Peer-to-Peer Algorithms for Sampling Generic Topologies

Biasing and Parameterizing Random Walks with Doubly Stochastic Convergence and Spectral Estimation

Doctoral Dissertation submitted to the

Faculty of Informatics of the *Università della Svizzera Italiana*

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

presented by

Cyrus P. Hall

under the supervision of

Prof. Antonio Carzaniga

September 2010

Dissertation Committee

Prof. Evanthia Papadopoulou	Università della Svizzera Italiana, Switzerland
Prof. Fernando Pedone	Università della Svizzera Italiana, Switzerland
Prof. Robbert van Renesse	Cornell University, United States of America

Dissertation accepted on 13 September 2010

Prof. Antonio Carzaniga
Research Advisor
Università della Svizzera Italiana, Switzerland

Prof. Michele Lanza
PhD Program Director

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Cyrus P. Hall
Lugano, 13 September 2010

To my parents, Donald Moore Hall and Jane Tracy Hall
. . . your late-night frolicking made this possible.

Abstract

Peer sampling is a fundamental building block of peer-to-peer networking, used to implement search, replication, and monitoring schemes. Yet existing peer-sampling algorithms suffer from one or more of the following problems: they require specially constructed topologies (e.g., gossip peer tables), are hard to parameterize correctly and efficiently (e.g., finding the necessary length of a sample walk), or do not offer precise statistical guarantees (e.g., independent and identically-distributed samples). This dissertation proposes two distributed algorithms to address these problems.

Doubly Stochastic Convergence, the first algorithm, assigns the link transition probabilities of a peer-to-peer network such that a random walk can sample peers uniformly. Using local operations, peers quickly converge to an assignment of probability such that the transition probability matrix representing the entire network is doubly stochastic, a property that guarantees that the sampling distribution is uniform.

The second algorithm, PeerImpulse, estimates the largest magnitude eigenvalues of a peer-to-peer network, which are directly related to the necessary length of a sample walk. PeerImpulse models the peer-to-peer network as a linear dynamic system defined by the transition probability matrix. The algorithm calculates the impulse response of the dynamic system using an efficient local process. The resulting impulse is then used to realize a rank-reduced approximation of the global system at each peer in the network. The high magnitude eigenvalues of this approximation closely match the eigenvalues of the full linear system.

Both algorithms are designed to work on any topology as long as basic connectivity assumptions are met. Neither assumes that new links between nodes can be established, and both attempt to use minimal network resources. Since the second algorithm allows for the on-line parameterization of sampling, sample walks can use the fewest hops necessary to sample with the desired precision. We evaluate both algorithms mainly using analytical and experimental methods, and find them to perform adequately across a wide-range of topologies commonly used to model peer-to-peer networks. We also evaluate Doubly Stochastic Convergence in conjunction with a full implementation of a commonly used distributed hash table, and find it to be resilient to heavy churn. The results of this analysis indicate that it is possible to create generic sampling algorithms that sample from a known and useful target distribution, and support the online parameterization of walk length.

Acknowledgements

First and foremost, I must thank my advisor and friend of the last seven and a half years, Antonio Carzaniga. I have had the privilege to work with Antonio both as a Masters student and as a Ph.D. candidate, and I have never stopped learning while in his presence. My constant goal is to learn at the inhuman rate that he seems to maintain.

I would not be submitting this document were it not for the community of Ph.D. candidates, post-docs, and professors in the Informatics faculty of USI. My special gratitude goes out to those who have made the entire trip from the very beginning of the faculty; your friendship is what kept me from leaving at times. This was not an easy process for me. Not only was much of the material a challenge, but my normal emotional oscillations were often amplified by distance from home. Your ability to listen and discuss was invaluable. I will not name names, as there are too many to list, but you know who you are (I hope).

Much of this work would be much poorer if not the for the help, encouragement, and wisdom of several key individuals. Jeff Rose helped me switch out of what was a dead-end path in sensor networking, and got me reengaged in research again. Shane Legg, who is likely destined to subjugate the human race with machine super intelligence, provided excellent mathematical guidance and encouragement when I was just starting to try and prove the correctness of Doubly Stochastic Convergence. I believe the spectral estimation work never would have happened without his early encouragement. Mark Carman drenched me with a great deal of his insight into statistics, far too little of which was absorbed into this thesis. Other critical players at points included Giovanni Ciampaglia, Laura Pozzi, Tom Shrimpton, and Giovanni Toffetti. No doubt, there are others I am forgetting.

To my committee: thank you for serving. Fernando, thank you for seemingly forgetting my horribly unprepared presentation on SkipGraphs and nevertheless joining the committee. Evanthia, thank you for your positive encouragement, but always honest feedback. And Robbert, thanks for jumping into the middle of the process. I could not be happier that you accepted. Academia would be a better place if there were more people like all of you.

Finally, I want to thank all the other people in my life who supported me. I must apologize to many of you: I am not a great long distance communicator, and have been distant much of these past five and half years in Lugano. This was undeserved. When

I did reach out you always responded with love and care, and for that you will always have a very warm place in my heart. To my Parents... thank you for listening to me bitch and moan for hours on Skype. I always seemed to save my worst for you, and you absorbed it with the unconditional love that I've grown too accustom to. Your years of teaching and care have shaped me into who I am today. I love you both.

Contents

Figures

Tables

Chapter 1

Introduction

One fundamental way to find and aggregate information distributed over a large network is to *sample* the network. If individual computers and their data can be sampled from the network, a picture of the state of the network can be built, and particular results found or computed. This enables useful services like search and aggregation to be built on top of what, at first, appears to be little more than a loose association of computers on a network.

Sampling is a fundamental building block of peer-to-peer networks that has been used to implement a variety of services. Unstructured networks use peer sampling to perform search and replication [32; 54]. Gossip algorithms rely on peer sampling for the efficient converge of estimations [42; 46]. Peer sampling is also used to try and understand the dynamics of networks that are too large for, or hostile to, centralized periodic monitoring [62; 75].

Despite their frequent use, modern peer sampling algorithms suffer from a common set of problems. Existing algorithms often assume underlying topological properties that do not hold in actual deployments, such as undirected links [6; 21; 75], or have limited applicability to a restricted class of network like distributed hash tables (DHTs) [48]. Alternatively, some sampling algorithms build their own independent topologies [42; 43; 82]. Modern sampling algorithms are also hard to parameterize, either because the information necessary for an efficient parameterization is hard to gather or because the sampling results are sensitive to the number of samples required over a time period. For example, the diameter of a network is useful in bounding the neccesary length of sample random walks. However, distributed computation of a general peer-to-peer network's diameter has proven difficult to calculate efficiently, although recent research is promising [56]. Another example are gossip-based sampling algorithms, which produce samples from a distribution that is thought to be near uniform. When a large number of samples are simultaneously requested, or the network experiences message loss or churn, the sample distribution becomes distorted, returning samples which are not identically distributed [77].

Our goal is to support topology-agnostic sampling algorithms that do not require additional links beyond those already present in the network and that can parameterize themselves in an on-line fashion after deployment. A peer sampling algorithm that accomplishes these goals would be truly *modular* and could run on a wide variety of peer-to-peer networks without modification. In this dissertation we develop two algorithms that address the problems of existing sampling schemes and move toward these goals. Taken together, the two algorithms we develop offer a platform to build a modular sampling system. The first biases a peer-to-peer network such that it can be sampled uniformly, while the second leads to a method for efficient parameterization of the sample walks.

1.1 Biasing for Peer Sampling

Our first distributed algorithm, *Doubly Stochastic Convergence* (DSC), iteratively transforms the global transition probability matrix of a peer-to-peer network such that it converges to be doubly stochastic. This transformation is accomplished via efficient local operations, with messages only sent to neighboring peers. The resulting doubly stochastic matrix assures that a random walk that uses the assigned transition probabilities will end at a particular peer in the network with uniform probability.

1.1.1 A Minimal Set of Assumptions

Our design goal was to develop an extremely versatile and unobtrusive method for biasing the network for peer sampling. In particular, we wanted to minimize the assumptions that the algorithm made about the underlying peer-to-peer network. Previous work has made one or more assumptions that seem, to us, both undesirable and unnecessary. First, previous biasing algorithms have all assumed that every link in the underlying peer-to-peer network is *bidirectional* (i.e., the graph is undirected) [6; 21; 75]. This is unrealistic in at least two respects. Peers are often hosted on networks that use so-called middle-boxes. Both network address translation (NAT) and firewalls can create situations in which a peer can send data to a neighbor, and yet is itself unable to receive data from the same neighbor. Most peer-to-peer networks assume some form of NAT-traversal is available and that peers can try and establish symmetric connections. However, such measures are slow, may require a third-party server, and are error and failure prone [29]. We did not want to limit the usefulness of DSC for networks that make use of directed links and do not have the need to bypass NAT or firewalls.

Similarly, most existing gossip-based peer-sampling algorithms implicitly require the ability to form symmetric connections to new peers at the beginning of every exchange round. Like the assumption of symmetric links in the underlying peer-to-peer network, this ignores the reality of firewalls and NAT, and the costs they impose.

Second, existing biasing algorithms also assume that each peer tracks both incoming

and outgoing links from neighboring peers, yet most peer-to-peer algorithms create link-structures that are logically *asymmetric*. This fact is often unstated, laying implicit in a particular algorithm.

A common class of peer-to-peer networks that do not need to track incoming links are DHTs. It is highly probable that a link in a DHT route table is asymmetric, such that peer x points at peer y, but peer y does not point back. On top of this, a link in a route table may be evicted before ever being used, such that it is unnecessary to contact a node just because it has been inserted into route table of a peer. This problem can be managed by annotating links, marking them as having been contacted or not, and by tracking which upstream neighbors have contacted a peer. However, such a scheme still requires a message to notify a neighbor when it is no longer being referenced. Since existing bias algorithms require an accurate count of the number of edges at each peer, such tracking would be mandatory in practice, adding complexity and overhead.

Unlike previous work, DSC only makes the assumption that the underlying topology is *ergodic*. In short, this means that there is a path from each peer to all other peers. There is no assumption that links are undirected, nor that new links can be created beyond those present in the underlying peer-to-peer network. Instead, DSC relies exclusively on the links provided to it by the underlying peer-to-peer network.

We believe the assumption of ergodicity is minimal, and that it can not be relaxed further. If a network is non-ergodic some nodes will be unable to sample others. Under such conditions it is not possible to generate identically distributed samples, as different parts of the network will have different reachability properties.

1.1.2 Synopsis of the DSC Algorithm

While DSC is described in detail in Chapter 4, we briefly outline it here. The goal of the algorithm is to assign the transition probabilities of a random walk such that the sum of incoming and outgoing probabilities are both equal to 1 for every peer. In other words, the columns and rows of the transition probability matrix of the random walk both sum to 1 after DSC has converged, making the transition matrix doubly stochastic. Since a doubly stochastic matrix is guaranteed to have a uniform stationary distribution, such a matrix assures a uniform sample distribution across all peers in the network, assuming the sampling random walk is of sufficient length.

A node cannot assume it has control over, or even influence, incoming transition probability, as communication may be unidirectional. This fact makes the core challenge of designing DSC to find a way to balance *incoming* probability by only changing the distribution of *outgoing* probability. The key tool that enables DSC to balance incoming and outgoing probability is a *self-loop* at each peer, a virtual edge that loops from a peer back to itself.

Since the self-loop is added in both the total sum of incoming and outgoing probability, a peer can use the self-loop to change both sums. By recognizing that a peer only has direct control over the probability distribution across its outgoing links, DSC

uses the self-loop to manipulate incoming probability and then adjusts the outgoing transition probabilities appropriately. DSC must maintain the invariant that the sum of outgoing probability equals 1. As such, when the self-loop is increased, the probability to transition across the other outgoing links must be lowered, with the inverse and converse situations holding as well.

From of these constraints comes the core of the DSC algorithm. When the sum of incoming probability at a peer is less than 1, DSC increases the self-loop to bring the sum equal to 1. At the same time, the sum of transition probability across the out-going links other than the self-loop is lowered by the same amount. After this rebalancing of probability, both the sum of in- and out-probability of the peer equal 1, and the peer is balanced. In basic DSC this is the only action related to the self-loop that is taken, although, as we will see in Chapter 4, churn and efficiency concerns complicate the situation.

Clearly (and axiomatically), probability is conserved after each update, with no probability introduced or subtracted. As updates continue, incoming probability is reduced at all nodes as peers with a deficit increase their self-loops and lower probability across their other outgoing links. These reductions of probability are absorbed by the peers with surplus incoming probability. Indeed, the ergodicity of the graph guarantees that a reduction of out-probability of one peer will be absorbed by some set of peers with surplus incoming probability, as the balance between the sum of in- and out-probabilities over the entire graph is invariant. A proof of convergence is offered in Chapter 4.

In order for each node to know the sum of in-probability, DSC sends probability update messages to all downstream neighbors once or more per round. Rounds can be asynchronous without slowing convergence, as long as all peers receive updates from their upstream neighbors within a single round. Failure to do so slows, but does not halt, convergence, as long as an update message is eventually received.

We stress that DSC is a *biasing* algorithm which changes the probability to transition from one peer in the network to another peer, in particular, a peer that is an immediate neighbor. This is similar to previous work that has also focused on biasing [6; 21; 75]. Building a sampling algorithm on top of an already biased network largely consists of taking random-walks of an appropriate length, where the length is determined by desired accuracy with respect to the target distribution [62]. Our second algorithm came about from an attempt to determine the necessary walk length in situ.

1.2 Spectral Estimation

Most algorithms that utilize sampling either make a priori estimates of the maximum necessary walk length, usually based on the expected diameter of the network, or use an iterative scheme that increases walk lengths when an operation such as search fails, up to some hard-coded maximum [17]. Neither scheme is ideal, as they both use more resources than necessary. Further, with the proposal of topologies that reorganize in

response to network conditions [87], the necessary length of a random walk may change over time as a network responds to attacks or other conditions. An algorithm that can estimate necessary walk length after deployment would therefore be quite useful.

