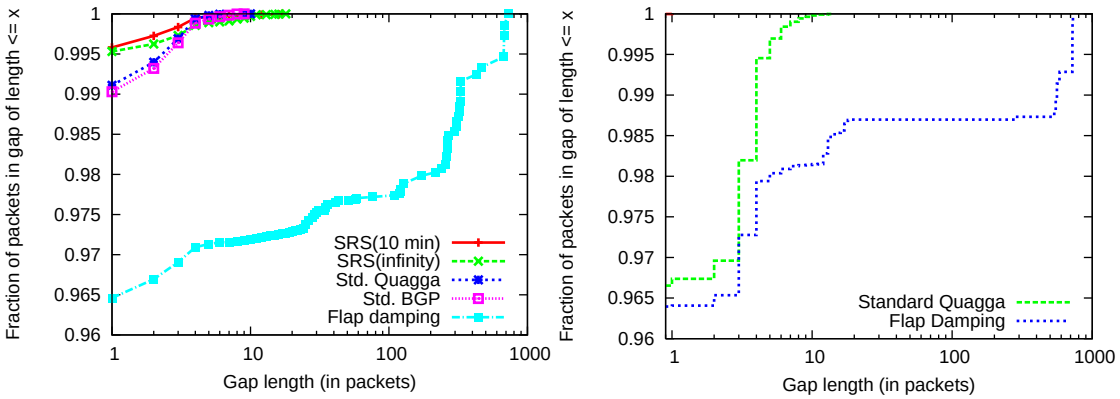
Table 5.3 classifies packets by their fate. The columns respectively indicate packets that were received, dropped because there was no route to the destination, sent along a virtual link which was down (i.e., dropped by iptables), dropped after exceeding the maximum hopcount, or dropped for unknown reasons (presumably, dropped by the underlying physical network).

Figure 5.7 depicts data-plane performance in the form of the distribution of "gap lengths". We say that a generated packet lies in a gap of length 0 if it was received by the destination, and it lies in a gap of length k if it is one of a run of k dropped packets. Gap lengths are significant since brief outages can often be masked by retransmission. Note that a gap length of k corresponds to an outage of $\approx 5k$ seconds.

Strategy	**Sent**	**Recv**	**Dropped**			
			No route	Link down	$>$ 30 hops	Unknown
Abilene Environment						
$SRS(10\text{ min})$	515900	513103	161	2636	0	0
$SRS(\infty)$	515900	512714	475	2696	5	10
Std Quagga	515900	509885	88	5918	1	8
Std BGP	515900	509441	123	6336	0	0
RFD	515900	496464	14277	5159	0	0
Random Graph Environment						
$SRS(\infty)$	810000	809981	0	0	0	19
Std Quagga	810000	782888	178	26532	43	359
RFD	810000	780812	14001	15186	1	0

Table 5.3: The fates of packets in the software router experiments.

In both environments $SRS(\infty)$ substantially decreases packet loss. In the Random Graph, it is able to avoid all unstable links within the 250-second period before measurements begin, resulting in zero measured packet loss. Although this is an artificial environment, it demonstrates that SRS successfully finds the stable paths. Flap damping, in contrast, fails to find them in a reasonable amount of time; this is somewhat surprising, given the simplicity of the bimodal failure pattern.



(a) Abilene environment

(b) Random graph environment

Figure 5.7: Gap lengths in the software router experiments under the two environments. Packets are spaced at $\approx$ 5-second intervals.

Despite its lower overall drop rate, $SRS(\infty)$ has a slightly longer tail in the gap length distribution (see Fig. 5.8), and more of its dropped packets are due to lacking a route provided by the control plane. This may represent increased convergence time as a result of longer paths. Surprisingly, even though it is given less flexibility, $SRS(10\,\text{min})$ is a win over $SRS(\infty)$ in every type of dropped packet, as well as in the tail of the gap length distribution. $SRS(10\,\text{min})$ still has a longer tail than Standard BGP, but only after the 99.98th percentile. Given its factor 2.3$\times$ reduction in packet loss, $SRS(10\,\text{min})$ presents a promising alternative.

Standard Quagga offers a marginal improvement over Standard BGP in the Abilene environment. RFD also reduces the fraction of packets sent along dead links, which is the largest cause of packet loss in Standard BGP and Standard Quagga. However, RFD pays for this with a large number of packets dropped due to having no route, and we can see from Figure 5.7 that these cause exceedingly long periods of outage in both environments.

Correlation with simulation

The goal of this section is to determine how well our simulator's interruption rate metric predicts data plane performance. We match the interruption rate, measured in simulation with *batching on*, with the packet loss rate of our software router experiments,

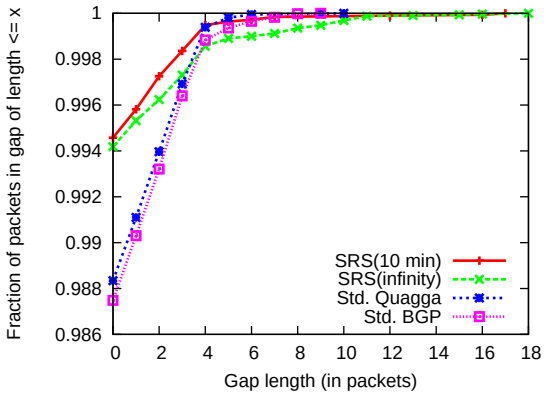


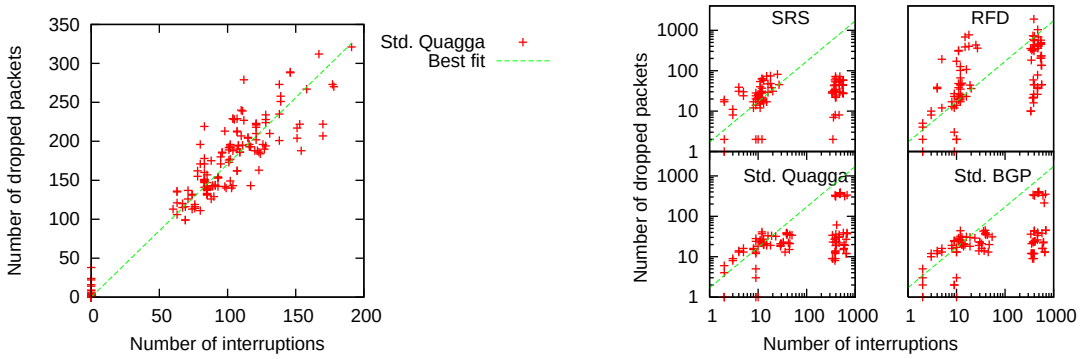
Figure 5.8: Fig. 5.7(a), zoomed in.

both run under the same environments.

We begin with the larger of the two environments, the random graph, using the Standard Quagga strategy. Figure 5.9(a) plots packet loss rate vs. number of interruptions for the 600 source-destination pairs (note that many overlap at $(0, 0)$). Although some variation is to be expected due to implementation details or timing such as the random intervals between packet generation, we see a strong correlation; the data has a correlation coefficient of 0.985896. A least-squares fit of $f(x) = a \cdot x$ yields $a \approx 1.7067$, which means that the average interruption in this environment causes a loss of ≈ 1.7 packets or ≈ 8.5 sec of availability.

But this is an average which does not describe every individual interruption event. Excluding drops due to the underlying network, packets can be lost in our software router for three reasons: having no route to the destination, forwarding along a link that is down, or exceeding the maximum hopcount (i.e., encountering a forwarding loop). As long as a working path actually exists, each of these cases must result from transient conditions involving an interruption. On the other hand, an interruption need not imply packet loss: a router could simply switch between two working routes.

This intuition is borne out in the Abilene environment, depicted in Figure 5.9(b) along with the best-fit line imported from the random graph environment. Abilene has only 110 source-destination pairs, rather than 600; and the trace is more heterogeneous, with a single highly unstable link and several which occasionally fail, compared with 10 persistently stable links. It is then not surprising that source-destination pairs' perfor-



(a) Random graph environment

(b) Abilene environment

Figure 5.9: Correlation between number of interruptions in simulations, and packet loss in experiments, for the two environments.

mance is more highly variable in the Abilene environment. In particular, in all four strategies, a number of points lie near the best fit line; and many points also lie below this line, corresponding to the fact that an interruption does not necessarily cause a packet loss. In the SRS, Standard Quagga, and Standard BGP strategies, fewer points lie far above the best fit line, corresponding to the fact that packet loss within the software router network does not occur without instability. Finally, in RFD, a large fraction of points have many more packet losses than the interruption rate alone would predict. This can be attributed to the fact the simulator counts instability separately from unavailability, the latter being the likely cause of these losses.

5.7 Evaluation with Route Views Update Feeds

In this section we evaluate the performance of SRS using feeds of BGP updates from actual Internet routers, as collected by Route Views [102]. Note that while the simulations of Section 5.5 used traces of AS-level link state availability that were *derived* from Route Views, in this case we use unmodified streams of updates, providing a much more realistic evaluation. However, the deployment scenarios are more limited: only a single node runs SRS, whereas Section 5.5 was able to explore effects of all or a subset of nodes running SRS.

In this evaluation, SRS provides very little benefit for the average IP prefix. We

find that one reason is correlation in paths: for most prefixes, although multiple paths are available, they have a common last link. For prefixes that have path diversity, SRS provides greater improvement.

5.7.1 Methodology

Our evaluation is based on a log of update messages and periodic snapshots of the routing tables (RIBs) from about 42 Internet routers, called *views*, which peer with a Route Views data collector. This data gives us a way to determine the control-plane effect of running SRS at a single router which peers with some subset of these views.

Specifically, our emulator proceeds as follows. In each trial, we create a router which has "virtual peerings" with a random subset of 5 of the Route Views views. This router receives one month of data, beginning with a snapshot of the RIB and proceeding with the update messages that it would receive from each of its virtual neighbors. We emulate the BGP and SRS decision processes over this data, recording interruptions and other measurements on a per-prefix basis. When collecting data we ignore the first $200,000$ seconds (2.3 days) to avoid startup effects.

The results we present are based on 40 trials for each of the two strategies: 10 trials from each month between November 2008 and February 2009. We show 95% confidence intervals.