One common technique to bound the necessary walk length over a graph is to make use of the *spectral gap*, that is, the distance between the two largest-magnitude eigenvalues in the spectral decomposition of a graph [16; 53; 74]. However, since such decomposition has traditionally required knowledge of the entire graph, such bounds have been calculated off-line before deployment, on either synthetic topologies from simulation or based on theoretical analysis. The core contribution of this part of the dissertation is an algorithm that can efficiently estimate the spectral gap in a deployed network without gathering global topology information at each peer.

We contribute a distributed algorithm that uses identification theory to locally estimate a linear dynamic system that approximates the spectral dynamics of the entire peer-to-peer system. The approximated system is then decomposed, and the largest magnitude eigenvalues used to estimate necessary walk length. Importantly, this algorithm is not specific to DSC, and can be used with any weighting of the links. Further, estimating the necessary walk length is only one use of the eigenvalues of a network, and the technique may prove useful to other estimation procedures [16].

1.2.1 Assumptions

The basic assumptions made by our spectral estimation algorithm are the same as those made by DSC: the underlying topology must be ergodic, links can be directed, and no additional links may be created beyond those present in the underlying peer-to-peer network.

The algorithm makes no assumption about the sum of in-probability. However, our initialization procedure currently assumes that the underlying system is stochastic, or can be normalized to be so. This is because the algorithm grew out of the desire to analyze transition probability matrices, and our initialization procedure converges to a value particular to such matrices. However, with modifications to the initialization procedure, we believe it should be possible to expand its scope to non-stochastic matrices. These caveats are covered in more detail in Chapter 5.

1.2.2 Synopsis of the PeerImpulse Algorithm

The core idea of the spectral decomposition algorithm is to treat the network as a discrete-time linear dynamic system and to compute the impulse response of the system. We have therefore named the algorithm *PeerImpulse*. The impulse response of a linear dynamic system can be used by various well known realization algorithms to recreate or approximate the original system. The quality of the approximation is, in the absence of noise and errors, based on the length of the impulse and the rank of the original system.

The operation of PeerImpulse is broken into three phases: the initialization of a distributed state-vector, the creation of the impulse response, and the realization of an approximate system at each peer. We now discuss each phase briefly.

In order to create the impulse response, we need to initialize the state of the linear system. PeerImpulse uses a single-round initialization procedure to set the values of the state-vector, where each peer holds a single specific entry in the state-vector. The initialization procedure is designed such that, with high probability, the impulse response is usable with the Ho-Kalman realization method, and that undesired parts of the graph spectrum are eliminated from the final spectral estimation.

Once the state-vector is initialized, PeerImpulse creates the impulse response via multiple rounds of a simple distributed process. Importantly, each round in the process can be calculated locally, with each peer sending messages to its immediate neighbors. In order to update a peer's entry in the state-vector the peer only needs to know the transition probabilities of its upstream neighbors, and the current value of their entry in the state-vector. Similar to DSC, a simple set of update packets is used to notify downstream neighbors of both these values. A peer then updates its entry in the state-vector, and the next round begins.

Every peer records the value of its entry in the state-vector after each update. The vector holding these recorded values is the contribution to the impulse response of that peer. After a significantly long impulse trace has been generated, each node uses this *local* contribution to realize an approximate system. The realization is performed with Kung's algorithm [52], a closely related variation of Ho-Kalman realization [40]. By using only the local contribution to the global impulse response, we avoid having to gather all the individual traces at each node, but in return introduce error into the realization process. As shown in Chapter 5, this error is minimal, and can be easily reduced further by a single gossip round of the eigenvalues of the realized approximate system.

1.3 How to Read this Dissertation

The dissertation is broken into four main parts beyond this introduction. The first part consists entirely of Chapter 2, and introduces the high-level concepts of peer-to-peer networking. Part two consists entirely of Chapter 3, and describes sampling in peer-to-peer networks. It first formally defines the problem and then discusses theory and related work. Chapters 4 and 5 make up the third part, describing the main contributions of the dissertation, Doubly Stochastic Convergence, and PeerImpulse. Finally, concluding thoughts and future directions are given in Chapter 6.

A reader knowledgeable in peer-to-peer systems may safely skip Chapter 2, as it contains no description of the contributions of the dissertation not already found in this introduction. Chapter 2 first offers a broad definition of peer-to-peer computing, and then discusses various principles that define the area. These principles very much

influenced the design of the algorithms in Chapters 4 and 5. Section 2.1 may be of interest to those with a background in distributed systems, as it briefly describes the relevant differences between peer-to-peer research and that area.

Chapter 3 reviews the theory used throughout the dissertation, including the graph theory, Markov-chain theory, and linear algebra used in Chapters 4 and 5. The exception is the description of discrete-time linear dynamic systems, which we delay until Chapter 5 where it is used. Sampling is introduced in Section 3.1, Section 3.2 discusses random walks as Markov chains, and Section 3.3 introduces spectral theory and analysis. After this introductory review of theory, Chapter 3 then discusses the most significant related work. Section 3.4.1 and 3.4.2 discuss related sampling algorithms, looking at random walk and gossip algorithms, respectively.

The core contributions of the dissertation begin in Chapter 4, which discusses the Doubly Stochastic Convergence algorithm. We first briefly provide additional theoretical exposition of the interesting properties of doubly stochastic matrices in the context of distributed sampling. The next part of the chapter details the design of DSC, providing both intuitive descriptions and formal proofs to demonstrate correctness under ideal conditions. The second half details results from extensive experiments designed to confirm the correctness and efficacy of the algorithm. Our experimental focus is on questions that could not be easily answered theoretically.

Chapter 5 presents the second major contribution of the dissertation, and discusses the PeerImpulse algorithm. The chapter is roughly split into three parts. Part one introduces discrete-time linear dynamic systems and the system realization problem. The second part uses the generalization of Markov-chains to linear-systems to develop a distributed algorithm for calculating the truncated impulse response of a linear system, and then uses it to perform system realization. The final part analyzes the resulting realized systems, and shows that their spectrum closely matches that of the entire peer-to-peer system.

Finally, Chapter 6 concludes the dissertation by briefly reviewing the presented work and offering a number of future research directions.

Chapter 2

Peer-to-Peer Networking Overview

In this chapter we attempt to give context to the work presented in this dissertation. In particular, we discuss *peer-to-peer networking* (P2P) as a method for utilizing distributed resources. Our discussion begins with definitions and then focuses on the unique properties and challanges of developing algorithms for peer-to-peer networks. We conclude by offering a brief overview of the two major classes of existing peer-to-peer networks, structured and unstructured networks, giving examples of each.

For a much more detailed overview of peer-to-peer computing than what is offered in this chapter, we suggest the technical report of Milojicic et al. entitled "Peer-to-Peer Computing" [59].

2.1 A Brief Introduction to Peer-to-Peer Networking

At the most general level, the field of peer-to-peer computing is focused on the study of distributed algorithms in which there is *cooperation* between participants. Participants in such networks are called *peers* or *nodes*; the terms are used synonymously in much of the literature and this document. Peers share one or more classes of *resources*, such as processor cycles, disk and memory storage, network bandwidth, etc., which are used by one or more distributed algorithms to provide desired functionality. A peer can be viewed as the basic aggregate unit of resources in a peer-to-peer system.

A *peer-to-peer network* defines a specific method of organizing peers into a *topology*, upon which one or more algorithms may run. The word topology is meant to invoke the notion of a class of graphs, such that any specific network graph produced by a particular peer-to-peer network has similar graph-theoretic properties to other graphs that are produced by the same network. The distribution of links in a topology typically follows a particular pattern, chosen such that the topology exhibits emergent properties beneficial to the particular algorithms the peer-to-peer network is trying to support [17; 42; 49; 57; 76].

A *peer-to-peer system* brings together one or more peer-to-peer networks, forming a

more complex system. A single computer will often play the role of different peers in each of the peer-to-peer networks that make up a peer-to-peer system. The combination of various topologies allows different classes of algorithms to be run, possibly over the same data, and allows for the implementation of interesting applications and services.

There is often a strong relation between a peer-to-peer topology and the distributed algorithms that can be run efficiently on it. It is very common to see a particular pairing of topology and algorithm inclusively referred to as a peer-to-peer network. For example, an algorithm that assumes the existence of peers with special roles is clearly limited to the subset of topologies that contain such special peers. The goal of the work in this dissertation is the opposite. We desire to find algorithms that can run on the largest set of topologies as possible.

While the above definitions are generally accepted in the peer-to-peer research community [1; 10; 20; 59; 68], a tighter definition is useful in the context of this work. We call a peer-to-peer algorithm *pure* if peers in the underlying topology can be added and removed (i.e., the network experiences *churn*) arbitrarily without the algorithm failing [68]. In other words, there is no special set of nodes that must be present for the algorithm to work. Similarly, we define a pure peer-to-peer network as one that generates a class of topologies with uniform or near-uniform peer role assignments.

Peer-to-peer networks and algorithms in which an essential subset of peers exists are said to be *hybrid* networks [68]. Hybrid networks typically have one or more strongly differentiated roles for various subsets of peers, allowing algorithms to make assumptions about resource availability. Often role differentiation is related to the class of resource being shared. For example, one set of peers may handle storage, and another provide computational power, a differentiation of resources often seen in volunteer computing projects [4; 7; 81]. In other networks roles are differentiated across functionality, as is done in peer-to-peer file-sharing services such as Napster and BitTorrent, both of which separate indexing, and storage and download [19; 20; 27].

Pure peer-to-peer networks and algorithms tend to generalize better. As our core interest is the sampling of generic peer-to-peer topologies, it makes sense for us to focus on the creation of pure peer-to-peer algorithms. It is therefore unsurprising that the algorithms presented in this dissertation are targeted at pure peer-to-peer networks.

2.1.1 Peer-to-Peer Compared to Client-Server Computing

The above definitions of peer-to-peer computing can be contrasted to the familiar *client-server* model, where a server provides all the resources necessary to fulfill requests from one or more clients. As has been pointed out by others [20; 59], the lines between these paradigms can become quickly blurred. Is a world-wide-web server farm that uses a distributed cooperative caching system a peer-to-peer system?

Realistically, modern systems are often a mixture of the two paradigms. One clear example of this phenomenon are the large data centers used by major web sites, for example, Amazon.com. Users access Amazon via the Hypertext Transfer Protocol (HTTP),

a client-server protocol. Yet, in order to fulfill a request, the web-server at Amazon relies on a large ecosystem of networked systems. Some make use of the classic client-server model (requests to internal services), yet others fit the pure peer-to-peer network model (e.g., Dynamo, for semi-persistent storage [23]). Google similarly makes use of a diverse set of architectural network paradigms [12; 22; 31].

2.1.2 Peer-to-Peer Networks as Overlay Networks

Peer-to-peer networks are typically constructed as *overlay networks*. Overlay networks add an additional level of routing-logic above the transport and network layers (i.e., TCP/IP), in what is often called the application layer, and provide state maintenance and management algorithms. Overlays usually define some set of forwarding functions to forward overlay packets using this state. The combination of the route-state maintenance and forwarding schemes define the service implemented by the overlay. In order to implement a higher-level of routing and forwarding, many overlay networks define an additional level of *logical routes* above what the networking layer already provides. Each logical route is typically composed of multiple network-layer links, i.e., multiple IP-level routes.

Overlay constructions are typically sub-optimal when compared with what could be achieved if the same service was implemented directly in coordination with the network layer [11; 15]. In return for this inefficiency, developers gain design flexibility and ease of deployment. It also separates design concerns. Indeed, while overlays are positioned at the application layer, it is more fitting to think of them as a distinct layer implementing higher-lever routing and transport services. This separation of concerns partially decouples design, implementation, and optimization from the network and transport layer. The programmer is then free to focus solely on whatever problems are at hand, while the complexities of network-level routing and transport are encapsulated in the lower layers.

The use of an overlay network in the design of new Internet services bypasses the need for strong social coordination during deployment. This has proven important in the continued evolution of the network, as individual autonomous systems and end-users can begin (or end) providing services in a piece-wise fashion. Multicast serves as a strong example of why such piece-wise role out is critical for eventual deployment.

Examples of systems that benefit from the separation of concerns when designed as overlay networks include application-level multicast [15], the Resilient Overlay Network (RON) [3], and The Onion Routing project (TOR) [25], all of which can also be considered peer-to-peer networks. Historically, the overlay paradigm has been popular with core Internet services, including network news (NNTP) [28], the domain name service (DNS) [60], and e-mail (SMTP) [50].

In general, networks that form overlays often exhibit the properties of peer-to-peer networks and reflect similar design principles. The two terms are sometimes used in a nearly synonymous fashion. However, there is nothing fundamental about peer-to-peer

networks that disallows their deployment at the network or transport level. Indeed, IP-multicast can be seen as a hybrid peer-to-peer architecture, even though it rests at the network level [24].

2.1.3 Principles of Peer-to-Peer

In research, the boundary between peer-to-peer research and what has traditionally been called distributed systems is blurry [20; 59]. For the purpose of making this differentiation clear, it may be more useful to discuss the principles of peer-to-peer systems rather than architectural properties, as we have done above. Aberer and Hauswirth give three important principles of peer-to-peer computing [1]:

- *Principle of sharing resources*: As discussed above, peer-to-peer systems involve resource sharing. In particular, a common goal of peer-to-peer systems is to aggregate resources such that otherwise unimplementable services are realizable.
- *Principle of decentralization*: While hybrid and pure peer-to-peer systems may vary in their level of decentralization, the goal of peer-to-peer research is to be as decentralized as possible while still providing the desired functionality. Indeed, decentralization often enables applications that would otherwise be hard to realize [4; 23].
- *Principle of self-organization*: Peer-to-peer systems must self-organize to a greater extent due to their decentralization.