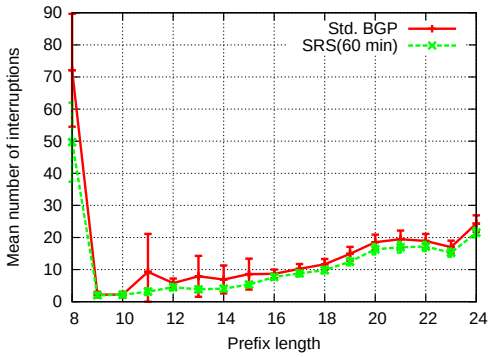
5.7.2 Results

In our experiments, SRS with delay $\delta = 60$ minutes decreases mean interruption rate by 12%:

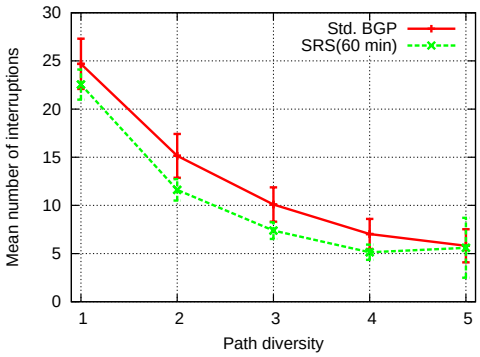
Strategy	Mean interruptions per prefix per trial
BGP	20.67
SRS(60 min)	18.28

We next examine how this result behaves as a function of the prefix length and diversity of available paths (to be defined below). These results are shown in Figure 5.10.

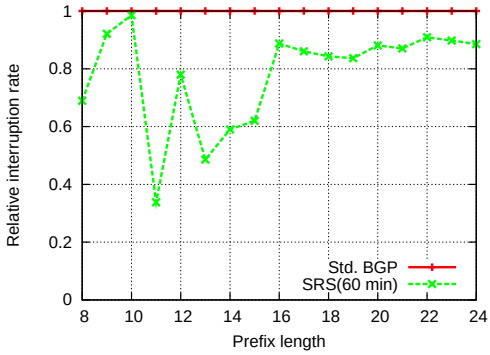
Prefix length. Figure 5.10(c) shows that SRS has similar performance on prefixes of length ≥ 16. For shorter prefixes, the variance in the results is quite high (Fig. 5.10(a)); note that there are relatively few prefixes of length ≤ 16 (Fig. 5.10(e)).



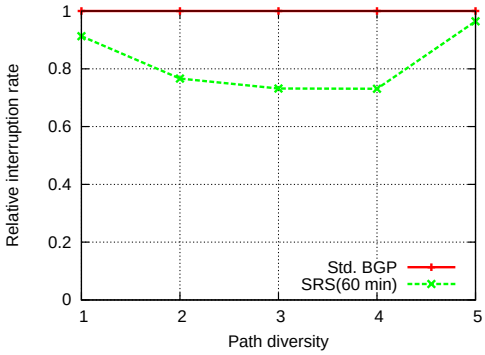
(a) Interruption rate by prefix length



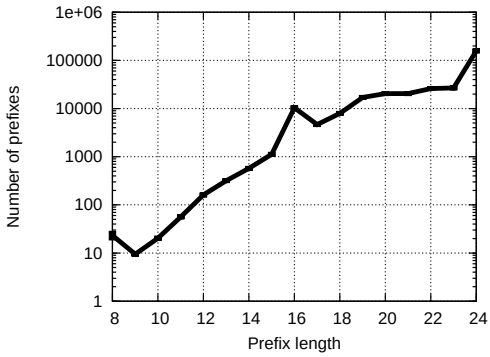
(b) Interruption rate by path diversity



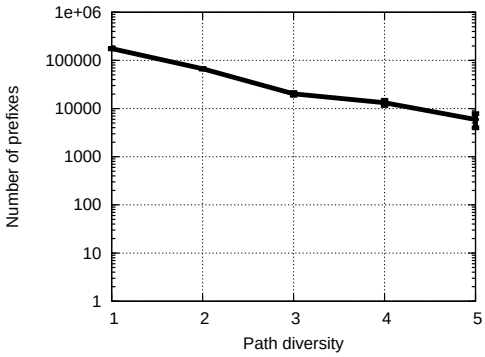
(c) Interruption rate by prefix length, relative to BGP



(d) Interruption rate by path diversity, relative to BGP



(e) Number of prefixes by prefix length



(f) Number of prefixes by path diversity

Figure 5.10: Results of Route Views update feed evaluation.

Path diversity. We define path diversity for each prefix as follows. At a given point in time, the path diversity is the number of distinct *penultimate ASes* among the available paths at that time. (This definition is essentially equivalent to the number of AS-disjoint routes.[1]) We then take the mean of this value across all times that there is at least one available route for the prefix. In plotting results, we group prefixes into bins by rounding their mean path diversity to the nearest integer i; we will refer to these prefixes as having diversity $\approx i$.

Figure 5.10(d) shows that for prefixes with diversity ≈ 1, SRS achieves only an 8.7% reduction in interruption rate compared with BGP. But performance improves substantially once there are multiple disjoint paths, with a 23% or 27% reduction when path diversity is ≈ 2 or ≈ 4, respectively. It is unclear why the interruption rate is *higher* for path diversity ≈ 5; however, note that the variance of the SRS measurements is much higher here (Fig. 5.10(b)). Considering all prefixes with diversity ≥ 2 as a group, there is a 24% reduction in interruption rate.

Conclusions. There are likely several reasons that SRS's improvement over BGP is more limited than our previous simulations suggested. First, as we have shown, SRS performs better with higher path diversity; but about 62% of prefixes have path diversity ≈ 1 (Fig. 5.10(f)). Even if the destination AS has multiple providers, this can occur if the AS is announcing the prefix to only one provider, which is a common way of performing traffic engineering. Our simulator did not consider traffic engineering.

Second, in this section we have emulated a *single* router running SRS, while most of our previous results dealt with the case that all routers run SRS. Performance may improve as a function of the extent of deployment of SRS.

5.8 Related Work

Approaches for improving stability of Internet routing typically fall into two classes: *modifying the routing protocol*, or *modifying route selection*.

Modifying the routing protocol. By appending information about the cause of a route update, the convergence process can be shortened [22, 80]. Loop-free convergence [16] aims to ensure certain correctness properties hold while routing updates are propagating. In-

[1] This is due to the fact that converged BGP routes for any given prefix form a tree; thus, if two routes differ in the hop immediately before the destination, they must differ in *all* hops, except the destination. However, there may be slight differences due to message timing and convergence.

stead of changing the network layer, data packets may be sent on overlay networks which can route around failures [53]. In contrast to these studies, we refrain from modifying BGP, and our SRS strategies focus on avoiding convergence entirely rather than reducing convergence overhead.

Modifying route selection. Some work has studied what path selection policies lead to guaranteed convergence. Griffin et al. [51] showed that a stable state exists if the ASes' policies do not contain a dispute wheel, while [41] showed that by setting policies according to certain locally-checkable guidelines (e.g., preferring customer routes), convergence to a stable state is guaranteed. These studies deal only with guaranteeing convergence due to properties of the policies, rather than link or node failures and recoveries.

By selecting among multihomed connections with low loss, performance and resilience can be improved [4]. We have overlapping goals, but address the problem in the Internet at large, rather than at multihomed edge sites only, and additionally target control plane load.

Perhaps most similar to our work, flap damping [122] suppresses the use of routes which are repeatedly and quickly advertised and withdrawn. Flap damping is known to have pathological behavior that can worsen convergence [80], and as we have seen, can severely impact availability. Duan et al. [32] improve flap damping's performance by recognizing certain sequences of updates that are not indicative of route flaps, but requires an AS to advertise information about its routing policy to neighboring ASes. Recently, Huston [59] proposed delaying updates for short intervals when they match a pattern likely to indicate BGP path exploration. This introduces inconsistent state that can result in loops and outages during the period of delay. All these proposals trade availability for stability, while SRS ensures that if a valid path to the destination exists, one will remain advertised.

5.9 Conclusion

This chapter characterized the space of techniques for improving stability in BGP. One of our main contributions was to develop algorithms to bound the best possible points in the stability-availability and stability-deviation tradeoff spaces.

Our second main contribution was the design and evaluation of a Stable Route

Selection scheme. Experimental and large-scale simulation results show that SRS achieves a significant improvement in control plane overhead and data plane reliability with only a small deviation from preferred routes. However, an evaluation of deployment at a single router using direct feeds of route update traffic from Internet routers indicates only a small improvement. One important, and likely difficult, question for future research is to determine how much improvement SRS could obtain using realisting route update traffic, but with many routers running SRS.

Chapter 6

Conclusion

This dissertation developed practical methods for distributed systems to adapt to and take advantage of heterogeneity. We introduced the Y_0 distributed hash table, which handles arbitrary node capacity distributions; the use of randomization in node selection, which reduces churn amid heterogeneous failure patterns; and Stable Route Selection, which can improve reliability of Internet routing while carefully managing tradeoffs with network operators' perferred routes, by exploiting heterogeneous patterns of route availability.

We also placed these results in the broader context of characterizing the effect of heterogeneity in distributed systems. In the individual systems we considered, we found that route length, congestion, load balance, control overhead, and data plane reliability can be improved in environments that are more heterogeneous in terms of capacity and failure pattern, compared with homogeneous systems of equal total capacity or failure rate. Complementing these examples, we introduced a framework, the price of heterogeneity, in which we exhibited multiple classes of systems in which heterogeneity is *provably* never a significant disadvantage, regardless of the details of workloads and capacity distributions.

In summary, we believe this dissertation makes a strong argument that heterogeneity in reliability and capacity can not only be handled, but rather should generally be viewed as an asset.

6.1 Limitations and Future Work

Our techniques and results are not without limitations. We next discuss some of these issues and future directions that they suggest.

Generalizing bounds on the price of heterogeneity. We bounded the price of heterogeneity of a variety of scheduling and load balancing problems, a low-diameter network construction problem, and the Random Replacement node selection strategy (in a probabilistic generalization of the model). Our results establish large classes of systems for which heterogeneity cannot be detrimental. But a higher-level goal is to generalize across systems: What fundamental characteristics of a system make it perform better as heterogeneity increases? Theorem 1 (p. 17) provides some of the flavor of this objective by providing sufficient conditions for bounding the PoH. Additionally, timing constraints may provide a hint as to what characteristics make a problem sensitive to heterogeneity. In that respect, a major open question is to resolve whether the PoH of precedence constrained scheduling is $\Theta(1)$, $\Theta(\log n)$, or somewhere in between.

Optimizing hierarchies. One of the most common techniques for dealing with heterogeneity is hierarchy. For example, Skype uses a network of high-capacity superpeers to relay calls [52]. The Internet is split into a hierarchy with one protocol (BGP) to route roughly at the level of autonomous systems, and separate intradomain routing protocols to route within autonomous systems. And we used a hierarchy in our Y_0 DHT of Chapter 3, by discarding low-capacity nodes from the DHT's ring, leaving only the nodes of capacity $\geq \frac{1}{2}$ to perform routing.