To these, we add the *principle of local and redundant state*. Both machines and users are highly faulty in wide-area peer-to-peer deployments. Programs crash, users quit, and packets are lost. The level of failure measured in wide-area peer-to-peer networks far out-paces what traditional distributed algorithms are designed to handle [73] and failure is often assumed to be byzantine. Therefore, the state needed for a distributed algorithm to make progress must be local to a peer, such that it does not depend on other nodes elsewhere in the network. State must also be redundant, such that if a particular piece of information is out of date, an equivalent or less optimal alternative can be used.

2.2 Major Classes of Peer-to-Peer Systems

Traditionally two main classes of peer-to-peer architectures have been identified, commonly referred to as *structured* and *unstructured* peer-to-peer networks. We consider these names to be historical in nature, and see them as inappropriate in light of the modern understanding of each. Indeed, *both* architectural classes are structured, in that both types of networks organize links around fundamental properties of their link structures. “Structured” peer-to-peer networks organize their link structures around metric-spaces

(e.g., DHTs), while "unstructured" networks are often organized around graph-theoretic properties that offer opportunities to design algorithms with probabilistic behavior (e.g., small-world, power-law, and normally-distributed networks [49; 87]). Networks that attempt to fuse multiple link-distributions architectures also exist [35; 65]. Even though we find the category names misleading, for consistency with other work in the field we will retain the terms in this dissertation.

We now briefly summarize each category and for each give an example of a major class of systems that fits within the category. This is not intended to be in any way a thorough survey of either class of network, but instead to serve as a quick introduction.

2.2.1 Structured Peer-to-Peer

Structured peer-to-peer networks organize their link structures around metric-spaces, often the metric-space of a class of data that is to be searched over [5; 72]. Topologies of structured networks are typically designed to divide the metric-space in such a way as to balance the complexity of search and the size of the link-table. This is typically accomplished by dividing the metric-space near-uniformly over all members of the network, with each peer taking responsibility for data located in its share of the metric-space. Links between peers are then structured such that greedy search algorithms can quickly locate the neighbor responsible a given value in the metric-space, often in logarithmic time.

Example: Distributed Hash Tables

The oldest type of structured networks are the *distributed hash tables*, or DHTs for short [57; 63; 66; 72; 85]. DHTs typically represent a finite linear metric-space, such as the natural numbers $\mathbb{N}$ or euclidean coordinates, and provide two methods for accessing the space. The first, PUT(id, val), stores an untyped value *val* at the node responsible for the part of the metric-space containing *id*. The second function, GET(id), performs the opposite operation, retrieving the value associated with *id*.

Both PUT and GET operations typically take $O(\log n)$ hops to find the appropriate node. This derives from the structure of links at each peer. Most DHTs adopt a strategy of exponentially distributed link-tables [57; 63; 66; 72; 85]. Most DHTs attempt attempt to exponentially increase the distance between the local id and each destination-node id in the link-table. For example, for a DHT over the natural numbers $\mathbb{N}$ from $0 \ldots 2^{64}-1$, which uses the modular distance function $d(id_1, id_2) = (id_2 - id_1) \bmod 2^{64}$, the sequence $\{1, 2, 4, \ldots, 2^{32}\}$ gives one possible set of exponential increasing distances in a link-table. In addition to these links, DHTs often must keep links to their closest neighbors in the metric-space for correctness and stability.

DHTs typically provide a uniform metric-space. This makes it easy to map various unrelated data items (values) to the metric-space using hash functions. In particular, DHTs normally employ *consistent hash functions*, which allow sub-ranges of the hash

space to be remapped independently of the entire space, allowing the networks to adjust under failure [45]. The typical pattern to store a data object o is to hash either all of o, a shared part of o, or a descriptor such as a string, the choice being dictated by the information available to users searching for the object. Such a system of storing and finding objects is simple, but issuing complex queries is difficult using such a simple hashing technique. More advanced systems hash multiple terms, form inverted-indexes [64], and estimate document page-rank [84], but research is ongoing into efficient multi-term query techniques.

Recent work on *skip-graphs* has presented several topologies with structures that inherently support multi-dimensional metric-spaces [5]. Based on the concept of skip-lists, skip-graphs do away with hashing, organizing *content* instead of ids from a metric space unrelated to the data stored. This allows for inexpensive range-queries and nearest neighbor searches, as the neighbors of a peer are related to its stored resources rather than its id. However, unlike DHTs, implementations of skip-graphs that can handle the level of churn experienced by peer-to-peer networks deployed across the Internet have yet to appear.

2.2.2 Unstructured Peer-to-Peer

Most modern unstructured networks are organized around graph-theoretic properties, such as the clustering coefficient, graph diameter, or expected hitting time [49; 54; 74; 76]. Others are the result of ad-hoc constructions and have only come to be understood after much research [17; 67]. In both cases, the core of unstructured networking is the use of random or semi-random search, both for the placement and discovery of data. Link-structures are sometimes random, but are more often constrained by desired graph properties. As such, the term "unstructured" refers more to the properties of the search algorithms than the link-structure they run over.

Example: Random Walks and Flooding

Unstructured topologies are often used when more complex search than DHTs allow for is desired. Since complex matching routines can be used on each peer, queries in unstructured networks can contain complex semantics, for example, range queries, multi-attribute queries, raw text search, regular expression search, etc.

There are two main algorithms used to distributed search in unstructured networks: *flooding* and *random walks*. Flooding sends a search message to all neighbors of a peer, who then in turn do the same, until the network is well explored. Random walks take a more resource constrained approach, sending a search message to only one random selected neighbor of a peer, which then repeats the process. Both schemes forward messages up to a maximum distance, often referred to as the *hops to live*.

Floods and random walks can be seen as two ends of a continuous spectrum of unstructured search techniques, all of which follow the same pattern when deciding

where to send a search message. At each hop in message propagation, a peer chooses q random neighbors and forwards the message to them. Random walks typically set q equal to 1 or 2, while floods select all peers for forwarding at each hop.

Additional algorithmic tweaks are often added for efficiency. Search protocols normally assign a unique id (*uid*) to each message. The uid can then be used to check if a message has visited a given peer or crossed a particular link previously [54]. In floods, uids are often used to drop packets that revisit a peer, as sending to the peer's neighbors again will not result in greater search coverage. Random walks can use uids to avoid crossing the same link twice [17]. Schemes that make use of greedy routing paired with random walks also exist [17; 65]. These take one of three forms. Search can first be dictated by greedy routing, until a message reaches a local minimum, at which point a random-walk is used. Alternatively, a random walk is taken until routing state is found that can more efficiently forward the message [17]. Finally, some schemes allow for any combination of greedy and random forwarding [65].

The basic dynamics of search and replication are well documented [32; 54]. Previous work has shown two important results for unstructured search [54]. First, for many unstructured search techniques, a power-law graph distribution is undesirable because it leads to a high message duplication rate when using flooding. Reliance on a small subset of high resource nodes can also lead to scalability and attack vulnerability issues [87]. Second, it is understood that random walks typically provide greater efficiency in finding information than flooding [32]. In particular, k-random walk strategies, in which k random walks are started from the searching node, can tune k such that the probability of finding a particular item is maximized at a given walk length and network size [54].

Other schemes have made use of both flooding and random walks. BubbleStorm [76] uses a hybrid random-walk/flood to implement replica placement and search across arbitrary topologies. Its key contribution is to dynamically tune the radius of replica placement and search such that the "bubble" centered around any two nodes overlaps. This overlap is designed to be large enough that a search using the hybrid random-walk will find an existing document with a desired probability.

2.3 Moving Forward

In this chapter we have given a brief overview of peer-to-peer networking and described the two main categories of existing peer-to-peer networks. In the next chapter we first introduce supporting theory for a deeper understanding of random walks, and then introduce the main related work of the dissertation.

Chapter 3

The Theory and Practice of Sampling Peer-to-Peer Networks

In this chapter we develop a formal model of peer-to-peer networks, and then use it to formally define the process of peer sampling. Our treatment is similar to that developed by Stutzbach et al. [75]. We then formalize random walks using Markov-chain theory, and discuss techniques useful in the analysis of static graphs. In particular, we discuss the use of spectral decomposition and the mixing time of a graph. Using the introduced theory, we then discuss existing sampling techniques and their limitations.

3.1 Sampling Definitions

A peer-to-peer network is typically modeled as an undirected graph $G = (V, E)$, where V are the peers (vertices) in the network and edges in E represent the links realized between them. An edge between peer u and peer v is denoted $(u, v) \in E$, and since the graph is undirected $(u, v) = (v, u)$.

There are two problems with this model. First, as we discussed in Chapter 1, peer-to-peer links are often logically or physically directed. Second, peer-to-peer networks are dynamic, with peers joining and leaving, and existing peers creating and dropping links. Therefore we model an *instantaneous peer-to-peer topology* as a *directed* graph $G_t = (V_t, E_t)$ over $n = |V_t|$ vertices, where $(u, v) \in E$ is ordered by direction. By instantaneous we mean a *snapshot* of a peer-to-peer network at a time t. We will drop the subscript t when it is clear we are talking about a particular graph or the particular graph is inconsequential. The dynamics of a peer-to-peer network over time are captured by the infinite, totally ordered set $\mathcal{G} = \{G_0, G_1, \ldots\}$ of time-indexed graphs.

A *peer sample* $X = v \in V$ is a single peer from V selected with probability $\pi(v)$ from the discrete probability distribution $\pi : V \to [0, 1]$. To be clear, our sample space is the nodes themselves, $V = \{v(0), v(1), \ldots, v(n)\}$, not the data stored at the peers in V.

In order for sampling to be useful, we will likely need more than a single sample.

For example, if we want to find the mean memory usage of peers in the network, we will want to sample enough peers that our confidence is high in the result (i.e., we have enough samples to rely on the central limit theorem). An intuitive way to build a sample set is to select a set of peers uniformly at random from V, sample the value we are interested in, and then calculate appropriate statistical aggregates.

Surprisingly, this simple method can be problematic. Since a peer building a sample set only has knowledge of immediate neighbors, and does not have global knowledge of the network, each sample must be gathered by a non-local process that takes some non-zero amount of time (i.e., a random walk; see Section 3.2). In addition, due to resource constraints, it may not be possible to gather all required samples simultaneously, resulting in the need for further additional time. Taken together, these properties of peer-to-peer sampling mean that the set of peers V may change between the beginning and end of sample collection. Assuming samples are collected between time t and $t+\delta$, where a time t can be used to refer to a graph snapshot G_t, we define $V_{t,\delta}$, the set of peers which it is possible to sample:

$$V_{t,\delta} \triangleq \bigcup_{j=t}^{t+\delta} V_j, \text{ where } G_j = (V_j, E_j). \tag{3.1}$$

We refer to the set of snapshots between t and $t+\delta$ as a *sample window* with temporal length δ.

If we are only interested in sampling peer ids, this scheme works perfectly fine. However, if we desire to sample the *properties* of peers (e.g., performance metrics, peer content, etc.), simply sampling nodes from the sample window leads to problems. Imagine that two evenly sized groups of peers that together make up the membership of a peer-to-peer network. Half of them come from a set of peers with short lifetimes of length α, while the other half are long-lived peers with a near-infinite lifetime. If the sample window has length α the peers will be evenly represented, half-and-half, in the set of samples. When the window is 2α, twice as many short lived peers will be in $V_{t,2\alpha}$ then long lived peers. The larger the sampling time window (i.e., the larger the multiplier n for α), the more over-represented short-lived peers become in $V_{t,n\alpha}$ and in any uniform set of samples taken from $V_{t,n\alpha}$. If the property of interest being sampled is related to peer lifetime, it will be skewed by this over-representation of short-lived peers (or, equivalently, under-representation of long-lived peers).

Merely sampling with replacement does not fix this problem. Instead, we must differentiate the same node at different times. Formally, we consider a peer to be different in each snapshot. For a single peer v we denote $v_t \in V_t$ at time t and $v_{t+1} \in V_{t+1}$ at time $t+1$, such that $v_t \neq v_{t+1}$. Even though v is the same peer in both snapshots, it can be sampled uniquely in either. We call this *multi-sampling*.

The formalism of multi-sampling should not be confused with the reality of the situation: the id of the peer does not actually change between snapshot, it is merely allowed to be sampled in both. When using random walks to sample a network, this

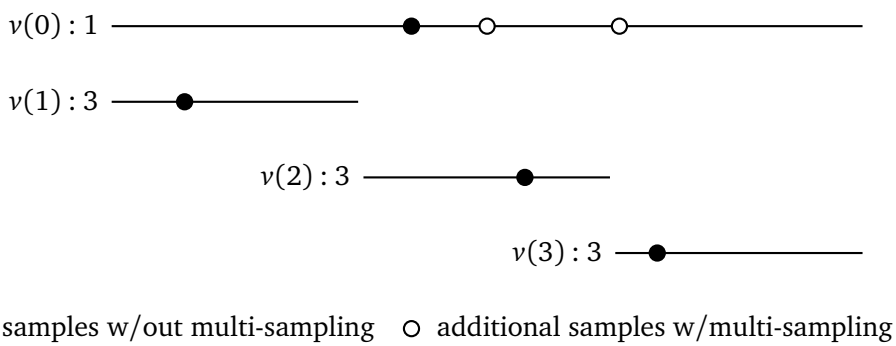


Figure 3.1. Four nodes shown across time. The value of each node is shown to the right of its name. With multi-sampling the calculated average weight is 2 for the entire window, while without it the average weight is 2.5, a biased result.

translates into accepting the result of every walk, even if two walks sample the same peer.