But what is the optimal point at which to split the hierarchy? As a particular instantiation of this question, in Y_0, what fraction of nodes should we discard from the ring to obtain the best possible performance? Discarding nodes has the negative effect of reducing total capacity that can be used for routing, as well as the positive effects of increasing the mean capacity of remaining nodes and decreasing the mean route length. What is the best tradeoff between these effects? A related question would take into account fairness: What fraction of nodes should be discarded to minimize costs for the nodes remaining in the ring?

Analysis of node selection strategies. In Chapter 4 we analyzed the Random Replacement strategy in a renewal process model, giving a way to determine the churn rate for any

given node session time distributions, under certain assumptions such as a "convergence" property of the distributions, and having a large number of nodes in the system. One area of future work would be to remove these assumptions.

A second area is to analyze other strategies. Despite its importance, we are aware of no comparable statistical analysis for the Longest Uptime strategy. Could it be shown, for example, in what cases RR comes within a constant factor of LU's churn? Can either strategy provably approach the optimal strategy which has knowledge of the future? Additionally, our analysis of the Passive Preference List is limited to the case that k, the number of nodes in use, is large. Our simulations make it clear that Passive PL performs better for small k, so it would be interesting to analytically determine the precise effect of k.

Realistic evaluation of Stable Route Selection with a multiple-router deployment. Our evaluation of SRS using update feeds from Route Views (Section 5.7) showed that SRS provides only a small improvement when deployed at one router. But our simulation results (Section 5.5.2) suggest that wider deployment brings greater overall stability. Quantifying this improvement while using realisting update patterns as in Section 5.7 would be very valuable in bringing SRS closer to real-world deployability.

It would also be helpful to apply our lower bounds algorithms to the direct feeds from Route Views. This would show what points in the stability-deviation tradeoff space are feasible using an environment that is more realistic than our simulation environment (albeit representing deployment at only a single router).

Adaptive routing architectures. The lower bounds of Chapter 5 provide a motivation for future work. Specifically, in our measured Internet-scale topology and inferred trace of failures, it is not possible to improve interruption rate by more than 8.1$\times$ using any route selection strategy, without sacrificing availability or violating policies. And this assumes an unbounded amount of deviation from preferred paths! Thus, in order to improve reliability by orders of magnitude, rather than working within the existing routing architecture as SRS did, we likely need to re-think the architecture with a focus on dependability and dynamic reaction to diverse operational environments.

To effectively react to failures and other events in the network, we require the ability to *obtain feedback* from the environment; a *flexible underlying infrastructure* that permits many possible actions; and *adaptive algorithms* which respond to the feedback within the flexibility afforded by the infrastructure.

The current Internet is deficient in all three of these areas. Automatic routing decisions are largely decoupled from data plane objectives like load balance, latency, and end-to-end availability. To a large extent, the feedback loop goes through humans: as much as Internet routing is automatic, it can also be said to be manual, with operators across the globe tweaking inputs to the BGP decision process to achieve desired traffic engineering or policy effects. This arrangement has a significant cost in human time, and neglects the useful information available to the two endpoints in a connection.

The Internet's routing infrastructure is also very inflexible. End-hosts have no choice in the paths their packets travel, and routers choose among only a fraction of the possible policy-compliant paths. What flexibility does exist is limited further since it may interfere with manual configuration.

A solution to the problems of feedback and flexibility is source-controlled routing: giving end hosts (or their representatives, edge routers) some amount of control over their packets' routes. This gives flexibility to the entities that have access to feedback, potentially yielding huge benefits in reliability and performance. But how close can we come to exposing to end hosts the full diversity of available policy-compliant paths, while still giving network providers sufficient control over their own networks, and allowing the system to scale? This is a significant routing protocol design challenge, and it leads to another question: source-controlled routing shifts the burden of failure detection and traffic engineering onto the end-hosts or other edge devices, which then need smart adaptive algorithms. It is possible that online learning algorithms could be leveraged to help select good routes, potentially with the help of collaboration among end-hosts or routers to share learned information.

6.2 A Final Remark

We leave the reader with one concluding remark.

Networking research has major challenges in providing seamless reliability, performance, scalability, security, and usability. A next-generation Internet architecture would be expected to support these and other objectives over decades of growth and for unexpected applications. In this context it is particularly important that architectural choices be based on a solid foundation. Much of the work in this dissertation has strived towards that goal by combining systems design with theoretical analysis, which can provide guarantees

of the behavior of a proposed design, as well as an understanding of what goals and trade-offs are and are not achievable. We hope that a flow of ideas between the systems and theory communities will flourish in order to meet the needs that arise as computer networks become an ever greater part of society.

Bibliography

[1] Abilene observatory data collections. `http://abilene.internet2.edu/observatory/`.

[2] M. Adler, Eran Halperin, R. M. Karp, and V. Vazirani. A stochastic process on the hypercube with applications to peer-to-peer networks. In *Proc. STOC*, June 2003.

[3] S. Agarwal, C. Chuah, S. Bhattacharyya, and C. Diot. Impact of BGP dynamics on router CPU utilization. In *Passive and Active Measurement Workshop*, April 2004.

[4] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *ACM SIGCOMM*, 2004.

[5] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Árabe, and D. A. Menascé. Using random task graphs to investigave the potential benefits of heterogeneity in parallel systems. In *Proc. ACM/IEEE conference on Supercomputing*, 1992.

[6] Virgílio Almeida and Daniel Menascé. Cost-performance analysis of heterogeneity in supercomputer architectures. In *Proc. ACM/IEEE conference on Supercomputing*, 1990.

[7] Barry C. Arnold. *Majorization and the Lorenz order: A Brief Introduction*, volume 43. Lecture Notes in Statistics, Springer-Verlag, 1987.

[8] Mehmet Bakkaloglu, Jay J. Wylie, Chenxi Wang, and Gregory R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, May 2002.

[9] Hitesh Ballani and Paul Francis. Towards a global IP anycast service. In *Proc. ACM SIGCOMM*, 2005.

[10] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Proc. NSDI*, 2004.

[11] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *NSDI*, 2004.

[12] Marcin Bienkowski, Miroslaw Korzeniowski, and Friedhelm Meyer auf der Heide. Dynamic load balancing in distributed hash tables. Manuscript, 2004.

[13] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer neetworks: Pick two. In *Proc. HOTOS*, May 2003.

[14] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. SIGMETRICS*, 2000.

[15] Andre Brinkmann, Kay Salzwedel, and Christian Scheideler. Compact, adaptive placement strategies for non-uniform capacities. In *Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, Winnipeg, Canada, 2002.

[16] S. Bryant and M. Shand. A framework for loop-free convergence. In *IETF Internet Draft*, 2006. `draft-bryant-shand-lf-conv-frmwk-03`.

[17] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. In *Proc. IPTPS*, February 2003.

[18] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Geometric generalizations of the power of two choices. In *Proc. SPAA*, 2004.

[19] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: routing on flat labels. In *ACM SIGCOMM*, 2006.

[20] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in a cooperative environment. In *Proc. of IPTPS*, February 2003.

[21] H. Chan, D. Dash, A. Perrig, and H. Zhang. Modeling adoptability of secure BGP protocols. In *ACM SIGCOMM*, 2006.

[22] J. Chandrashekar, Z. Duan, J. Krasky, and Z. Zhang. Limiting path exploration in BGP. In *IEEE INFOCOM*, 2005.

[23] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.

[24] F. A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *Proc. 8th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 581–590, 1997.

[25] Ian Clarke. Freenet's next generation routing protocol, 2003. `http://freenet.sourceforge.net/index.php?page=ngrouting`.

[26] Francesc Comellas and Charles Delorme. The (degree, diameter) problem for graphs. `http://www-mat.upc.es/grup\_de\_grafs/grafs/taula\_delta\_d.html`.

[27] Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP*, 2001.

[28] Frank Dabek, Jinyang Li, Emil Sit, Frans Kaashoek, Robert Morris, and Chuck Blake. Designing a DHT for Low Latency and High Throughput. In *Proc. NSDI*, 2004.

[29] Asit Dan and Don Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proc. ACM SIGMETRICS*, pages 143–152, 1990.

[30] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 205–220, 2007.

[31] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simluation of a fair queueing algorithm. In *Internetworking: Research and Experience*, volume 1, pages 3–26, June 1990.

[32] Z. Duan, J. Chandrashekar, J. Krasky, K. Xu, and Z.-L. Zhang. Damping BGP route flaps. In *IEEE International Performance Computing and Communications Conference*, 2004.

[33] C-T. Ee, V. Ramachandran, B-G. Chun, K. Lakshminarayanan, and S. Shenker. Resolving inter-domain policy disputes. In *ACM SIGCOMM*, 2007.

[34] Emulab. Hardware overview, Emulab classic, Accessed February 2008. `https://users.emulab.net/trac/emulab/wiki/UtahHardware`.

[35] A. Feldmann, H. Kong, O. Maennel, and A. Tudor. Measuring BGP pass-through times. In *Passive and Active Measurement Workshop*, April 2004.

[36] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs. Locating Internet routing instabilities. In *ACM SIGCOMM*, 2004.

[37] W. Feng, F. Chang, W. Feng, and J. Walpole. Provisioning on-line games: A traffic analysis of a busy Counter-Strike server. In *Internet Measurement Workshop*, 2002.

[38] Philippe Flajolet, Daniele Gardy, and Loys Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. In *Discrete Applied Mathematics*, pages 207–229, 1992.

[39] P. A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. In *Journal of the ACM*, pages 31–39, January 1974.

[40] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazieres. Oasis: Anycast for any service. In *NSDI*, 2006.

[41] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, December 2001.

[42] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller. Hierarchical P2P systems. In *Proc. ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, 2003.

[43] Michael R. Garey and David S. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[44] Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in dynamic structured P2P systems. In *Proc. IEEE INFOCOM*, Hong Kong, 2004.

[45] P. Brighten Godfrey, Matthew Caesar, Ian Haken, Yaron Singer, Scott Shenker, and Ion Stoica. Stable internet route selection. In *NANOG 40*, June 2007.

[46] P. Brighten Godfrey and Richard M. Karp. On the price of heterogeneity in parallel systems. In *18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, July 2006.

[47] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. In *ACM SIGCOMM*, 2006.

[48] P. Brighten Godfrey and Ion Stoica. Heterogeneity and load balance in distributed hash tables. In *Proc. IEEE INFOCOM*, 2005.