If we are interested in sampling peer properties, we would like to perform sampling such that aggregate metrics calculated over the sample set accurately reflect overall properties of the network. Figure 3.1 shows why multi-sampling is essential to accomplish this. The figure shows four peers, $v(0), \ldots, v(3)$, across time for a single sample window. Each peer has an associated value, $v(0) \to 1$, and $v(1), \ldots, v(3) \to 3$. At each point in time the average weight in the network is 2, as $v(0)$ is alive for the entire sample window, while the other peers are each alive for non-overlapping thirds of the sample window.

If every peer was sampled without the use of multi-sampling (the samples represented by black dots), $v(0)$ could only be sampled once, affectively under-weighting it for its actual lifetime. The empty dots would be rejected, since they would sample an already seen peer a second and third time. Similarly, $v(1), \ldots, v(3)$ are overweighted, counted as if they were alive for the entire sample window. This overweighting results in a biased average of the peer values of 2.5.

On the other hand, if multi-sampling is used, then $v(0)$ can be sampled multiple times. The inclusion of the additional samples of $v(0)$ correctly weights for the longer lifetime of the peer in the sample window, resulting in an accurate average of the peer values of 2. As the number of samples in the sample set grows, the sample average approaches the average of the instantaneous mean across all live peers in the network, assuming samples are uniformly spread over the sample window. Since at every time in the sample window of Figure 3.1 the instantaneous average is 2, the desired result for the average of a sample set is also 2.

Clearly the accurate weighting of peers improves with additional samples. However, in practice the number of samples is much less than the number of nodes, such that multi-sampling only becomes important if there is a correlation between the property

being sampled and the expected lifetime of peers. Since often such a correlation can not be ruled out a priori, we adopt multi-sampling for the remainder of the dissertation.

Using the formal definition of the sample window that supports multi-sampling, uniformly sampling across the members of $V_{t,\delta}$ will no longer over- or under-sample peer populations or properties, allowing one to correctly calculate statistical aggregates over long time windows. Using the numerical example above, when the sample window has length 2α, both short-lived and long-lived peers will now be properly represented, with each making up one-half of the peers in $V_{t,2\alpha}$. This decoupling of peer session lengths and sampling is an important property of a properly functioning peer-to-peer sampling scheme, as the time needed to generate a sample in a peer-to-peer network can be on the order of tens of seconds, and the time needed to collect a sufficiently sized sample set can be on the order of minutes.

We can now define a *sample set*, $\mathscr{X}_{t,\delta}$, a set of peer samples gathered over a contiguous subset of δ graphs in $\mathscr{G}$, in the sample window of t to $t+\delta$,

$$\mathscr{X}_{t,\delta} \triangleq \left\{ v \in V_{t,\delta} \;\middle|\; \exists i \in Q, v = X_i \right\}, \quad Q \triangleq \left\{ j \in \mathbb{N} \;\middle|\; 1 \leq j \leq q \right\} \tag{3.2}$$

where X_i is sample i out of a total of q total samples.

Both definitions of $V_{t,\delta}$, with and without multi-sampling, produce interesting sample sets. If one is interested in sampling *unique* peers, the first formulation is appropriate. However, we generally find the second formulation using multi-sampling to be more interesting, as it allows for the accurate measurement of peer properties, assuming the underlying sampling distribution is near-uniform. Up to this point we have referred to both definitions under the generic term "peer sampling," but all discussions in the remainder of this dissertation assume the use of multi-sampling.

3.2 Random Walks as a Markov Chain Process

In order to generate a peer sample, a peer needs some way to sample peers that are not in its neighbor list. One way to gather samples from a network that is well understood from a theoretical viewpoint is to use random walks. A random walk can be thought of as a message that explores the network on behalf of the originating node, sending back what it finds at the end of the walk (the sample).

A random walk is a stochastic process. From a peer v, a walk proceeds by picking a peer to transition to, from the neighbors of v. In particular, this next peer is selected from the probability distribution defined by the probabilities p_{uv} such that $(u,v) \in E$. A walk has a finite length, which we will refer to as the *total hops to live*. At any given point in a walk, the node has a remaining number of transitions to make, the *hops to live* (htl).

Clearly, each step in a random walk is determined by two factors: the peer at which the walk is currently located, and the distribution of probability across the out-edges of that peer. As such, random walks exhibit the *Markov property*, as each step only depends

on the current peer and a limited past history. Indeed, in the case of random walks, the size of past history is zero, and the location of the walk after the transition to v_{t+1}, and the previous location v_{t-1}, are independent. Formally:

$$\Pr(v_{t+1} = v \mid v_0 = a, v_1 = b, \ldots, v_t = u) = \Pr(v_{t+1} = v \mid v_t = u) \tag{3.3}$$

Since the previous and next state are independent, we can model a random walk as a discrete Markov chain with order 1. A Markov-chain of order m has the property that

$$\Pr(X_n = q_n \mid X_{n-1} = q_{n-1} \ldots X_0 = q_0) = \Pr(X_n = q_n \mid X_{n-1} = q_{n-1} \ldots X_{n-m} = q_{n-m}), \tag{3.4}$$

where X_n is the nth random variable in a sequence $X_1, \ldots, X_n$, each with the Markov property. The value q_n is taken from a finite set of possible states S, the *state space* of the Markov chain.

For random walks, the state space consists of the vertices V of a graph G. Thus we can rewrite the probability using the transition probabilities defined on the edges of G:

$$\Pr(X_n = v \mid X_{n-1} = u) = p_{uv} \text{ where } u, v \in V \tag{3.5}$$

We will simply use p_{uv} to represent this probability for the remainder of this dissertation.

We represent the transition probabilities as an $n \times n$ matrix P, where $P(u, v) = p_{uv}$. This is the *transition probability matrix*. Note that row v of the matrix represents the probabilities to transition to the neighbors of peer v, and that the column u lists the transition probabilities of the upstream neighbors of a peer u. Since $\sum_{v \in V} p_{uv} = 1$ for every vertex u and all transition probabilities are non-negative, the transition matrix is non-negative and right-stochastic.

Using P, we can find the probability to be at any vertex after a certain number of steps of the random walk. Let the row-vector x denote the probabilities of being at each state in Markov chain (i.e., each vertex in the graph). Given the current distribution x_t, we can find the probability after another step via simple vector-matrix multiplication, $x_{t+1} = x_t P$. For example, when a random walk starts at a state (vertex) v, $x_0 = [0, \ldots, 0, 1, 0, \ldots, 0]$, with the 1 in position v of x.

3.2.1 Unique Stationary Distribution

Perron-Frobenius theory states that if a non-negative stochastic Markov chain is *ergodic*, then the probability to arrive at a given node after a sufficiently long walk stabilizes, independently of the starting point, to a unique *stationary distribution* π, where a state v has a *visitation probability* of $\pi(v)$ [8; 55; 61; 34]. This property is sometimes referred to as the Fundamental Limit Theorem of Markov Chains. In this context, ergodic means the chain is both irreducible and aperiodic. For the chain to be irreducible there must exist a number q such that $P^q(u, v) > 0$ for all pairs u and v in V. Aperiodicity adds the condition that there is some $n \geq q$ such that $P^{n+d}(u, v) > 0$ for all $d \geq 0$. To put it

another way, a Markov chain is ergodic if every state is reachable from any other state, and, beyond a certain sequence length, has some probability to be reached at all greater sequence lengths.

In practice, a peer-to-peer network that is both strongly connected and aperiodic will have an ergodic Markov chain. These are reasonable assumptions. If a topology is not strongly connected then there is some set of peers for which no path exists to a subset of V. The usefulness of such a network is most likely limited. Similarly, aperiodicity is highly likely to occur naturally in a peer-to-peer topology. Making matters even easier, random walk biasing algorithms make use of self-loops, and any peer with a self-loop has an aperiodic state in the associated Markov chain. A single aperiodic state guarantees that the entire chain is aperiodic. We show this with a proof adapted from Norris (proof of Lemma 1.8.2 [61]).

Theorem 3.2.1. *If P is irreducible and has an aperiodic state i, then all other pairs of states u and v in V, $P^n(u,v) > 0$ for all sufficiently large n.*

Proof. Since i is aperiodic there exists r such that $P^r(u,i) > 0$ and an s such that $P^s(i,v) > 0$. Therefore there must exist an aperiodic path through state i such that

$$P^{r+l+s}(u,v) \geq P^r(u,i)P^l(i,i)P^s(i,v) > 0$$

for all sufficiently large x. Since such a path can be constructed for any pair of states $u, v \in V$, a value n exists that takes the maximum value of $r+l+s$ across all paths, such that $P^{n+d}(u,v) > 0$ for all pairs in V and $d \geq 0$. □

The stationary distribution π can be easily derived from the probability transition matrix P and any initial distribution of probability x. The Perron-Frobenius theorem implies that any initial probability distribution must converge to π after sufficient steps in a Markov process, such that $\pi = x_{n+1} = x_n P = xP^n$ for a sufficient n. Calculating this value using vector-matrix multiplication is slow in practice. Therefore we turn to spectral analysis (Section 3.3) to derive the stationary distribution and other useful information about the Markov chain representing a random walk.

For additional information about the Fundamental Limit Theorem of Markov Chains, Grinstead and Snell offer an excellent introduction in their *Introduction to Probability* [34], including an entire section dedicated to a proof of the theorem. Norris offers a much tighter proof in his book *Markov Chains* (Theorem 1.7.7) [61].

3.2.2 Caveats

We note that the properties discussed in this section assume a *static* topology, or, at least, a topology that changes so infrequently that the random walks used for sampling are not effected by churn. Such conditions are not likely to exist in real peer-to-peer networks. Part of the future work is to find or develop a theoretical framework that more

closely matches the underlying assumptions about peer-to-peer systems (see Chapter 6). However, as we show with the results presented in Section 4.6.5 of Chapter 4, it is possible to correct for high rates of churn such that sampling remains feasible. We currently assume that churn during walks does not decrease their independence from their starting location, although we do not have a theoretical analysis to support this.

3.3 Spectral Analysis

Spectral analysis (also called spectral theory) is the study of properties related to the spectral decomposition of a matrix. In particular, spectral analysis using eigendecomposition factorizes a matrix into a canonical form that exposes fundamental properties of the system. In this dissertation we use the eigendecomposition and decompose the $n \times n$ matrix P as,

$$P = Q\Lambda Q^{-1}, \tag{3.6}$$

where the i^{th} column of Q is the eigenvector q_i of P, and Λ is a diagonal matrix with diagonal elements corresponding to the eigenvalues λ_i of P. The eigenvalues and eigenvectors satisfy the linear equation,

$$Pq_i = \lambda_i q_i. \tag{3.7}$$

We consider the eigenvalues to be ordered from largest to smallest magnitude, $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$, where $|\lambda|$ is the modulus of an eigenvalue, with the associated eigenvectors sorted identically. Intuitively, the eigenvectors are those vectors which the matrix P only shrinks or elongates under multiplication. In particular, the magnitude of the i^{th} eigenvector is changed by a factor of λ_i. All other vectors experience both rotation and a change in magnitude under multiplication by P.

The particular class of matrices we are interested in, transition probability matrices, are left-stochastic in nature. As stated by Perron-Frobenius theory, such matrices have a stationary distribution of π. Clearly then, π must be the eigenvector associated with eigenvalue 1, such that $\pi P = 1\pi$. Note that this is a *left-eigenvector* and *-eigenvalue*, as our transitions are defined row-to-column. Alternatively, we could write $P^T\pi = 1\pi$, where P^T signifies the transpose of P.

Importantly, Perron-Frobenius theory also tells us that 1 must be the highest eigenvalue in the system. We know that the system converges, and has a stationary distribution, so 1 must both be an eigenvalue and be the highest eigenvalue, since higher eigenvalues would mean the system did not converge. Therefore, $\lambda_1 = 1$ and $\lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n| > 0$.

The implications for random-walks are two-fold. First, we can use the eigendecomposition to find the stationary distribution of a particular transition probability matrix P without the iterative calculation of $x_{t+1} = x_t P$. This is invaluable, particularly since x converges to π at a rate related to the magnitude of λ_2. Indeed, λ_2 also determines the

rate at which random walks converge to the stationary distribution. Thus we can also use λ_2 to characterize the efficiency of walks using a given transition probability matrix. We now discuss how this can be accomplished.

3.3.1 Mixing Time

The necessary length of a random walk to converge to the stationary distribution is given by the *mixing time* τ. In other words, once a random walk has reached the mixing time, the vector x representing the probability for a random walk to be in a particular state (i.e., at a particular peer) is near the stationary distribution π.

We can calculate τ through repeated matrix multiplication, however this method is inconvenient and slow. Instead, approximations for τ can be computed using results from spectral graph analysis. One bound on the mixing-rate is given by Sinclair, [21; 70]

$$\tau = O\left(\frac{\log(n)}{1-|\lambda_2|}\right).$$

This formulation is limited. It assumes that the Markov-chain associated with the graph G is *reversible*, such that if a state u can transition to v, then v can transition to u. This may not hold in the topologies we study, as we do not assume bidirectionality. The mixing time is not an absolute value, as the second component of the eigendecomposition only asymptotically goes to 0 as the number of hops goes to infinity. As such, the multiplicative constant hidden in the big-O notation is important in order to find the appropriate walk length.

We specify a *minimum precision* ϵ, defined as the maximum possible contribution from the distribution associated with the second eigenvector. This contribution decreases exponentially at the rate given by λ_2. Defining the maximum value in the normalized eigenvector associated with λ_2 as $\max(q_2)$, the contribution can never be greater than $\lambda_2 \max(q_2) \leq \lambda_2$. Assuming a worse-case scenario that a walk starts with a maximal component of q_2, at the node associated with $\max(q_2)$, then the contribution of q_2 to the random-walks probability distribution decreases at the rate λ_2^t, where t is the number of hops taken. As such, we can easily solve for the number of hops t necessary to reach a given ϵ:

$$\begin{aligned}(\lambda_2)^t &= \epsilon \\ t\log\lambda_2 &= \log\epsilon \\ t &= \log\epsilon/\log\lambda_2 \\ &= \log_{\lambda_2}\epsilon\end{aligned}$$

This, too, is only an upper bound, as it makes worse case assumptions about the starting point of a sample walk. However, as we will see in Chapter 5, it is easy to use this bound to closely estimate the required number of hops to converge to the stationary distribution for a given network topology.