[49] R. L. Graham. Bounds on multiprocessing timing anomalies. In *Bell Sys. Technical Journal*, pages 1563–1581, 1966.

[50] T. Griffin and B. Premore. An experimental analysis of BGP convergence time. In *ICNP*, 2001.

[51] T. Griffin, F. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10(2), April 2002.

[52] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the Skype peer-to-peer VoIP system. In *IPTPS*, 2006.

[53] K. Gummadi, H. Madhyastha, S. Gribble, H. Levy, and D. Wetherall. Improving the reliability of Internet paths with one-hop source routing. In *OSDI*, 2004.

[54] Krishna Gummadi, Ramakrishna Gummadi, Steve Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proc. ACM SIGCOMM*, 2003.

[55] Francis B. Hildebrand. *Advanced Calculus for Applications*. Prentice-Hall, 2nd edition, 1976.

[56] Jingfeng Hu, Ming Li, Weimin Zheng, Dongsheng Wang, Ning Ning, and Haitao Dong. SmartBoa: Constructing p2p overlay network in the heterogeneous internet using irregular routing tables. In *Proc. IPTPS*, 2004.

[57] Ryan Huebsch, Joseph Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proc. of VLDB*, September 2003.

[58] G. Huston. 2005—a BGP year in review. In *21st APNIC Open Policy Meeting*, February 2006.

[59] G. Huston. Damping BGP, 2007. `http://www.potaroo.net/presentations/2007-12-02-dampbgp.pdf`.

[60] Geoff Huston. The BGP instability report, Accessed May 20, 2009. `http://bgpupdates.potaroo.net/instability/bgpupd.html`.

[61] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Leveraging BitTorrent for end host measurements. In *8th Passive and Active Measurement Conference (PAM)*, Louvain-la-neuve, Belgium, April 2007.

[62] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. IPTPS*, 2003.

[63] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[64] David Karger and Matthias Ruhl. New Algorithms for Load Balancing in Peer-to-Peer Systems. Technical Report MIT-LCS-TR-911, MIT LCS, July 2003.

[65] David Karger and Matthias Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *Proc. SPAA*, 2004.

[66] E. Koutsoupias. Coordination mechanisms for congestion games. In *Sigact News*, December 2004.

[67] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley

Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proc. ASPLOS*, Boston, MA, November 2000.

[68] N. Kushman, S. Kandula, and D. Katabi. Can you hear me now?! it must be BGP. In *Computer Communication Review*, 2007.

[69] C. Labovitz and A. Ahuja. Experimental study of internet stability and wide-area backbone failures. In *Fault-Tolerant Computing Symposium (FTCS)*, 1999.

[70] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet routing convergence. In *ACM SIGCOMM*, 2000.

[71] Jonathan Ledlie and Margo Seltzer. Distributed, secure load balancing with skew, heterogeneity, and churn. In *Proc. INFOCOM*, 2005.

[72] Jonathan Ledlie, Jeff Shneidman, Matthew Amis, Michael Mitzenmacher, and Margo Seltzer. Reliability- and capacity-based selection in distributed hash tables. Technical report, Harvard University Computer Science, September 2003.

[73] D. Leonard, V. Rai, and D. Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. In *SIGMETRICS*, 2005.

[74] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of DHT routing tables. In *Proc. NSDI*, 2005.

[75] Jinyang Li, Jeremy Stribling, Robert Morris, M. Frans Kaashoek, and Thomer M. Gil. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *Proc. INFOCOM*, 2005.

[76] T. Li. Router scalability and Moore's law. In *Workshop on Routing and Addressing, Internet Architecture Board*, October 2006.

[77] Gurmeet Manku. Balanced binary trees for ID management and load balance in distributed hash tables. In *Proc. PODC*, 2004.

[78] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[79] Gurmeet Singh Manku, Moni Naor, and Udi Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In *STOC*, 2004.

[80] Z. Mao, R. Govindan, G. Varghese, and R. Katz. Route flap damping exacerbates Internet routing convergence. In *ACM SIGCOMM*, 2002.

[81] Z. M. Mao, R. Bush, T. Griffin, and M. Roughan. BGP beacons. In *IMC*, 2003.

[82] Albert W. Marshall and Ingram Olkin. *Inequalities: Theory of Majorization and its Applications*. Academic Press, 1979.

[83] MetaMachine. eDonkey, Accessed July 2004. `http://www.edonkey2000.com/`.

[84] D. Meyer, L. Zhang, and K. Fall. Report from the IAB workshop on routing and addressing. In *Internet-Draft*, April 2007.

[85] James Mickens and Brian Noble. Predicting node availability in peer-to-peer networks. In *ACM SIGMETRICS poster*, 2005.

[86] Alper Tugay Mizrak, Yuchung Cheng, Vineet Kumar, and Stefan Savage. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In *Proc. IEEE Workshop on Internet Applications*, 2003.

[87] Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. SPAA*, 2003.

[88] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys*, 2004.

[89] A. Odlyzko. Internet pricing and the history of communications. In *Internet Services, L. McKnight and J. Wroclawski, eds., MIT Press*, 2001.

[90] C. Panigl, J. Schmitz, P. Smith, and C. Vistoli. RIPE Routing-WG recommendations for coordinated route-flap damping parameters. In *Document ID ripe-229*, October 2001.

[91] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. In *IEEE/ACM Transactions on Networking*, volume 1, pages 344–357, June 1993.

[92] KyoungSoo Park and Vivek Pai. Comon: A monitoring infrastructure for PlanetLab. `http://comon.cs.princeton.edu/`.

[93] PlanetLab. `http://www.planet-lab.org`.

[94] Quagga software routing suite. `http://quagga.net`.

[95] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, 2001.

[96] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for DHTs: Some open questions. In *Proc. IPTPS*, 2002.

[97] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). In *RFC1771*, March 1995.

[98] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore prototype. In *Proc. USENIX File and Storage Technologies (FAST)*, 2003.

[99] Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker. Fixing the embarrassing slowness of OpenDHT on PlanetLab. In *Proc. WORLDS*, 2005.

[100] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proc. USENIX Annual Technical Conference*, June 2004.

[101] Michael W. Rhodehamel. The bus interface and paging units of the i860 microprocessor. In *Proc. IEEE International Conference on Computer Design*, pages 380–384, 1989.

[102] Route Views project. `http://routeviews.org`.

[103] A. Rowstron and P. Druschel. Storage management and caching in apst, a large-scale, persistent peer-to-peer storage utility. In *SOSP*, 2001.

[104] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. Middleware*, 2001.

[105] Stefan Saroiu. Private communication, 2003.

[106] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. MMCN*, San Jose, CA, USA, January 2002.

[107] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. MMCN*, San Jose, CA, USA, January 2002.

[108] P. Smith and C. Panigl. RIPE routing working group recommendations on route-flap damping. In *Document ID ripe-378*, May 2006.

[109] SprintLink BGP dampening policy. `https://www.sprint.net/index.php?p=policy_bgp_damp`.

[110] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs, and Hui Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proc. ACM SIGCOMM*, 2004.

[111] J. Stewart. *BGP4: inter-domain routing in the Internet*. Addison-Wesley, New York, 1999.

[112] S. Stidham. On the optimality of single-server queueing systems. In *Operations Research*, volume 18, pages 708–732, 1970.

[113] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proc. SIGCOMM*, 2002.

[114] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, 2001.

[115] Jeremy Stribling. Planetlab all pairs ping. `http://infospect.planet-lab.org/pings`.

[116] L. Subramanian, S. Agarwal, J. Rexford, and R. Katz. Characterizing the internet hierarchy from multiple vantage points. In *IEEE INFOCOM*, 2002.

[117] L. Subramanian, M. Caesar, C. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A next-generation interdomain routing protocol. In *ACM SIGCOMM*, 2005.

[118] Sonesh Surana, Brighten Godfrey, Karthik Lakshminarayanan, Richard Karp, and Ion Stoica. Load balancing in dynamic structured P2P systems. In *Performance Eval-*

uation: Special Issue on Performance Modeling and Evaluation of Peer-to-Peer Computing Systems, 2005.

[119] Subhash Suri, Csaba D. Tóth, and Yunhong Zhou. Selfish load balancing and atomic congestion games. In *Proc. SPAA*, 2004.

[120] Garret Swart. Spreading the load using consistent hashing: A preliminary report. In *International Symposium on Parallel and Distributed Computing (ISPDC)*, 2004.

[121] K. Tati and G. M. Voelker. On object maintenance in peer-to-peer systems. In *Proc. IPTPS*, 2006.

[122] C. Villamizar, R. Chandra, and R. Govindan. BGP route flap damping. In *RFC2439*, November 1998.

[123] Michael Walfish, Hari Balakrishnan, and Scott Shenker. Untangling the web from DNS. In *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[124] F. Wang, N. Feamster, and L. Gao. Quantifying the effects of routing dynamics on end-to-end Internet path failures. Technical Report TR-05-CSE-03, University of Massachusetts, 2003.

[125] F. Wang and L. Gao. On inferring and characterizing internet routing policies. In *IMC*, 2003.

[126] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. In *ACM SIGCOMM*, 2006.

[127] Hakim Weatherspoon, Byung-Gon Chun, Chiu Wah So, and John Kubiatowicz. Long-Term Data Maintenance: A Quantitative Approach. Technical Report UCB/CSD-05-1404, EECS Department, University of California, Berkeley, July 2005.

[128] Adam Wierman, Takayuki Osogami, Mor Harchol-Balter, and Alan Scheller-Wolf. How many servers are best in a dual-priority FCFS system? In *Performance Evaluation*, volume 63, pages 1253–1272, 2006.

[129] Bernard Wong, Aleksanders Slivkins, and Emin Gun Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. ACM SIGCOMM*, 2005.

[130] Zhichen Xu, Mallik Mahalingam, and Magnus Karlsson. Turning heterogenity into an advantage in overlay routing. In *Proc. IEEE INFOCOM*, 2003.

[131] Xiangying Yang and Gustavo de Veciana. Service capacity of peer to peer networks. In *Proc. INFOCOM*, 2004.

[132] Bo Zhang, T. S.Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang. Measurement-based analysis, modeling, and synthesis of the Internet delay space, 2006. `http://www.cs.rice.edu/~bozhang/synthesizer.html`.

[133] B. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz. Brocade: Landmark routing on overlay networks, 2002.

[134] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of NOSSDAV*, June 2001.

Appendix A

Proofs for Chapter 2

A.1 NP-Completeness of SIMULATION

Define the problem SIMULATION as follows: given $\alpha \geq 1$, C, and $C' \succeq C$, is there an α-simulation of C with C'?

Fact 3 SIMULATION *is NP-complete.*

Proof: Clearly the problem is in NP. To show NP-hardness we reduce from PARTITION [43]. In that problem, we are given a set S of n positive integers, and must decide whether there exists an $R \subset S$ for which $\sum_{r \in R} r = \frac{1}{2} \sum_{s \in S} s$.

Normalize the elements of S so that $\sum_{s \in S} s = n$. Set $\alpha = 1$, $C = S$ and $C' = (\frac{n}{2}, \frac{n}{2}, 0, \ldots, 0)$. If $C' \succeq C$, then (α, C, C') is a valid instance of SIMULATION, and it is easy to verify that S can be partitioned in half iff there exists a 1-simulation.

If $C' \not\succeq C$, then $\sum_{i=1}^{k} c'_{[i]} < \sum_{i=1}^{k} c_{[i]}$ for some k, where $c_{[i]}$ denotes the ith largest component of C. Since C' has only two positive elements, this must happen for $k = 1$, which implies that $c_1 > \frac{n}{2}$. In that case, there can be no perfect partition of S, so we can map onto any "no" instance of SIMULATION. ■

A.2 Proof of Corollary 3

In the mixed-integer program of Chudak and Shmoys [24], machines are divided into groups of equal speed, and jobs are assigned to machine groups. For our purposes, we

may assume w.l.o.g. that all machines have different speeds, in which case the program becomes the following. Variable $x_{kj} \in \{0,1\}$ represents the assignment of job j to machine k, and $t(j)$ represents the completion time of job j. We seek to minimize the makespan D subject to

$$\sum_{k=1}^{n} x_{kj} = 1 \qquad \forall j \in J \tag{A.1}$$

$$\frac{1}{c_k} \sum_{j=1}^{n} \ell(j) x_{kj} \leq D \qquad \forall k : c_k > 0 \tag{A.2}$$

$$x_{kj} = 0 \qquad \forall k : c_k = 0 \tag{A.3}$$

$$\sum_{k=1}^{n} \frac{\ell(j) x_{kj}}{c_k} \leq t(j) \qquad \forall j \tag{A.4}$$

$$\sum_{k=1}^{n} \frac{\ell(j) x_{kj}}{c_k} \leq t(j) - t(j') \qquad \forall j' \prec_J j \tag{A.5}$$

$$t(j) \leq D \qquad \forall j, \tag{A.6}$$

which can be interpreted as requiring that (1) each job is on some machine, (2) each machine finishes by time D, (3) zero-capacity machines aren't used, (4) the completion time of each job is at least its processing time, (5) precedence constraints are respected, and (6) all jobs finish by time D.

Let LP be the relaxation of this program where $x_{kj} \in [0,1]$, and let $LP(C)$, $LP(C')$, $OPT(C)$, and $OPT(C')$ denote the optimal values of LP and of precedence constrained scheduling, with some given capacities C and $C' \succeq C$ and workload $(J, \ell, \prec_J)$. For any C and $C' \succeq C$, we will show $OPT(C') \leq O(\log n) \cdot LP(C') \leq 2 \cdot O(\log n) \cdot LP(C) \leq O(\log n) \cdot OPT(C)$. The first inequality is due to [24]; the last is due to the fact that LP is a relaxation of PCS. We show the second inequality by verifying the conditions of Theorem 1 for the optimal values of LP. Properties 1 through 3 follow directly, with $\beta = 2$. For Property 4, let (x_{kj}) be an optimal solution to $LP(C)$, and suppose $c'_1 = c_1 + c_2$, $c'_2 = 0$, and $c'_k = c_k$ for $k \in \{3, \ldots n\}$. We show (x'_{kj}) is a feasible solution for $LP(C')$ with the same makespan D and completion times $t(j)$, where for all j, $x'_{1j} = x_{1j} + x_{2j}$, $x'_{2j} = 0$, and for $k \in \{3, \ldots, n\}$, $x'_{kj} = x_{kj}$. Constraints (A.1), (A.3), and (A.6) are obviously satisfied. To verify (A.4) and (A.5), note that the time to process a job doesn't increase:

$$\sum_{k} \frac{\ell(j) x'_{kj}}{c'_k} = \ell(j) \left(\frac{x_{1j} + x_{2j}}{c_1 + c_2} + \sum_{k \geq 3} \frac{x_{kj}}{c_k} \right) \leq \sum_{k} \frac{\ell(j) x_{kj}}{c_k}.$$

Finally, (A.2) is satisfied since

$$\begin{aligned}\frac{1}{c_1'}\sum_{j=1}^{n}\ell(j)x_{1j}' &= \frac{1}{c_1+c_2}\sum_{j=1}^{n}\ell(j)(x_{1j}+x_{2j}) \\ &\leq \max\left(\frac{1}{c_1}\sum_{j=1}^{n}\ell(j)x_{1j},\ \frac{1}{c_2}\sum_{j=1}^{n}\ell(j)x_{2j}\right) \\ &\leq D.\end{aligned}$$

A.3 Proof of Theorem 5

Let $C = (1,1)$ and $C' = (2,0)$. Suppose J consists of mk jobs of size 1 arriving at times $0, 1, \ldots, mk-1$ and m jobs of size k arriving at times $0, k, 2k, \ldots, mk-k$. These can be scheduled as they arrive on the C-machines, for a total response time of $\Theta(mk)$. Now consider scheduling these jobs on the single C'-machine of nonzero capacity. For any schedule, we have one of two cases:

In Case 1, fewer than $1/2$ of the large jobs are scheduled during time $[0, km]$. Then $\geq m/2$ large jobs wait, on average, at least time $km/2$ before they are executed. After normalizing by job length, we have that the total response time of just these jobs is $\geq \frac{m}{2} \cdot \frac{km}{2} \cdot \frac{1}{k} = \Theta(m^2)$.

In Case 2, at least $1/2$ of the large jobs are scheduled during time $[0, km]$. Ignoring all large jobs except these, we can produce an optimal such schedule by setting $t(j_1) = r(j_1)$ for each small job j_1, and inserting each large job j_2 at its specified time $t(j_2)$, delaying the small jobs only as much as necessary. Since each large job takes time $k/2$ to execute on the machine of capacity 2, we must delay starting $\Theta(k)$ small jobs by time $\Theta(k)$ each, so total response time increases by $\Theta(k^2)$. This occurs for each of $\geq m/2$ large jobs that we insert, for a total slowdown of $\geq \Theta(mk^2)$.

Finally, since m is arbitrary, take $m = k^2$ so in either case total response time is $\Omega(k^4)$, compared with $\Theta(k^3)$ for the C-machines.

Appendix B

Proofs for Chapter 3

Proof of Claim 1: A node is discarded only if its current lazily-updated estimate of its capacity is $< \gamma_d$ w.h.p. (where the probability is due to estimation error guarantee) which implies that its real capacity is less than $\gamma_c\gamma_u\gamma_d$. Thus the total capacity discarded is at most $n\gamma_c\gamma_u\gamma_d$, the fraction of the system's capacity discarded is $\leq \gamma_c\gamma_u\gamma_d$, and the claim follows. ■

The next several lemmas are used to prove Theorem 2.

Lemma 2 *If node v has at least one ID in the ring and $\alpha = \Theta(\log n)$, then (1) v has between $\alpha c_v/(\gamma_c\gamma_u) - O(1)$ and $\alpha c_v\gamma_c\gamma_u + O(1)$ IDs w.h.p., and (2) v has at least $\gamma_d\alpha(n) - O(1)$ IDs w.h.p.*

Proof: (1) Note that, due to the estimation error parameters, the factor γ_c lazy update of $\tilde{c}_v$, and the factor 2 lazy update of $\tilde{n}$, we always have $\tilde{c}_v$ within a factor $\gamma_c\gamma_u$ of c_v and $\tilde{n}$ within a factor $2\gamma_n$ of n w.h.p. Thus, the number of IDs that v chooses is at most $\lfloor 0.5 + \tilde{c}_v\alpha(\tilde{n})\rfloor \leq \tilde{c}_v\alpha(\tilde{n}) + O(1) \leq \gamma_c\gamma_u c_v\alpha(2\gamma_n n) + O(1) \leq \gamma_c\gamma_u\alpha(n) + O(1)$, where the last step follows since $\alpha(n) = \Theta(\log n)$. The lower bound follows similarly, noting that we are not concerned with discarded nodes. (2) Similarly, if v has decided to stay in the ring, we must have $\tilde{c}_v \geq \gamma_d$ and the bound follows by the above technique. ■

We now break the ring into **frames** of length equal to the smallest spacing parameter s_{min} used by any node.

Lemma 3 *Each node's spacing is between $\frac{1}{2\gamma_n n}$ and $\frac{2\gamma_n}{n}$ w.h.p., so $s_{min} = \frac{1}{2\gamma_n n}$ w.h.p.*

Proof: A node updates its spacing when it is more than a factor 2 from its current estimate of $1/n$, which is within a factor γ_n of the true value of $1/n$ w.h.p. ■

Lemma 4 *Let $\beta = (1-\gamma_c\gamma_u\gamma_d)/(\gamma_c\gamma_u)$. For any $\varepsilon > 0$, when $\alpha \geq \frac{8\gamma_n}{\beta\varepsilon^2}\ln n$, each frame contains at least $(1-\varepsilon)\beta\alpha n s_{min} - O(1)$ IDs w.h.p.*

Proof: Assume that no node has more than one ID in any frame. If this is not the case, we can break the high-capacity nodes for which it is false into multiple "virtual nodes" without disturbing the rest of the proof.