3.4 Related Work

Now that we have introduced the essential elementary theory, we discuss two types of related peer sampling work. We distinguish two broad categories of techniques for peer sampling: random-walk peer sampling, and gossip-based schemes. The second of these techniques is somewhat similar to random walks, but instead uses continuous mixing processes to create local caches of samples at each peer, rather than independent walks.

3.4.1 Random-Walk Peer Sampling

We present two random-walk schemes. Although more exist [30; 86], the dynamics are similar, with the differences focused on improving the mixing time of the walks or sampling from non-uniform distributions.

Metropolis-Hastings

Metropolis-Hastings is a discrete Markov-chain random-walk biasing algorithm [13; 21; 38; 58; 75]. Originally introduced as a computationally inexpensive technique to bias random walks in early computers without the memory to store two sets of transition probabilities, Metropolis-Hastings allows the re-biasing of transition probabilities using only local topological information. To do this, it makes use of a property of reversible Markov-chains called *detailed balance* or *time-reversibility*:

$$\pi(u)p_{uv} = \pi(v)p_{vu} \tag{3.8}$$

Simply, time reversibility says that the probability to be in any given state times the probability to transition to a neighbor must be the same as the probability to be in that neighboring state times the probability to transition back.

Metropolis-Hastings makes use of time-reversibility by setting the desired sample probabilities and solving for the transition probabilities. In general, P can be transformed into a new transition matrix Q such that a target distribution μ becomes the new stationary distribution. This can be done as follows:

$$q_{uv} = \begin{cases} \min\left(\frac{\mu(v)p_{vu}}{\mu(u)p_{uv}}, 1\right) p_{uv} & \text{if } u \neq v, \\ 1 - \sum_{u \neq v} q_{uv} & \text{if } u = v. \end{cases} \tag{3.9}$$

Note that the first term in the minimum is just the ratio of the two sides of the detailed balance equation. The case where $u \neq v$ balances p_{uv} and p_{vu} by lowering the transition probability from the state with excess likelihood to transition. This excess probability is transformed into a self-loop in the second case, such that detailed balance is maintained. Intuitively, the node with less probability to be reached increases its probability by inducing a self-loop.

When the desired target distribution is uniform [21; 75], such that $\mu = [\frac{1}{n}, \ldots, \frac{1}{n}]$, Equation 3.9 simplifies to

$$q_{uv} = \begin{cases} \min\left(\frac{d(u)}{d(v)}, 1\right) p_{uv} & \text{if } u \neq v, \\ 1 - \sum_{u \neq v} q_{uv} & \text{if } u = v, \end{cases} \tag{3.10}$$

where $d(u)$ is the degree of u. Degree is used in place of p_{uv} because the initial transition probabilities are assumed to be $P(u,v) = \frac{1}{d(u)}$.

Metropolis-Hastings is advantageous in peer-to-peer networks due to its local nature. Only degree information from immediate neighbors is needed to balance the entire graph, such that no iterative process is necessary. This is clearly an advantage under churn, when a global process might be too slow. However, this advantage is offset by the fact that the network *must* be symmetric such that the underlying Markov chain is time-reversible. This means all links must be bidirectional. For the reasons stated in Chapter 1, we feel that this is an unrealistic assumption, both because of network middle-boxes, and because of the imposition of additional requirements on the underlying peer-to-peer network.

Maximum Degree and Random Weight Distribution

Maximum Degree and Random Weight Distribution are related algorithms to Metropolis-Hastings [6]. Instead of using information about the degree of immediate neighbors, peers assume a maximum allowable degree across the entire network, and then bias their edges appropriately:

$$q_{uv} = \begin{cases} 1/d_{max} & \text{if } u \neq v, \\ 1 - d_i/d_{max} & \text{if } u = v, \end{cases} \qquad \begin{cases} 1/\rho & \text{if } u \neq v, \\ 1 - d_i/\rho & \text{if } u = v, \end{cases} \tag{3.11}$$

Maximum Degree | Random Weight Distribution

where d_{max} is the largest degree of any node in the network and $\rho \geq d_{max}$. These biases result in a uniform transition probability across all edges in the graph, except for the self-loops. This results in much higher self-loops than Metropolis-Hastings, but eliminates the need to communicate degrees between neighbors.

Awan et al. introduced the Random Weight Distribution to reduce the high self-loops resulting from use of Maximum Degree [6]. They use an iterative algorithm to distribute the excess weight in the initial self-loops while assuring that the transition probabilities between neighbors remain symmetric, such that the stationary distribution remains uniform.

Like Metropolis-Hastings, Maximum Degree and Random Weight Distribution assume symmetric links. It is possible that Random Weight Distribution results in faster mixing networks than Metropolis-Hastings, although the evidence presented in this regard is not conclusive [6].

3.4.2 Gossip-based Peer Sampling

We discuss two gossip-based sampling techniques, Cyclon [82] and the Peer Sampling Service (PSS-$*$) [43]. In general, gossip-based sampling techniques have a different set of common properties than random-walk based sampling services. The dynamics of latency are typically reversed: instead of high response time and low age latency, gossip algorithms commonly exhibit high (or variable) age and low response time. Further, churn recovery is an inherently global process, as new peers (or peer properties) must be mixed into the network before being sampled with the correct frequency from the target sample distribution. Failed peers must be expunged, typically via some sort of timeout process. This is in contrast to random-walk techniques, where only link weights are updated, often just those immediately next to a joining or leaving/failed peer.

Cyclon

Cyclon [82] typifies the design of a gossip peer sampling process that uses a peer cache to store samples. At each round, a subset of cached samples are exchanged with one or more neighbors. Unlike the random-walk techniques we have discussed, Cyclon is completely autonomous from the underlying peer-to-peer network link-table. Instead, Cyclon treats the peer cache as its link table. This has the advantage that Cyclon can optimize the peer caches to assure near-uniform sample probability by maintaining a nearly equal number of references to each peer in the network. The downside is the overhead incurred from having to continuously form new network connections each shuffle, and an inability to piggy-back on host-network traffic.

The main contribution of Cyclon to the gossip literature is a simple algorithm to control the injection and deletion of peer samples in the peer caches. In each gossip round a peer injects, on average, a single new reference to itself, with an age of 0. The age is then incremented by 1 at each round of gossip. When the age of a sample reaches a pre-determined limit, the sample is removed from whatever peer cache happens to hold it. After an initial period equal in length to the maximum sample age, the expected number of new samples and of samples deleted for a given peer are equal each round, and are identical across all peers. As such, every peer has the same number of samples referring to it being gossiped at any given time, leading to equal probability of selection over the entire network.

Cyclon offers an interesting sampling interface. When Cyclon is in a converged state (i.e., enough gossip rounds have taken place to randomize each peer cache), the first sample returned from the sampling service is uniform. When Cyclon has converged, this cache contains a *set* of Q peers uniformly selected from all sets of size Q. A single peer selected from this set is uniform sampled from the membership of the entire network. However, additional peers are not. If additional sampling is done with replacement, the first peer sampled will have a very high probability to be sampled again ($1/Q$) compared to its desired frequency ($1/N$, where there are N peers in the network). On the other

hand, if the sampling is without replacement, the originally sampled peer will have *no* probability to be sampled again. In either case, additional samples are not independent or identically distributed.

For large N, this effect is minimal when using sampling without replacement. We therefore characterize Cyclon as "pseudo-uniform." However, when large numbers of samples are taken in quick succession, this effect may be undesirable. Our desire to sample peer properties across time falls into this category.

The age of samples is variable in Cyclon. Cyclon defines several different rules for returning samples from the peer cache, some of which bias toward or against old samples, and others which are indifferent to age. In this last case, the difference between samples may vary as much as the maximum lifetime of an entry in the cache. By default, response time is low, as there are samples in the cache, but, as for all sampling algorithms, response time can increase if a peer process desires that samples come from a distribution very close to uniform. This is a problem for all gossip-based sampling services, especially for sampling tasks with bursty sample frequency, as the probability of sampling peers from the local peer cache quickly deviates from uniform, particularly when sampling with replacement. Multiple gossip rounds must be run for samples in the cache to approach uniform sample probabilities. This increases response time for sampling tasks with low error tolerance, or in the presence of bursts of sample requests.

Churn recovery in Cyclon is global. When a peer joins a network, it takes a number of gossip rounds at least equal to the lifetime of a sample in the cache to integrate such that it will be sampled with the proper probability. Similarly, upon failure, all references to the failed peer must timeout in the peer caches before the peer will no longer be erroneously sampled.

The Peer Sampling Service

The Peer Sampling Service [43], or PSS-$*$, encompasses a set of gossip-based sampling schemes. The algorithm are similar to Cyclon: a separate overlay topology is formed, and each node caches peer references. Then, during gossip, each peer exchanges a subset of the cached references.

PSS-$*$ offers a wide range of sample algorithms. In particular, PSS-$*$ defines three main parameters: *peer selection* describes the process of selecting the peer to gossip with during the current gossip round; *view propagation* defines how information is gossiped; and *view selection* determines which peers are sent and which will be kept during gossip exchange.

Of the possible combinations explored in the parameter space, several settings are essential in PSS-$*$ to assure sampling is effective. The first of these is using *push-pull* view propagation, such that peer samples are exchanged in both directions during a gossip exchange. Pure push or pull exchange leads to poor sample propagation, and worse, topology disconnection. Second, view selection appears important to handle churn successfully. A scheme termed *healer* by PSS-$*$ keeps the freshest entries after

an exchange, intuitively keeping those that are most likely to be alive. The more fresh entries that are kept, the better a PSS-$*$ scheme handles churn.

As with Cyclon, PSS-$*$ does not offer uniformity if multiple samples are requested between gossip rounds. Instead, the authors argue that uniformity is achieved by sampling *across time*. This makes the scheme inappropriate when many uniform samples are needed at once. From a global perspective, PSS-$*$ protocols do not maintain a uniform number of peer samples across the network for each peer. If a sampling task is distributed across multiple peers, and samples are taken during the same rounds, some nodes will be more likely to be sampled than others. This makes PSS-$*$ inappropriate for distributed sampling tasks which desire uniformity in their samples.

3.5 Moving Forward

This concludes the introductory material in this dissertation. We now discuss the Doubly Stochastic Convergence algorithm, which relaxes the typical assumptions of random walk biasing algorithms, such that it only requires an ergodic topology.

Chapter 4

General Uniform Sampling

This chapter is an expanded version of a paper published at Euro-Par 2009 entitled Uniform Sampling for Directed P2P Networks *[37] and a technical report of the same name [36]. While the core algorithm remains the same, additional work on feedback has significantly improved the performance since the previous publications, and our experimentation has expanded to larger topologies.*

Selecting a random peer with uniform probability across a peer-to-peer network is a fundamental function for unstructured search, data replication, and monitoring algorithms. Such uniform sampling is supported by several techniques. However, current techniques suffer from sample bias and limited applicability.

In this chapter, we present a sampling algorithm that achieves a desired uniformity while making essentially no assumptions about the underlying peer-to-peer topology. This algorithm, called *Doubly Stochastic Convergence* (DSC), iteratively adjusts the probabilities of crossing each link in the network during a random walk, such that the resulting transition matrix is doubly stochastic. DSC is fully decentralized and is designed to work on both directed and undirected topologies, making it suitable for virtually any peer-to-peer network. Our simulations show that DSC converges quickly on a wide variety of topologies, and that the random walks needed for sampling are short for most topologies. In simulation studies with FreePastry, we show that DSC is resilient to high levels of churn, while incurring a minimal sample bias.

4.1 Introduction

Our overarching goal is to create a generic "plug-and-play" framework for sampling properties (bandwidth, load, etc.) and data of interconnected peers in an arbitrary peer-to-peer network. Ideally, the resulting measurements should give an unbiased view of the current distribution of the properties over the network, which is useful for immediate parameter tuning as well as arriving at a correct understanding of network dynamics over multiple sample runs.

In this chapter we focus on a fundamental component of such a framework. Specifically, we implement a *uniform sampling function* that returns a peer chosen uniformly at random amongst all peers in a network. This function is applicable to networks of any topology, and requires no global knowledge, and supports multi-sampling as described in Section 3.1. In addition to serving as a basis for monitoring, uniform random sampling is a useful building block in distributed systems in general, where it is used to support search, maintenance, replication, and load-balancing [32; 74].

As discussed in Chapter 3, existing techniques for uniform sampling are based on biased random walks or gossip. In the first case, a node is selected at the end of a sufficiently long random walk in which the probabilities of following each link are adjusted to obtain a uniform visitation distribution across the network. Existing algorithms, such as Metropolis-Hastings and maximum-degree, use local properties of nodes to compute their visitation probabilities, which they then use to bias the transition probabilities [6; 21; 74]. The locality of this information makes these algorithms practical and efficient. However, these algorithms must assume that all links are bidirectional, and therefore may not be universally applicable.

Another approach to sampling peers is gossip based peer sampling. In gossip based sampling, nodes maintain a pool of addresses of other peers which is continuously exchanged and shuffled with other peers through a gossip protocol [43; 82]. Gossip sampling is tunable and efficient in terms of traffic, and is effective for applications such as search, load-balancing, and topology construction. However, it is less appropriate for statistical sampling, as such sampling can be expected to have bursty high demand. Gossip handles such demand by significantly increasing the rate of gossip (and the associated overhead), which is necessary to provide independent and identically distributed samples. Yet even when no samples are being requested the higher rate must be maintained, else peer caches will be out of date during the next burst.