Consider any particular frame f. Let X_v be the indicator variable for the event that node v chooses an ID in f and let $X = \sum_v X$. We wish to lower-bound X. Suppose v chooses m_v points. Since f covers a fraction s_{min} of the ID space, we have $E[X_v] = m_v s_{min}$. By Lemma 2, $m_v \geq \alpha c_v/(\gamma_c\gamma_u) - O(1)$ for nodes R in the ring. Thus,

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{v\in R}\mathrm{E}[X_v] \\
&\geq \sum_{v\in R} s_{min}\left(\alpha c_v/(\gamma_c\gamma_u) - O(1)\right) \quad \text{(Lemma 2)} \\
&\geq -O(1) + \sum_{v\in R}\frac{s_{min}\alpha c_v}{\gamma_c\gamma_u} \\
&= -O(1) + \frac{s_{min}\alpha}{\gamma_c\gamma_u}\sum_{v\in R} c_v \\
&\geq -O(1) + \frac{s_{min}\alpha}{\gamma_c\gamma_u}\cdot(1-\gamma_c\gamma_u\gamma_d)n \quad \text{(Claim 1)} \\
&= \beta\alpha n s_{min} - O(1),
\end{aligned}$$

with β defined as in the lemma statement. (Note that although Claim 1 was stated in the context of Chord, it applies to our partitioning scheme without modification.) A Chernoff bound[1] tells us that

$$\begin{aligned}
\Pr[X < (1-\varepsilon)\mathrm{E}[X]] &< e^{-(\beta\alpha n s_{min} - O(1))\varepsilon^2/2} \\
&= O(e^{-\beta\alpha n s_{min}\varepsilon^2/2}) \\
&< O(e^{-\beta\alpha\varepsilon^2/(4\gamma_n)}) \quad \text{(Lemma 3)} \\
&= O(n^{-2})
\end{aligned}$$

when $\alpha \geq \frac{8\gamma_n}{\beta\varepsilon^2}\ln n$. Again by Lemma 3, there are at $\leq 2\gamma_d n$ frames, so the lemma follows from a union bound over them. ■

[1] $\Pr[X < (1-\delta)\mu)] < e^{-\mu\delta^2/2}$

Proof of Theorem 2: In the following, we will assume that each frame has at least $r = (1-\varepsilon)\beta s_{min} n\alpha(n) - O(1)$ IDs. By Lemma 4 this occurs w.h.p. for sufficiently large α.

Consider any node v. If v is discarded, its share is 0, so we can assume v is in the ring. We will bound v's share of the ring and then apply this result to all nodes.

We have already broken the ring into $\Theta(n \log n)$ frames; we now divide each frame into d very small **buckets** (we will later take $d \to \infty$). Call a bucket **occupied** when some node's ID lands in it. Next we define a sequence of random variables $X_1, X_2, \ldots$ as follows. Each X_i is 1 when some particular bucket is occupied, and is 0 otherwise. The corresponding buckets are defined as follows:

- X_0 corresponds to the bucket preceeding (i.e., counterclockwise from) node v's clockwise-most ID.
- For any $i > 0$, if X_{i-1} is unoccupied, then X_i is the bucket preceeding X_{i-1}'s bucket. Otherwise X_{i-1} is occupied, and X_i corresponds to the bucket preceeding v's ID that most closely preceeds X_{i-1}.

In other words, the X_i's begin at v's clockwise-most ID, and walk along the ring counterclockwise. But whenever we hit an occupied bucket then we have found the end of the interval of the ID space which is owned by that one of v's IDs, and so we skip to the next one of v's IDs and continue walking counterclockwise from there.

We will use the X_is to analyze v's share of the ID space, but we first have to handle the dependence between the X_is. Define $p_i := \Pr[X_i = 1 | X_1, \ldots, X_{i-1}]$, and $p := 1 - \left(1 - \frac{1}{d}\right)^{r-2}$, where r lower-bounds the number of IDs per frame from the assumption at the beginning of this proof.

Lemma 5 *$p_i \geq p$, for any outcome of $X_1, \ldots, X_{i-1}$.*

Proof: Suppose X_i most closely preceeds node v's ℓth ID. Suppose that $S = X_j, \ldots, X_{i-1}$ is the sequence of Xs located in the same frame as X_i (note all such Xs must be contiguous in this manner). S must take the following form. First, the run of buckets preceeding node v's $(\ell - 1)$th ID may have spilled over into this frame, and then terminated at some node's ID; this corresponds to a run of zeros terminated by a 1 in S. Second, there is a run of zero or more empty buckets preceeding the ℓth ID, until we reach X_i itself. Thus, among S there is at most one 1 and zero or more zeros.

By the assumption at the beginning of the proof of Theorem 2, there are $\geq r$ IDs in X_i's frame. Any zeros in S correspond to empty buckets in the frame, thus constraining the area in which those r IDs may occur, which can only increase p_i; and any ones in S reduce the number of those IDs which may land in X_i's bucket, which reduces p_i. In addition, one ID is that of v itself. Taking the worst case, we have two IDs whose locations are determined and $r-2$ IDs uniform-randomly placed in the frame, so the probability that X_i's bucket is hit is $\geq 1-\left(1-\frac{1}{d}\right)^{r-2}$, as desired. ■

If we see m ones in the first x X_is, then by the definition of the sequence, node v's share of the ring is x buckets, plus m partial buckets which contain v's IDs, for a total of $\leq x+m$ buckets. We now show that we see the required m successes w.h.p. when $x=\frac{m}{p(1-\delta)}$. To do this, we will use a Chernoff bound on the X_is, which the above lemma shows we can do even though the X_is are not independent. Let $X=\sum_{i=1}^{x} X_i$. We have $\mathrm{E}[X]=xp=\frac{m}{1-\delta}$, so

$$\begin{aligned}
\Pr[X<m] &= \Pr\left[X<(1-\delta)\cdot\tfrac{m}{1-\delta}\right] \\
&\leq \exp\left\{-\tfrac{m}{1-\delta}\cdot\tfrac{\delta^2}{2}\right\} && \text{(Chernoff bound)} \\
&\leq \exp\left\{-\tfrac{\gamma_d\alpha-O(1)}{1-\delta}\cdot\tfrac{\delta^2}{2}\right\} && \text{(Lemma 2 part (2))} \\
&= O(n^{-2}),
\end{aligned}$$

where the last step holds as long as

$$\alpha \geq \frac{4(1-\delta)}{\gamma_d\delta^2}\ln n. \tag{B.1}$$

Thus, with probability $\geq 1-O(n^{-2})$, node v owns at most $x+m=\frac{m}{p(1-\delta)}+m$ buckets, each of size s_{min}/d. Normalizing by v's fair share c_v/n, we have

$$\text{share}(v) \leq \frac{1}{c_v/n}\cdot\left(\frac{ms_{min}}{dp(1-\delta)}+\frac{ms_{min}}{d}\right).$$

Since d is arbitrary, we can take the limit as $d\to\infty$. Note that $dp\to r-2$ and by Lemma 4 $r-2\geq(1-\varepsilon)\beta s_{min}n\alpha(n)-O(1)$ w.h.p., so

$$\begin{aligned}
\text{share}(v) &\leq \tfrac{1}{c_v/n}\cdot\tfrac{ms_{min}}{(1-\delta)((1-\varepsilon)\beta s_{min}n\alpha(n)-O(1))} \\
&\leq \tfrac{1}{c_v}\cdot\tfrac{m}{(1-\delta)(1-\varepsilon)(1-\varepsilon')\beta\alpha(n)} \\
&\leq \tfrac{1}{c_v}\cdot\tfrac{\alpha(n)c_v\gamma_c\gamma_u+O(1)}{(1-\delta)(1-\varepsilon)(1-\varepsilon')\beta\alpha(n)} && \text{(Lemma 2 part (1))} \\
&\leq \tfrac{(1+\varepsilon'')(\gamma_c\gamma_u)^2}{(1-\delta)(1-\varepsilon)(1-\gamma_c\gamma_u\gamma_d)}
\end{aligned}$$

with probability $1 - O(n^{-2})$ for any $\varepsilon', \varepsilon'' > 0$ and sufficiently large n. Taking a union bound over all $\leq n$ nodes in the ring, this is true of all nodes with probability $1 - O(n^{-1})$. Finally, we require that α is the maximum of the requirement in Equation B.1 and that of Lemma 4. Setting $\delta = \varepsilon$ for convenience of presentation, we have

$$\max\left\{\frac{4(1-\varepsilon)\ln n}{\gamma_d \varepsilon^2}, \frac{8\gamma_n\gamma_c\gamma_u \ln n}{(1-\gamma_c\gamma_u\gamma_d)\varepsilon^2}\right\} \leq \frac{8\gamma_n\gamma_c\gamma_u \ln n}{(1-\gamma_c\gamma_u\gamma_d)\gamma_d\varepsilon^2},$$

as required by the theorem. ■

Proof of Theorem 4: The bounds on the maximum variation in n and the average capacity are such that each node which joined the system before the start of the period will update its IDs at most once during the period, and new nodes which join will never update their IDs. Since the cost of that movement is amortized over time, it can increase our adaptivity by at most a factor $(1 + \varepsilon)$ for any $\varepsilon > 0$ and a sufficiently long period. It therefore suffices to analyze the load moved directly on or off joining and leaving nodes. Since a fraction $1 - \gamma_c\gamma_u\gamma_d$ of the capacity is not discarded, the expected share of a node v is $\frac{1}{1-\gamma_c\gamma_u\gamma_d} + o(1)$ (the $o(1)$ term is due to rounding the number of chosen IDs to an integer). When v joins or leaves, the perfect partitioner must reassign at least a fraction c_v/n of the ID space, and we move at most a fraction $(\frac{1}{1-\gamma_c\gamma_u\gamma_d} + o(1))c_v/n$ in expectation. Taking the ratio of these, the result follows. ■

Proof of Theorem 5: Consider any node v. We have three cases. First, if $c_v = \Omega(n/\log n)$, then the theorem is true even if v is connected to all nodes. Second, if all nodes except for $O(c_v \log n)$ were discarded from the ring, then the theorem is true even if v is connected to all nodes in the ring.

Otherwise, consider the region R of ID space within distance $\Theta(\frac{c_v \log n}{n})$ of v's IDs.R has size $\Theta(\frac{c_v \log n}{n})$ w.h.p. The expected number of nodes choosing IDs within R is $\Theta(c_v \log n)$ (since we are not in the first or second cases above). A Chernoff bound shows there are *at least* this many nodes in R w.h.p., so that v will need no sequential neighbors outside of R; thus, the number of distinct nodes in R is an upper bound on the number of distinct sequential neighbors that v has. Finally a second Chernoff bound shows that there are *at most* $O(c_v \log n)$ distinct nodes in R w.h.p. ■

Proof of Theorem 6: Assume for now that in each step, the algorithm for construction of the Shared Discretization successfully finds a neighbor that covers at least distance $\frac{k \log n}{n}$ of the ID space for some constant k to be picked later. Then since we are given that $E(I_v)$

can be covered by $f(\frac{n}{c_v \log n})$ contiguous segments of size $\frac{c_v \log n}{n}$, the algorithm will cover I_v while creating at most $\left\lceil \frac{(c_v \log n)/n}{(k \log n)/n} \right\rceil \cdot f(\frac{n}{c_v \log n}) = \Theta(c_v f(\frac{n}{c_v \log n})) = O(c_v f(n))$ edges, where in the last step we used the nondecreasing property of $f(\cdot)$.

It remains only to show that the algorithm does find a neighbor which covers distance $\frac{k \log n}{n}$ in each step, and that it does so without contacting too many nodes. A technique similar to that of Lemma 4 shows that when $k = \frac{\gamma_d}{4\gamma_n}$ and $\alpha = \Theta(\log n)$ is sufficiently large and assuming node capacities are $O(n/\log n)$, the expected number of suitable nodes is $\Theta(\log n)$, and by a Chernoff bound there will be $\Theta(\log n)$ suitable nodes w.h.p. Furthermore, there will be at most $\Theta(\log n)$ node IDs between the owner of p and such a node, so by searching through the successor lists we can find v by contacting $\Theta(1)$ other nodes. ■

Claim 8 *If n random points are chosen on the circle of circumference 1, then the distance along the circle between any two points is at least $1/n^{2+\varepsilon}$ with probability at least $1 - 1/n^{\varepsilon}$ for any $\varepsilon > 0$.*

Proof: Let X_i be the event that the distance from point i to its clockwise successor is less than $1/n^{2+\varepsilon}$. For any setting of the locations of the other $n-1$ points, the area in which point i can fall if X_i occurs is at most $(n-1)/n^{2+\varepsilon} < 1/n^{1+\varepsilon}$. Thus, $\Pr[X_i] \leq 1/n^{1+\varepsilon}$. By a union bound, $\Pr[X_1 \vee \ldots \vee X_n] \leq 1/n^{\varepsilon}$. ■

Appendix C

Proofs for Chapter 4

C.1 Proof of Theorem 8

Lemma 6 *The equation*

$$x = \frac{2}{\alpha d} \sum_{i=1}^{d} \frac{1}{\mu_i} \left(1 - E \left[\exp \left\{ -\frac{\alpha}{2(1-\alpha)} x \cdot L_i \right\} \right] \right) \tag{C.1}$$

has at most one positive solution in terms of x, assuming $\alpha \in (0,1)$ and $E[L_i] < \infty$ for all i.

Proof: It is sufficient that the RHS is concave in x, since in this case the equation has at most two solutions, one of which is always at $x = 0$. We can justify differentiating the RHS w.r.t. x under the integral implied by the expectation, since $\mathrm{E}[L_i] < \infty$ (see [55], p. 365). Thus, the second derivative is $-\frac{\alpha}{2d(1-\alpha)^2} \sum_{i=1}^{d} \frac{1}{\mu_i} \left(\int_0^\infty \exp \left\{ -\frac{\alpha}{2(1-\alpha)} x \cdot \ell \right\} \ell^2 f(\ell) d\ell \right)$ which is clearly negative for all x since $f(\ell) \geq 0$. ■

Lemma 7 *In the stochastic model of Section 4.3.1, churn is equal to $\frac{2}{T\alpha n}$ times the number of failures of in-use nodes.*

Proof: Since all recoveries are instantaneous, each failure of an in-use node increases churn by $\frac{1}{T\alpha n}$ and each subsequent node reselection increases churn by the same amount. ■

Proof of Theorem 8: We will analyze the expected churn using the fact that the number of failures is equal to the total number of times a node is selected. Let L_i be a session time of node i selected uniformly at random over all its sessions in the run of the system. We will abuse notation and refer to L_i as both a session and the length of that session. Let R_i

be the number of reselections during L_i, and let $\beta = 1 - \alpha$. Finally, define

$$S_i = \begin{cases} 1 & \text{if } i \text{ is selected in lifetime } L_i \\ 0 & \text{otherwise.} \end{cases}$$

Since each reselect picks one out of the βn available nodes u.a.r., if there are r selections during a particular node's session, the probability it is selected during that session is

$$\begin{aligned} \Pr[\text{node } i \text{ picked after } r \text{ reselects}] &= 1 - \Pr[\text{none of } r \text{ reselects picks node } i] \\ &= 1 - \left(1 - \frac{1}{\beta n}\right)^r, \\ &\leq (1+\varepsilon)\left(1 - e^{-r/\beta n}\right), \end{aligned} \tag{C.2}$$

for any $\varepsilon > 0$ and sufficiently large n (since $\beta > 0$). Thus, letting c be the steady-state rate of reselections as given by the convergence condition (Definition 4 on p. 75),

$$\begin{aligned} E[S_i] &= E\left[S_i \cdot 1_{(R_i \leq (1+\varepsilon)cL_i)} + S_i \cdot 1_{(R_i > (1+\varepsilon)cL_i)}\right] \\ &\leq E\left[(1+\varepsilon)\left(1 - e^{-(1+\varepsilon)cL_i/\beta n}\right)\right] + \Pr[R_i > (1+\varepsilon)cL_i] \quad \text{(by Eq. C.2)} \\ &\leq (1+\varepsilon)^2 E\left[1 - e^{-cL_i/\beta n}\right] + \varepsilon \quad \text{(since } f_i\text{'s are convergent)} \\ &\leq (1+\varepsilon) E\left[1 - e^{-cL_i/\beta n}\right] + \varepsilon, \end{aligned} \tag{C.3}$$

where in the last step we have reset ε appropriately for notational convenience. Now letting N_i be the number of sessions that node i has in time $[0, T]$, by Lemma 7, we have

$$\begin{aligned} E[C] &= \frac{2}{\alpha n T} \sum_{i=1}^{n} E\left[N_i S_i\right] \\ &= \frac{2}{\alpha n T} \sum_{i=1}^{n} E\left[N_i S_i \cdot 1_{(N_i \leq (1+\varepsilon)E[N_i])} + N_i S_i \cdot 1_{(N_i > (1+\varepsilon)E[N_i])}\right] \\ &\leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \left((1+\varepsilon)E[N_i]E\left[S_i\right] + E\left[N_i \cdot 1_{(N_i > (1+\varepsilon)E[N_i])}\right]\right) \\ &\leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \left((1+\varepsilon)E[N_i]E\left[S_i\right] + \varepsilon E\left[N_i\right]\right), \end{aligned}$$

where the last step follows from the Central Limit Theorem, which we can apply since we have assumed finite mean and variance of the distribution f_i. Now by the Strong Law of Large Numbers, $E[N_i]/T \to 1/\mu_i$ as $T \to \infty$, where μ_i is the mean session time of node i . Thus,

$$E[C] \leq \frac{2}{\alpha n T} \sum_{i=1}^{n} \frac{(1+\varepsilon)T}{\mu_i} \left(\varepsilon + E\left[S_i\right]\right)$$

$$\leq \frac{2}{\alpha n}\sum_{i=1}^{n}\frac{(1+\varepsilon)}{\mu_i}\left(\varepsilon+(1+\varepsilon)E\left[1-e^{-cL_i/\beta n}\right]+\varepsilon\right) \quad \text{(by Eq. C.3)}$$

$$\leq O(\varepsilon')+(1+\varepsilon')\frac{2}{\alpha n}\sum_{i=1}^{n}\frac{1}{\mu_i}\left(1-E\left[\exp\left\{-cL_i/\beta n\right\}\right]\right),$$

for any $\varepsilon' > 0$ and sufficiently large n and T. Since we have assumed (Section 4.3.4) that the nodes are divided into d groups of n/d equivalent nodes, for notational convenience we can say w.l.o.g. that node $i \in \{1,\dots d\}$ belongs to group i, so that the above bound reduces to

$$E[C] \leq O(\varepsilon')+(1+\varepsilon')\frac{2}{\alpha d}\sum_{i=1}^{d}\frac{1}{\mu_i}\left(1-E\left[\exp\left\{-cL_i/\beta n\right\}\right]\right).$$

A similar technique gives the lower bound

$$E[C] \geq (1-\varepsilon')\frac{2}{\alpha d}\sum_{i=1}^{d}\frac{1}{\mu_i}\left(1-E\left[\exp\left\{-cL_i/\beta n\right\}\right]\right).$$

Since these bounds are true for any $\varepsilon' > 0$, the result follows after substituting $c = \frac{\alpha n}{2}\cdot E[C]$ as given in Definition 4 (see Section 4.3.4). The uniqueness of the solution for $E[C]$ follows from Lemma 6. ■

C.2 Worst-Case Analysis of Random Replacement

Definition 6 *A function $f(X)$ is* Schur convex *(resp.* concave*)* in X *when $X' \succeq X$ implies $f(X') \geq f(X)$ (resp. $f(X') \leq f(X)$).*

Theorem 9 ***(Theorem 3.2 in [7])*** *$X' \succeq X$ if and only if $E[X'] = E[X]$ and $E[h(X)]$ is Schur convex in X for every continuous convex function $h : \mathbb{R}^+ \to \mathbb{R}$.*

Proof of Corollary 8: Fixing some j, we must show that the positive solution for x in Equation C.1 (given in the statement of Lemma 6) is Schur concave in L_j. It is sufficient to show that the RHS is Schur concave in L_j for every $x > 0$, and hence that $\mathrm{E}\left[\exp\left\{-\frac{\alpha}{2(1-\alpha)}\cdot x\cdot L_j\right\}\right]$ is Schur convex. Rewriting that expression as $\mathrm{E}\left[h\left(L_j\right)\right]$ where $h(y) := \exp\left\{-\frac{\alpha}{2(1-\alpha)}\cdot x\cdot y\right\}$, we see that h is continuous and convex, so by Theorem 9, $\mathrm{E}\left[h\left(L_j\right)\right]$ is Schur convex. ■

Proof of Corollary 9: Let D be the degenerate random variable which is constantly 1. A

standard fact (see [7]) is that $L \succeq D \cdot \mathrm{E}[L]$ for any L. Thus, since $\mathrm{E}[C]$ is Schur concave in each L_i, the positive solution to Equation C.1 is maximized when L_i is constantly μ_i for all i. In this case, when $\mu_i = \mu$ for all i, Equation C.1 reduces to

$$x = \frac{2}{\alpha\mu}\left(1 - \exp\left\{-\frac{\alpha}{2(1-\alpha)} \cdot x \cdot \mu\right\}\right)$$

whose only positive solutions are $\leq 4/\mu$. Thus, $\mathrm{E}[C] \leq 4/\mu$ while any fixed or Preference List strategy has churn $\geq 2/\mu$. ■

C.3 Facts Concerning Dynamic Strategies

We require a dynamic strategy to keep k nodes in use whenever $\geq k$ are up, and as many as possible otherwise. So during any time period in which $\leq k$ nodes are up, all dynamic strategies have the exact same behavior and the same churn. For the Facts that follow, we may therefore restrict our attention to the case that there are always $\geq k$ nodes up. In this case, each failure and reselect costs $1/k$, and we can think about the strategy of in-use nodes as k *chains* $v^1, \ldots, v^k$ where each $v^i = (v^i_1, \ldots, v^i_{m_i})$ is such that v^i_j is selected in response to the failure of v^i_{j-1}.