Our algorithm falls withing the general category of random-walk sampling. It is distinguished from other random-walk sampling algorithms by avoiding reliance on the ability to calculate the visitation probability at each node. Instead, Doubly Stochastic Convergence adjusts the transition probabilities iteratively, converging to a state in which the sum of transition probabilities into each node equals 1. The resulting transition matrix is said to be *doubly stochastic*, and is guaranteed to induce uniform visitation probabilities.

The contribution of Doubly Stochastic Convergence is to remove the assumption of bidirectional links in the network while maintaining desirable statistical properties (i.e., uniformity of the sample distribution) for the sample walks. Furthermore, the algorithm we propose is fully distributed and uses only localized information. These features make it applicable to virtually any peer-to-peer system, as well as to a wide range of applications. In practice, the algorithm performs well on both static and dynamic topologies, keeping the ratio between the maximum and minimum sample probability below 1.2 for realistic churn conditions. Further, our algorithm generates link-biases that keep the

expected sample walk length reasonably short, between 20 and 50 hops for 10000-node static (no churn) topologies of various kinds, and around 23 hops for 1000-node Pastry networks under churn.

The rest of this chapter proceeds as follows. In Section 4.2 we show that doubly stochastic transition matrices offer a theoretically sound platform for uniform sampling. Section 4.3 presents a basic version of our Doubly Stochastic Convergence algorithm. We give a proof of the convergence of this basic version of DSC, and then present a more advanced variant that deals with failure (Section 4.4) and reduces the length of the random walks (Section 4.5). Section 4.6 evaluates DSC in simulations on static topologies, and within a concrete system under churn. Finally, Section 4.7 concludes with a discussion of future work.

4.2 Finding a Uniform Stationary Distribution

Since we want to sample peers uniformly starting from any node, our goal is to ensure that the stationary distribution π of the Markov chain associated with the network to be sampled is *uniform*. The theory of Markov chains provides a sufficient condition that leads to such a stationary distribution. Specifically, every ergodic Markov chain with a *doubly-stochastic* transition matrix has a uniform stationary distribution. A matrix is doubly stochastic when every row and every column sums to 1. In other words, if the sum of the transition probabilities of all the *incoming* edges of every node is 1 (the sum of the transition probabilities of all the *outgoing* edges is 1 by definition), then the transition matrix is doubly-stochastic, and the stationary distribution is uniform. Defining $\mathbf{1}$ as the row vector $[1,\ldots,1]$ of dimension n:

Theorem 4.2.1. *Let P be an ergodic doubly stochastic transition matrix, and $x_0,\ldots,x_t,\ldots$ be a sequence where $x_{t+1} = x_t P$. Then $\lim_{t\to\infty} x_{t+1} = \frac{1}{n}\mathbf{1}$ and $\pi = x_{t+1}$.*

Proof. Assume $x_t = \frac{1}{n}\mathbf{1}$ is a probability distribution across the vertices of G. Since $x_{t+1} = x_t P$, the following holds for all $v \in V$:

$$x_{t+1}(v) = \sum_{u\in V} x_t(u)p_{uv} = \frac{1}{n}\sum_{u\in V} p_{uv} = \frac{1}{n}$$

The last reduction comes from the fact that each column sums to 1, and, therefore, the value at each index of x_{t+1} equals $1/n$ and the distribution is stationary. Since the Fundamental Limit Theorem of Markov Chains states there is a unique stationary distribution for an irreducible, aperiodic Markov chain (see Section 3.2.1), x_{t+1} must be the sole stationary distribution π for P. Therefore, for all doubly stochastic matrices, $\pi = \frac{1}{n}\mathbf{1}$. □

As Theorem 4.2.1 holds, the goal of DSC is to bias the transition probability of edges in an ergodic graph G such that the transition matrix P representing the graph

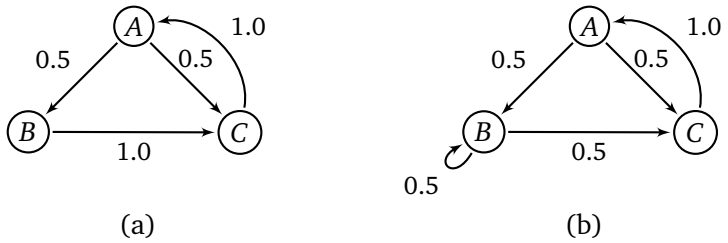


Figure 4.1. An ergodic graph. (a) Labeled with initial transition probabilities. (b) Balanced using a self-loop.

G is doubly stochastic. This will ensure that samples generated by appropriate length random walks will be uniform across all peers in the peer-to-peer network.

4.3 The DSC Algorithm

One can uniformly sample a network by first assigning transition probabilities to edges so that the stationary distribution is uniform, and then performing random walks of sufficient lengths, bounded by the mixing-time. In this section we discuss an algorithm that solves the first of these requirements. After presenting the proposed algorithm, we first prove convergence, and then discuss the results and various improvements.

4.3.1 Basic DSC

As stated in Chapter 3, Section 3.2, the transition probability matrix P is stochastic by definition, as the outgoing edges of every node have a total probability of 1. Our task is to assign those probabilities so that, for each node, the probabilities of the incoming edges also add up to 1, thereby making P doubly stochastic. We refer to this process as *balancing*.

In a fully decentralized balancing algorithm, a node v cannot directly control the probabilities assigned to its incoming edges, which are controlled by v's predecessors. At any given time, v may only know about a subset (possibly none) of its predecessors. Even if v could control the probability of some of its incoming edges, changing one of these probabilities would affect the balance of probabilities for other nodes. To gain some degrees of freedom for the purpose of balancing, we introduce an extra edge, the *self-loop*, linking each node back to itself. The self-loop is both an incoming and outgoing edge, and therefore can be used by a node to make up deficits in its own incoming probability.

Figure 4.1 exemplifies the difficulties of balancing, and the use of the self-loop. Neither node B or C can be balanced without removing edge (A, C). B has a deficit of incoming probability, yet A cannot rebalance its out-probability to satisfy B without

removing all probability from (A, C). Conversely, C has too much incoming probability, and B can not redirect the excess probability elsewhere. Setting $p_{AC} = 0$ balances all in- and out-probabilities, but renders the graph periodic, and hence no longer ergodic. The solution, shown in Figure 4.1(b), is to use the self-loop (B, B) to increase the in-probability of B to 1, also reducing the in-probability of C to 1, balancing P and keeping G ergodic. This leads directly to the core intuition of DSC: *increasing* the self-loop for nodes with in-probability deficits, and therefore reducing the probability across their outgoing edges, *decreases* the excess in-probability of other nodes.

It will be useful to define two related sets for each vertex v.

$$N^-(v) \triangleq \{u \in V \mid (u, v) \in E\} \qquad \text{(predecessors)}$$
$$N^+(v) \triangleq \{w \in V \mid (v, w) \in E\} \qquad \text{(successors)}$$

$N^-(v)$ holds the *predecessors* of v, while $N^+(v)$ holds the *successors* of v. Note that these two sets are not necessarily disjoint. Indeed, for any peer-to-peer topology with bidirectional links, $N^+(v) \cap N^-(v) \neq \emptyset$, as both (u, v) and (v, u) exist. If the self-loop (v, v) is present for vertex v, $N^+(v)$ and $N^-(v)$ are also not disjoint. By definition, the sum of probabilities across the out-edges $N^+(v)$ is 1, such that it forms a valid probability distribution.

We define the sum of in- and out-probability at a vertex v as

$$\mathrm{In}(v) \triangleq \sum_{u \in N^-(v)} p_{uv}; \quad \mathrm{Out}(v) \triangleq \sum_{u \in N^+(v)} p_{vu}.$$

Both $\mathrm{In}(v)$ and $\mathrm{Out}(v)$ must sum to 1 in order for P to be doubly stochastic. Since both $N^+(v)$ and $N^-(v)$ contain (v, v), when the self-loop increases or decreases a peer v must decrease or increase, respectively, the sum of probability across both $N^-(v)$ and $N^+(v)$.

At any given time, a peer is in one of three states, which we represent with membership in an associated set:

$$V^+ \triangleq \{v \in V \mid \mathrm{In}(v) > 1\} \qquad \text{(surplus in-probability)}$$
$$V^= \triangleq \{v \in V \mid \mathrm{In}(v) = 1\} \qquad \text{(balanced in-probability)}$$
$$V^- \triangleq \{v \in V \mid \mathrm{In}(v) < 1\} \qquad \text{(deficit in-probability)}$$

Either a peer has an in-probability surplus, so it is in V^+, it has an in-probability deficit, so it is in V^-, or it is balanced, and in $V^=$.

The total sum of surplus in-probability of all peers in V^+ equals the total deficit of in-probability of all peers in V^-. This stems from the fact that $\forall v \in V$, $\mathrm{Out}(v) = 1$, and therefore the mean of $\mathrm{In}(v)$ must equal 1. Any in-probability over 1 at a particular vertex v must result in a deficit at some other vertex u, and vice-versa. Therefore, if we can move vertices from V^- to $V^=$, we will also force vertices from V^+ into $V^=$, leading to $V^=$ containing all the peers in V, and to P becoming doubly stochastic.

Promoting vertices from V^- to $V^=$ is simple: increase the self-loop p_{vv} by the amount $1 - \text{In}(v)$, bringing $\text{In}(v) = 1$. This is the strategy of the basic version of DSC, with each node $v \in V$ executing the following two steps:

1. Every q seconds, v updates its self-loop p_{vv}. When $\text{In}(v) < 1$, the self-loop is increased by $1 - \text{In}(v)$. If $\text{In}(v) \geq 1$, no action is taken.

2. Every q/ρ seconds, v sends updates to all $u \in N^+(v) \setminus v$, notifying them of their current transition probability, p_{vu}. Successor nodes store this value, using it to calculate $\text{In}(v)$ in step 1.

In this basic version of DSC out-going probabilities are kept uniform and p_{vu} always equals $(1 - p_{vv})/|N^+(v)|$.

When step 1 is executed we say there has been an *advancement*. The time q should be selected with bandwidth, network latency, and churn in mind (see Section 4.6). In particular, a short q leads to faster convergence, but also increases the chance to miss updates when network latency is high. The frequency of updates, ρ, can be set higher when packet loss or latency is a problem, or to 1 if on-time delivery is assured.

Step 2 of DSC sets and sends evenly balanced probabilities. In most situations this turns out to be a non-optimal balance of probability in terms of the length of the random walks. In Section 4.5 we discuss a variant of the algorithm that uses feedback and generally converges faster and exhibits shorter walk lengths. However, in situations where receiving feedback from nodes in $N^+(v)$ is not possible, or it is inconvenient to do so, the basic version of DSC still converges relatively quickly in practice.

4.3.2 Convergence of Basic DSC

We now prove that basic DSC changes the transition probabilities in any network topology such that they converge to a uniform stationary distribution. DSC accomplishes this by iteratively reducing the distance between P and a doubly-stochastic matrix.

Theorem 4.3.1. *If G is an ergodic graph, DSC updates P iteratively in such a way that, in the limit, P converges toward doubly stochastic.*

Let $D(P)$ measure a distance from P to a doubly stochastic matrix as follows:

$$D(P) \triangleq \sum_{v \in V} d(v), \text{ where } d(v) = |1 - \text{In}(v)|$$

We show that, after i iterations of the DSC algorithm, $D(P)$ must decrease by a factor of $(1 - \epsilon)^i$ where $\epsilon > 0$ and ϵ does not depend on i. Therefore, $\lim_{i \to \infty} D(P_i) = 0$ and P becomes doubly stochastic.

Proof. Let $D(P_0) > 0$ be the initial distance, and therefore $|V^+| \geq 1 \wedge |V^-| \geq 1$. Let $u \in V^-$ be the node with the maximal in-probability deficit $d(u) = 1 - \text{In}(u)$, and consider the

execution of DSC on u. The effect of u updating its self-loop is to move the in-probability deficit of u onto the in-probabilty of peers in $N^+(u)$. A successor node $q \in N^+(u)$ can be in one of the following states when this update takes place:

- $q \in V^=$ or $q \in V^-$: in these cases, q has to completely relay its share of the deficit of u (along with its own) to its successors. There is no net reduction in $D(P)$.
- $q \in V^+$: in this case some or all of the deficit passed from u to q will be absorbed by the in-probability surplus at q, with a corresponding net reduction in $D(P)$.

As DSC advancements continue, and the deficit of u is relayed two and more hops away from u, more of the deficit may be absorbed by other nodes in V^+. However, in order to find a minimum reduction in deficit (and therefore distance) in a bounded amount of steps, we focus on the node $w \in V^+$ with the maximal in-probability surplus $s(w)$, and determine the minimum amount of $d(u)$ that gets absorbed by $s(w)$, and in how many steps. To do that, we focus on the shortest path from u to w.

Since G is strongly connected, the length of the shortest path from u to w is at most $n-1$, where $n = |V|$. At each hop along that path, the deficit is evenly divided among all successors, and is therefore progressively reduced. Since at each hop there are at most $n-2$ successors, the deficit that reaches w must be at least $d(u)/(n-2)^{n-1}$, which can be absorbed completely or up to $s(w)$. Therefore, after $n-1$ iterations, the overall distance $D(P)$ is reduced by at least $\min(d(u)/(n-2)^{n-1}, s(w))$.