A (graceful) *leave* is an event in which the selection algorithm decides to transition a node from *in use* to *available.* By the definition of churn in Section 4.2.1, a failure costs as much as a leave, which results in the following.

Fact 4 *Fix the pattern of failures. Suppose some strategy of node selections (v^i_j) leaves a machine and has churn C. Then there is another strategy (w^i_j) which never leaves and has churn $\leq C$.*

Proof: Let v^i_j be a node which is left at some time t_1 before its next failure at time t_2. Consider two cases: (1) node v^i_j is not used during $[t_1, t_2)$. Then let v^i_ℓ, $\ell > j$, be the node in use in chain i at time t_2 by strategy v. Form strategy w by deleting all nodes in chain i between j and ℓ, and continuing to use node v^i_j during $[t_!, t_2)$. The two strategies differ only during $[t_1, t_2)$, during which v incurs at least the cost of one leave, while w incurs at most the cost of one failure. Case (2): node v^i_j is used again during $[t_1, t_2)$. If it is used again by chain i, we need only delete any intervening nodes in the chain to form w. Otherwise, if it is used by some other chain ℓ, we can swap the chains as follows: chain i stays on node v^i_j until it fails, and then continues following chain ℓ's selections from time

t_2 onward. Chain ℓ, rather than switching onto node v_j^i, follows chain i's former selections. Clearly constructing strategy w in this way cannot increase the total number of failures and reselections. Iterating this argument completes the proof. ■

Fact 5 *With full knowledge of the future, the Optimal strategy of Section 4.2.2 is optimal.*

Proof: We will show that any node selection strategy $v^1, \ldots, v^k$ can be modified iteratively so that at each step churn does not increase, and the resulting strategy is Optimal.

If v is not equivalent to Optimal, then there exists some node v_j^i which is selected at time t_1 and fails at some future time $t_2 > t_1$, and at the time it is selected, there is an available node u which next fails at time $t_3 > t_2$. If u is never in use during $[t_1, t_3)$ then clearly we can use u instead of v_j^i, and then return to chain v^i at time t_3. Otherwise, suppose u is used during $[t_1', t_3)$ for some $t_1' > t_1$ by some chain v^ℓ (note that Fact 4 allows us to assume w.l.o.g. that u is used continuously by one chain until t_3). Then we can modify chain v^i to use u from t_1 until it fails at time t_3, and thereafter follow the former chain v^ℓ; and we can modify chain v^ℓ to follow the former chain v^i beginning at time t_1'. This does not introduce any new failures.

Iterating the above steps results in a strategy identical to Optimal. ■

C.4 Facts Concerning Fixed Strategies

Definition 7 *The decision problem* BEST FIXED STRATEGY *(BFS) is as follows:*

- ***Instance:*** *A set V of n nodes; for each node $v \in V$ a sequence of failure and recovery times $f_1 < r_1 < f_2 < r_2 \ldots$; an integer k; and a rational c.*
- ***Question:*** *Does there exist a set $S \subseteq V$ of $\geq k$ nodes such that, when using S under the given pattern of failures, the churn incurred is $\leq c$?*

One might object that this definition is not realistic, since k does not directly control the number of nodes in use: for example, the definition allows picking a set of k nodes that are always down. But the proofs that follow do not make use of such pathological cases, and transfer directly to the variant of the problem where $|S|$ is unconstrained but we are required to have an average of $\geq k$ nodes in use over time.

Fact 6 BEST FIXED STRATEGY *is **NP**-complete.*

Proof: Clearly the problem is in **NP**. To show **NP**-hardness, we reduce from MAX CLIQUE, an instance of which consists of a graph $G = (V, E)$ and a clique size s. First assume that *exactly* k nodes are required by BFS. We reduce the MAX CLIQUE instance to an instance of BFS as follows:

Set $k = s$. There are $n = |V|$ nodes in the BFS instance identified with the n nodes in G. All nodes are up all the time, except as we will specify. For each pair of nodes $v, w \in V$, we set aside a period of time P_{vw} in the trace during which each node other than v and w fails and recovers, all at independent times. Let $[t_1, t_4]$ be some arbitrary subinterval of P_{vw} during which there are no failures. Then if $(v, w) \in E$, we have v and w fail and recover at independent times during $[t_1, t_4]$. Otherwise, they are down during overlapping periods, according to the following sequence of events, where $t_1 < t_2 < t_3 < t_4$:

- t_1: v fails;
- t_2: w fails;
- t_3: v recovers;
- t_4: w recovers.

Now suppose we pick some set $S \subseteq V$ of k nodes to use. Note that if both v and w are in S and $(v, w) \in E$, or if one of v or w is not in S, then the churn during P_{vw} is $n \cdot 2/k$ since each node fails and recovers at independent times (so each event costs $1/k$). However, if $v, w \in S$ and $(v, w) \notin E$, then v and w have overlapping failures and the churn is $(n-1) \cdot \frac{2}{k} + \frac{2}{k-1} > 2n/k$. Summing over all $n(n-1)$ periods, we have that if S corresponds to a k-clique in G, then the churn is $n(n-1)n\frac{2}{k}$, but otherwise the churn is strictly greater. Thus, asking for a k-clique in G is equivalent to asking whether there exists a fixed set of k nodes such that the churn incurred when using S is $\leq n(n-1)n\frac{2}{k}$.

To handle the case that $> k$ nodes are permissible, we can construct additional failures such that the number of nodes chosen will dominate the churn, forcing $|S| = k$. To do this, in a "fresh" time period with no other failures, we have all the nodes fail sequentially, and then recover sequentially in the reverse order. Regardless of which nodes are chosen, the same pattern arises among the chosen nodes, for a churn during this period of $\Theta(\log |S|)$. Repeating this pattern $\Theta(n(n-1)n\frac{2}{k})$ times is sufficient to ensure that regardless of the pattern of failures representing the graph structure, a smaller S has lower churn, so the optimal S has $|S| = k$. ■

Fact 7 *Picking the k nodes with fewest failure and recovery events is a k-approximation for* BEST FIXED STRATEGY.

Proof: Each event costs ≤ 1, while the optimal strategy must pick a set of nodes with at least as many events, each with cost $\geq \frac{1}{k}$. ■

Appendix D

Proofs for Chapter 5

Definition 8 *The decision problem* MIN INTERRUPTIONS *is as follows:*

- ***Instance:*** *A directed graph $G = (V, E)$; a destination node $d \in V$; for each edge, a set of times that it is available; and an integer k.*
- ***Question:*** *Does there exist a set of routes to d for each node v such that some $v \rightsquigarrow d$ route is in use whenever there is an available $v \rightsquigarrow d$ path, the routes are path-consistent (as defined in Sec. 5.2.1), and the total number of interruptions is $\leq k$?*

Theorem 9 MIN INTERRUPTIONS *is **NP**-complete.*

Proof: Clearly the problem is in **NP**. To show **NP**-hardness, we reduce from SAT. Given an instance of SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct an instance of MIN INTERRUPTIONS as follows. We create:

- a destination d;
- a node for each variable x_i;
- a node for each clause C_j;
- for each variable node x_i, two pairs of edges: $x_i \rightarrow a_i^0 \rightarrow d$ and $x_i \rightarrow a_i^1 \rightarrow d$, where a_i^0 and a_i^1 are intermediate nodes introduced only so that we can have two $x_i \rightsquigarrow d$ routes without using two edges between the same pair of nodes;
- for each variable x_i or $\bar{x}_i$ which appears in clause C_j, an edge $C_j \rightarrow x_i$.

We now set the availability of these edges as follows:

- Edges $x_i \to a_i^0 \to d$ are available during the time interval $[0, 10]$, and edges $x_i \to a_i^1 \to d$ are available during $[5, 15]$.
- If $\bar{x}_i \in C_j$, then the edge $C_j \to x_i$ is available during $[0, 10]$. Otherwise, if $x_i \in C_j$, then $C_j \to x_i$ is available during $[5, 15]$.

Finally, letting ℓ be the number of clauses with both negative and positive literals, we set $k = m + \ell + 4n$.

Having constructed the instance, we now show that there is a set of legal routes which results in $\leq k$ interruptions if and only if the SAT instance is satisfiable.

First, suppose $C_1, \ldots, C_m$ can be satisfied. Set the x_i's equal to a satisfying assignment. We construct routes as follows. For the x_i nodes, note that they must use both $x_i \to a_i^0 \to d$ and $x_i \to a_i^0 \to d$ to cover the whole period of availability $[0, 15]$; the key is *when to switch* from the former to the latter. If x_i is false, we have the switch occur at time 10, so that x_i is uninterrupted during $[0, 10]$; and if x_i is true, we switch at time 5 so that it is uninterrupted during $[5, 15]$. Now consider a clause variable C_j. We have two cases:

- If C_j has both positive and negative literals, it will have an available route during $[0, 15]$. Since the clause is satisfied, it has some satisfied literal, x_i or $\bar{x}_i$. By our choice of routes for the x_i nodes, x_i will have no interruptions during either $[0, 10]$ (if the literal is $\bar{x}_i$) or $[5, 15]$ (if the literal is x_i). Note there is an edge $C_j \to x_i$, so we have node C_j use the route through x_i during this time period. During the remaining interval (either $[10, 15]$ or $[0, 5]$), it can use any of its other routes. Thus, the node C_j has one interruption at time 15 and one at either 5 or 10, for a total of 2 interruptions.
- If C_j has only negative or positive literals, it will have an available route during either $[0, 10]$ or $[5, 15]$, respectively. Following the above argument, it will have a continuously available route during this period, thus experiencing 1 interruption.

The the clause-nodes therefore encounter $m + \ell$ interruptions. There are 2 interruptions on each variable-node x_i, plus one on each a_i^0 and a_i^1. The grand total is therefore $m + \ell + 4n = k$, as desired.

Next, we assume the instance can be routed with $\leq m + \ell + 4n$ interruptions, and show that the SAT instance is satisfiable. Since the x_is and a_is have a total of $4n$

interruptions, the clause-nodes C_j must have $\leq m + \ell$ interruptions. By our construction, this can only occur if clauses with both positive and negative literals have 2 interruptions, and other clauses have 1 interruption. Without loss of generality, we can assume that all interruptions occur at time 5, 10, or 15 (since otherwise route changes can be delayed until the following link state change without increasing the number of interruptions). It follows that every C_j has a route through some x_i which is available continuously for either $[0, 10]$ or $[5, 15]$. From this it follows that we can construct a satisfying truth assignment by setting x_i to be false if x_i switches routes at time 10, or setting it to true if it switches at time 5. ■