Since $d(u)$ is the maximal deficit, and since there are at most $n-1$ nodes in V^-, it must be that $d(u) \geq D(P_0)\frac{1}{2(n-1)}$. For the same reason, $s(w) \geq D(P_0)\frac{1}{2(n-1)}$. This gives us a lower bound for the reduction of $D(P)$ after $n-1$ iterations, $D(P_{n-1}) \leq D(P_0)(1-\epsilon)$ where

$$\epsilon = \min\left(\frac{1}{2(n-1)(n-2)^{n-1}}, \frac{1}{2(n-1)}\right) = \frac{1}{2(n-1)(n-2)^{n-1}}.$$

This means that $D(P_i)$ decreases after a fixed number of steps $(n-1)$ by at least a factor $1-\epsilon$ that does not depend on time. Therefore $D(P_i)$ converges exponentially to 0. □

The main intuition behind the proof is that the ergodicity of the graph assures that any reduction of out-probability at a node v in V^- (caused by the node increasing its self-loop) will eventually reach at least one node u in V^+, where it will be at least partially absorbed by u's surplus, leading to a global reduction of in-probability deficit. In fact, *any* reduction of out-probability will be completely absorbed by some set of nodes in V^+, as the balance between the sum of in- and out-probabilities is invariant. Neither the propagation time (number of advancements), nor the factor of in-probability absorbed by u, depend on time. As a result, the global in-probability deficit exponentially converges to zero, leading to a doubly-stochastic transition matrix.

The bound given in proof to Theorem 4.3.1 is extremely loose, and convergence is much faster in practice. Further, we note that the ordering of node advancements can

play a role in determining the rate of convergence. For example, in Figure 4.1, updating A and C first is clearly not optimal. We have observed that a different advancement ordering may change the rate of convergence by a factor of two for some topologies.

While this basic version of the algorithm works well, it has several undesirable properties. We work on fixing these by introducing relaxation and feedback.

4.4 Churn and Relaxation

Churn events (i.e., nodes joining or leaving the network) have the effect of increasing self-loops. A node u joining the network may move its new successors from $V^=$ or V^- into V^+. The successors of u may find themselves in the peculiar state of having a surplus of in-probability while having a self-loop greater than 0, a situation that cannot occur in basic DSC without churn.

Nodes leaving the network have a similar effect. Predecessors of a leaving node u will increase their probabilities across remaining out-edges, while successors will have to increase their self-loop to make up for lost in-probability. Just as when a node joins the network, when the node u leave, it can force some $v \in N^+(q)$, where $q \in N^-(u)$, into V^+ while having a non-zero self-loop.

Taken together, these two phenomena lead to ever increasing self-loops, which are problematic as they tend to increase the mixing time. In other words, if we intend to keep a reasonable mixing time in a network with churn, we cannot rely on an algorithm that only increases self-loops. We therefore allow DSC to also lower self-loops. In particular, if a node u is in V^+, and has a self-loop $p_{uu} > 0$, u decreases its self loop by $\min(p_{uu}, \text{In}(u) - 1)$, setting it to zero if possible, or as close to zero if not. In the same operation, u must also raise the transition probabilities to its successors accordingly. In turn, this can cause successors $q \in N^+(u)$ to also lower their self-loop, and so on. However, this effect is dampened each successive step away from u, and is unlikely to propagate to the entire network, as small increases in probability are absorbed by nodes in V^- or $V^=$ with non-zero self-loops. We call this process *relaxation*.

Relaxation is essential to the function of DSC in a network experiencing churn. Without it, self-loops will approach 1, and the probability to transition to other nodes will approach 0, leading sample walk lengths to approach infinity. We note that relaxation does not require bidirectional links, allowing it to handle churn even in the absence of bidirectional communication. Additional details of relaxation are provided in the context of our experimental analysis in Section 4.6.5.

4.5 Bidirectional Links and Feedback

The basic version of DSC discussed above in Section 4.3.1 does not use information about the state of successor nodes, as without bidirectional links we cannot gather

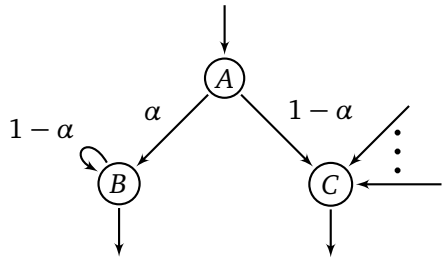


Figure 4.2. Nodes B and C demonstrate the need for feedback.

such information via local communication. However, not using such information can slow convergence and, more seriously, is likely to increase the mixing-time due to high self-loops probabilities. When bidirectional links are present, DSC can use *feedback* to improve the quality of the out-edge probability weighting.

4.5.1 Feedback and Self-loops

Our base assumption when designing the feedback mechanism in DSC is that self-loops with high transition probability increase the necessary length of sample walks. Feedback can help reduce self-loops by indicating to predecessors that a peer needs more or less incoming probability.

An example of probability imbalance where feedback can help reduce self-loops is given in Figure 4.2. The digram shows two nodes, B and C, downstream from A. Node C has many incoming links, including the link from A, and is in V^+ due to excess incoming probability. Conversely, B has a deficit of incoming probability, is in V^-, and must assign itself a self-loop. Assuming that A does not have any self-loop transition probability, it can assign a transition probability α to one peer, B in this example, and then must assign $1-\alpha$ to the other peer, C. Since B only has a single incoming link, DSC must set its self-loop such that B has a total incoming probability of 1. Since $p_{AB} = \alpha$, DSC must therefore set the self-loop of B to be $1-\alpha$.

By sending a message to A, node B could signal that it needs additional probability, such that its self-loop can be decreased. This is the strategy we adopt. In addition to storing the transition probability sent by a predecessor v, a node u also replies with an acknowledgment containing feedback information that allows v to intelligently rebalance probability across links in $N^+(v)$. The process of sending acknowledgements is termed *feedback*, while the balancing process is called *normalization*, so-called because it must result in a set of out-probabilities that sum to 1 and a self-loop that sets $\text{In}(u) = 1$ if possible. Normalization encompasses the act of advancement discussed in Section 4.3.1, updating the self-loop and performing relaxation if necessary. We discuss it in greater detail starting in Section 4.5.4.

In the example of Figure 4.2, B indicates to A that it would prefer additional transition probability across p_{AB}. Similarly, node C can indicate that it has too much incoming

probability. This allows A to shift probability from p_{AC} to p_{AB}, raising α and lowering the self-loop at B. At the same time, the total incoming probability at C is lowered, bringing it closer to $V^{=}$. This second effect bypasses the normal convergence of DSC by lowering the incoming probability at C without the need to wait for other nodes in the network to increase their self-loops (lowering their other out-probabilities). This has the effect of speeding up convergence and decreasing the total sum of self-loops across the network.

We now discuss two possible implementations of feedback. The first uses feedback to asymptotically approach an optimal distribution of probability, while the second uses feedback to, in the optimal situation, converge immediately.

4.5.2 Asymptotic Feedback

We first discuss *asymptotic feedback*, which was the original scheme implemented in DSC. Although it does not converge to an optimal balance of in-probability in the best-case, we will later see in the analysis that its behavior is often superior to the optimal feedback scheme.

In asymptotic feedback a node u sends the *feedback value* $f_{vu} = p_{vu}(1/\text{In}(u) - 1)$ to all predecessors $v \in N^-(u)$. This value is the absolute change each upstream link would need to apply to bring u closer to being balanced, and such that the new value of $\text{In}(u)$ would be closer or equal to 1 after adjustment of the self-loop. This can be directly shown. First we find the new value of incoming probability, assuming all predecessors change their probabilities as requested:

$$\begin{aligned}
\overbrace{\left(\text{In}(u)\right)}^{\text{Cur. value}} + \overbrace{\sum_{v \in N^-(u)\setminus u} p_{vu}\left(\frac{1}{\text{In}(u)} - 1\right)}^{\text{Requested difference}} &= p_{uu} + \sum_{v \in N^-(u)\setminus u} p_{vu} + \sum_{v \in N^-(u)\setminus u} p_{vu}\left(\frac{1}{\text{In}(u)} - 1\right) \\
&= p_{uu} + \sum_{v \in N^-(u)\setminus u} p_{vu}\left(1 + \frac{1}{\text{In}(u)} - 1\right) \\
&= p_{uu} + \frac{1}{\text{In}(u)} \sum_{v \in N^-(u)\setminus u} p_{vu} \\
&= p_{uu} + \frac{\text{In}(u) - p_{uu}}{\text{In}(u)}
\end{aligned}$$

We now show that in each case, when $\text{In}(u) < 1$, $\text{In}(u) > 1$, or $\text{In}(u) = 1$, that the new sum of incoming probability is closer or equal to 1. In the case $\text{In}(u) > 1$, simple observation shows

$$\text{In}(u) = p_{uu} + (\text{In}(u) - p_{uu}) > p_{uu} + \frac{\text{In}(u) - p_{uu}}{\text{In}(u)} \geq \frac{\text{In}(u)}{\text{In}(u)} = 1, \tag{4.1}$$

and in the case where $\text{In}(u) < 1$,

$$\text{In}(u) = p_{uu} + (\text{In}(u) - p_{uu}) < p_{uu} + \frac{\text{In}(u) - p_{uu}}{\text{In}(u)} \leq p_{uu} + \frac{\text{In}(u)}{\text{In}(u)} = p_{uu} + 1. \tag{4.2}$$

In either case, p_{uu} may be 0. In the case $\text{In}(u) < 1$ (Equation 4.2), the new sum may be greater than 1 by some fraction of p_{uu}, since the self-loop is not adjusted by the feedback process. This situation is dealt with via relaxation of the self-loop, setting it to the minimum necessary value such that the node stays in $V^{=}$, as described above in Section 4.4. It should be clear that when $\text{In}(u)$ equals 1, feedback is sent that requests no change be made, and $\text{In}(u)$ remains the same in the ideal case.

Importantly, the feedback value is calculated after peer u has updated its recorded p_{vu} values for various predecessor nodes, but *prior* to updating the self-loop, such that sum $\text{In}(u)$ will not sum to 1 unless the node happens to be balanced. (More details about the order of operation are given in Section 4.5.5.)

4.5.3 Optimal Feedback

A better option in some situation may be to send $p_{vu}(1/\sum_{v\in N^-(u)\setminus u} p_{vu} - 1)$ as the feedback value, which directly sets the new value of $\text{In}(u)$ in the ideal case. We call this *optimal feedback*. When we refer to the *ideal case* we mean that all feedback messages received by the predecessors of u allow for the exact changes that u desires. In practice, this almost never happens in large topologies, which means that optimal feedback value does not always perform as well as asymptotic feedback in practice.

It should be clear that the optimal feedback value converges immediately (after relaxation) in the ideal case. Using the notation N^{-u} to stand for $N^-(u) \setminus u$:

$$\begin{aligned}
\text{In}(u) + \sum_{v\in N^{-u}} p_{vu}\left(\frac{1}{\sum_{w\in N^{-u}} p_{wu}} - 1\right) &= p_{uu} + \sum_{v\in N^{-u}} p_{vu}\left(1 + \frac{1}{\sum_{w\in N^{-u}} p_{wu}} - 1\right) \\
&= p_{uu} + \frac{1}{\sum_{w\in N^{-u}} p_{wu}} \sum_{v\in N^{-u}} p_{uv} \\
&= p_{uu} + 1
\end{aligned}$$

As in asymptotic feedback, p_{uu} may need to be adjusted by relaxation after predecessors change their transition probabilities. Unlike asymptotic feedback, after relaxation the remaining in-probability is always 1.

4.5.4 Normalization

Once feedback is received, it must be applied such that the resulting out-probabilities are a proper probability distribution. It is unlikely that a node will ever find itself in the ideal case. Instead, the requests will likely be unequal, and a compromise allocation will have to be reached by DSC. During our experimentation with DSC, we tested two different forms of normalization, *standard normalization* and *differential normalization*. We now describe each in turn.

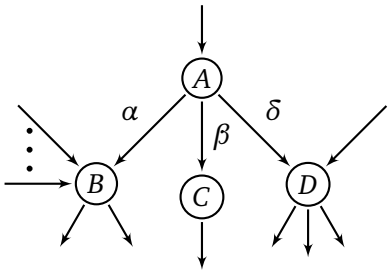


Figure 4.3. Standard normalization can behave poorly when feedback is unbalanced.

Standard Normalization

Standard normalization takes a very basic approach, first applying each feedback request, and second summing the resulting values and then normalizing each value by the sum and the self-loop. Formally,

$$\forall u \in N^+(v) : p_{vu} \leftarrow (1 - p_{vv}) \frac{p_{vu} + f_{vu}}{1 + \sum_{u \in N^+(v)} f_{vu}}, \tag{4.3}$$

where f_{vu} is the feedback value, and $\leftarrow$ indicates assignment.

In order to analyze standard normalization, we split the feedback values into two sets: those requesting less transition probability at node v, $F^+(v)$; and those requesting additional transition probability, $F^-(v)$, where $\mathrm{PF}(v)$ and $\mathrm{NF}(v)$ are the respective sums of each set at node v. We can now rewrite the right-hand side of Equation 4.3 as

$$(1 - p_{vv}) \frac{p_{vu} + f_{vu}}{1 + \mathrm{PF}(v) + \mathrm{NF}(v)}. \tag{4.4}$$

Clearly, if $\mathrm{NF}(v) + \mathrm{PF}(v) = 0$, we are in the ideal situation and each successor receives the probability allocation they desire. However, if either $|\mathrm{PF}(v)| > \mathrm{NF}(v)$ or $|\mathrm{PF}(v)| < \mathrm{NF}(v)$, the successors of v will see larger or smaller changes in their incoming probability than requested.

The graph in Figure 4.3 makes this clear. Assume A has no self-loop, and feedback values $f_{AC} = \beta$ and $f_{AD} = \delta$ are in $F^-(A)$, and $f_{AB} = \alpha$ is in $F^+(A)$. Then if $\beta + \delta$ is greater than $|\alpha|$, $1 + \beta + \delta + \alpha$ will be greater than 1, and p_{AB} will be assigned a lesser value than desired,

$$\frac{p_{AB} + \alpha}{1 + \beta + \delta + \alpha} < p_{AB} + \alpha, \tag{4.5}$$

and p_{AC} and p_{AD} will also be assigned lesser values than desired,

$$\frac{p_{AC} + \beta}{1 + \beta + \delta + \alpha} < p_{AC} + \beta \quad \text{and} \quad \frac{p_{AD} + \delta}{1 + \beta + \delta + \alpha} < p_{AD} + \delta. \tag{4.6}$$

The reverse situation holds as well, and when $|\alpha| > \beta + \delta$, $p_{AB} + \alpha$ is larger than desired, as $1 + \beta + \delta + \alpha$ is then less than 1:

$$\frac{p_{AB} + \alpha}{1 + \beta + \delta + \alpha} > p_{AB} + \alpha. \tag{4.7}$$

The same is true for $p_{AC} + \beta$ and $p_{AD} + \delta$, which will also be larger than desired.

This skewing of probability when $F^+(v)$ and $F^-(v)$ are unbalanced can have detrimental effects on the speed of convergence of DSC. In particular, a successor node may find itself constantly switching between asking for more and less probability as it oscillates from V^+ and V^-. We term this detrimental phenomenon *probability oscillation*. In order to combat such oscillation, we introduce a simple exponential dampening on feedback, and use the dampened value $\overline{f_{vu}}$ instead of the raw feedback f_{vu}:

$$\overline{f_{vu}} \leftarrow \alpha f_{vu} + (1 - \alpha)\overline{f_{vu}}. \tag{4.8}$$

This dampened feedback smooths out oscillations in the feedback requests, lowering the effective feedback value. In practice, we find that dampening can increase the rate of convergence on static topologies by a factor of 3 or greater, although for many topologies it has little effect. The effect of dampening on dynamic topologies with churn is less pronounced.

Differential Normalization

Differential normalization sums both the requests for more probability and less probability, and fulfills the requests of the lower magnitude sum. This design was in response to the probability oscillation experienced with standard normalization. Differential balance also takes into account changes in the value of the self-loop, allowing it to make maximal use of available probability.

Like standard normalization, a node v using differential normalization first sorts received feedback into two sets, $F^+(v)$ and $F^-(v)$, with $|\text{PF}(v)|$ and $|\text{NF}(v)|$ representing the sum of absolute values of the elements of the sets. Unlike standard normalization, any difference in the value of the self-loop since the previous run of normalization is also added to the appropriate sum as well. This allows differential normalization to make use of reductions of the self-loop as unwanted probability (i.e., as if the self-loop was in $F^+(v)$), and increases as desired probability (as if the self-loop was in $F^-(v)$).

If the sum of either $\text{PF}(v)$ of $\text{NF}(v)$ is 0, no action is taken. Else, the process continues depending upon whether $|\text{PF}(v)|$ or $\text{NF}(v)$ is larger:

- $|\text{PF}(v)| > \text{NF}(v)$: For each feedback value f_{vu} in $F^+(u)$, the new value of p_{vu} is calculated as $p_{vu} \leftarrow p_{vu} + \frac{\text{NF}(v)}{|\text{PF}(v)|} f_{vu}$. For each feedback value f_{vu} in $F^-(u)$, the new value of p_{vu} is calculated as $p_{vu} \leftarrow p_{vu} + f_{vu}$.

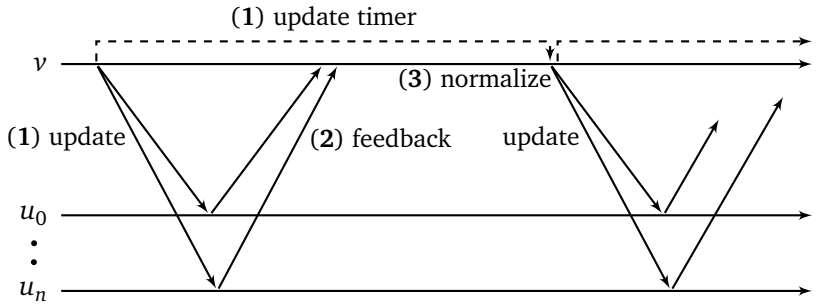


Figure 4.4. Ideal ordering of operations in DSC.

- $|\mathrm{PF}(v)| < \mathrm{NF}(v)$: For each feedback value f_{vu} in $F^-(u)$, the new value of p_{vu} is calculated as $p_{vu} \leftarrow p_{vu} + \frac{|\mathrm{PF}(v)|}{\mathrm{NF}(v)} f_{vu}$. For each feedback value f_{vu} in $F^+(u)$, the new value of p_{vu} is calculated as $p_{vu} \leftarrow p_{vu} + f_{vu}$.

All feedback is directly applied if $|\mathrm{PF}(v)| = \mathrm{NF}(v)$, such that $p_{vu} \leftarrow p_{vu} + f_{vu}$ for all successors u in $N^+(v)$.

At this point in differential normalization the total out probability, excluding the self-loop, of node v is the same as before applying feedback. However, the self-loop may have changed since normalization was last run, reducing or increasing the probability available to edges in $N^+(v) \setminus v$. If this is the case, we need to slightly adjust the final probabilities such that $\mathrm{Out}(v) = 1$. The is accomplished via a normalization very similar to the one used in standard normalization,

$$\forall u \in N^+(v) : p_{vu} \leftarrow p_{vu} \frac{1 - p_{vv}}{\sum_{N^+(v)\setminus v} p_{vu}}. \tag{4.9}$$

Unless the change in the self-loop is large, this bias changes the transition probabilities only slightly. An alternative scheme could do away with this final weighting step in almost all instances by reserving probability for the self-loop before calculating the new transition probability values.

4.5.5 Order of Operations

The ordering of updates, feedback, and normalization is important, although the wrong ordering will only slow convergence. Figure 4.4 gives the ideal situation, where a node (1) sends probability updates to its predecessors, (2) receives and records feedback, and then (3) applies feedback, adjusts the self-loop, and normalizes its out-edges. A new round is then begun.

4.6 Simulation Analysis

Our analysis of DSC considers two main questions.

1. How does DSC perform across a wide range of static topologies, both in terms of convergence speed, and expected sample walk length?

2. Is DSC resilient under churn conditions?

In exploring the first question, we desire to both demonstrate the correctness of DSC and to quantify convergence dynamics. We also explore the effect of different levels of link-directness on convergence, and show that the performance of DSC degrades gracefully as the number of directed links increases.

After having established the basic efficacy of DSC, we wish to show that the algorithm continues to perform well under churn conditions. Although our proof of the convergence theorem (Theorem 4.3.1) shows that DSC always converges, a result the following evaluation of static topologies fully supports, the process of churn constantly changes the underlying topology such that DSC can never completely converge. Our desire is to show that DSC can nevertheless minimize the difference between the most and least likely to be sampled peer, and effectively bias the network even as it changes.

Crucial to functioning of DSC under churn is the relaxation process. In the above description of relaxation we claimed that relaxation is a local phenomenon, but offered no proof. In the results on topologies under churn we show that relaxation is effective – to a point. We discuss the limitations of the current implementation and discuss future directions to improve the performance of relaxation.

To characterize convergence and walk lengths, we use a discrete-event simulation of DSC. To analyze the effects of churn, we run a concrete implementation of DSC integrated within the FreePastry framework [26] under various levels of churn.

4.6.1 Static Setup

We simulate DSC over a diverse set of generated graphs. In particular, we use three types of generative models commonly used in peer-to-peer research:

- *Kleinberg:* peers are connected in a lattice, along with q long distance links chosen with probability inversely proportional to the squared distance [49]. Kleinberg topologies fall into the category of "small-world" networks [78; 83], yet maintain good greedy routing properties.

- *Erdős-Rényi:* edges are selected independently with a fixed probability related to the target number of edges [2]. Erdős-Rényi topologies are considered as the standard for "random graph" models.

- *Pastry*: vertices are connected with immediate neighbors in a lattice (the leaf-set). Additional edges emulate the Pastry route table [66]. Our Pastry graphs are calculated statically from a set of randomly generated ids, such that the resulting route tables are optimal.

We use variants of these standard models to obtain directed graphs. All graphs have the same number of vertices, $n \in \{100, 1000, 10000\}$, and the same number of edges $|E| = n \log n$. This number of edges corresponds to a peer–edge ratio found in many deployed systems. The exception to this rule are the Pastry topologies, where the number of edges are dictated by the size of the leafset and the number of peers in the route table of each peer. This means that the Pastry graphs typically have more than two times the number of edges of the other topologies.

We run a number of versions of DSC across the generated topologies. The main parameters of exploration are:

1. *Percentage of directed edges,* ρ. We experiment with six different levels of directness: $0.0, 0.2, 0.4, 0.6, 0.8, 1.0$. The topologies are biased when they are initially loaded by the simulator, with each edge considered separately. Those that are marked as directed do not allow feedback to be sent along them.

2. *Feedback value*: We experiment with both asymptotic and optimal feedback.

3. *Normalization technique*: We experiment with both standard normalization and differential normalization.

4. *Dampening factor,* α: We experimented with five settings of the exponential decay dampening factor: $0.2, 0.4, 0.6, 0.8, 1.0$. A value of 1.0 indicates that no dampening is used.

To measure the quality of the resulting sampling service, we compute and use two main values. The *uniformity ratio*,

$$r = \frac{\max(\pi)}{\min(\pi)},$$

highlights the worst-case difference in sample probability, and therefore gives a concise but conservative characterization of the uniformity achieved by DSC. The *statistical distance* d (sometimes called the total variation distance),

$$d\left(\frac{1}{n}, \pi\right) = \frac{1}{2} \sum_{v \in V} \left| \frac{1}{n} - \pi(v) \right|,$$

measures the normalized absolute difference between π and the uniform distribution. Theoretically, the statistical distance is equivalent to an upper bound on the probability that an optimal algorithm can correctly determine if a large set of samples from V deviates from uniform.

In all figures below, the uniformity ratio r and statistical distance d are plotted as a function of the number of advancements per peer, i/n, where i is the total number of advancements over the network. Notice that advancements execute in parallel, so i/n represents the number of rounds of advancements in the network.

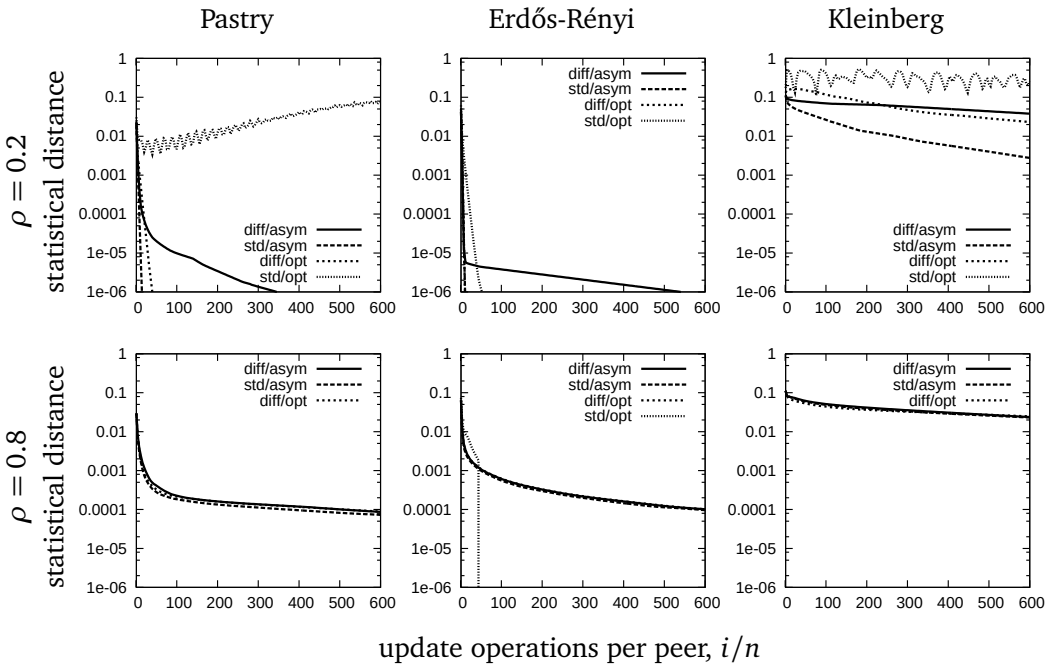


Figure 4.5. Statistical distance d after i/n updates for both norms and feedback types, for $\rho \in \{0.2, 0.8\}$, $\alpha = 0.6$, 10000 vertices.

4.6.2 Static Convergence

We first desire to show that DSC converges when parameterized properly. We simulate DSC on 10 topologies for each generative model and network size, and plot the median value of r and d. Figure 4.5 shows the statistical distance for all four combinations of feedback and normalization over 10000 vertex graphs: differential normalization with asymptotic feedback (diff/asym), standard normalization with asymptotic feedback (std/asym), differential normalization with optimal feedback (diff/opt), and standard normalization with optimal feedback (std/opt). For now we set the exponential dampening factor to 0.6, which we will later see is the best value of those tested, and look at topologies that are either slightly ($\rho = 0.2$) or very ($\rho = 0.8$) directed. Standard normalization with optimal feedback setting biased topologies such that the sparse eigendecomposition algorithm that we used could not successfully decompose them within a reasonable time. Therefore, the line for standard normalization with optimal feedback is missing for the Pastry and Kleinberg topologies when $\rho = 0.8$.

Figure 4.5 shows that standard normalization with optimal feedback does not perform well for Pastry and Kleinberg topologies. We have confirmed that these topologies do indeed converge (after > 3000 cycles), but they tend to experience periods of decreasing length where the distance from doubly stochastic increases. This is a clear example where feedback can hurt convergence. The other normalization/feedback

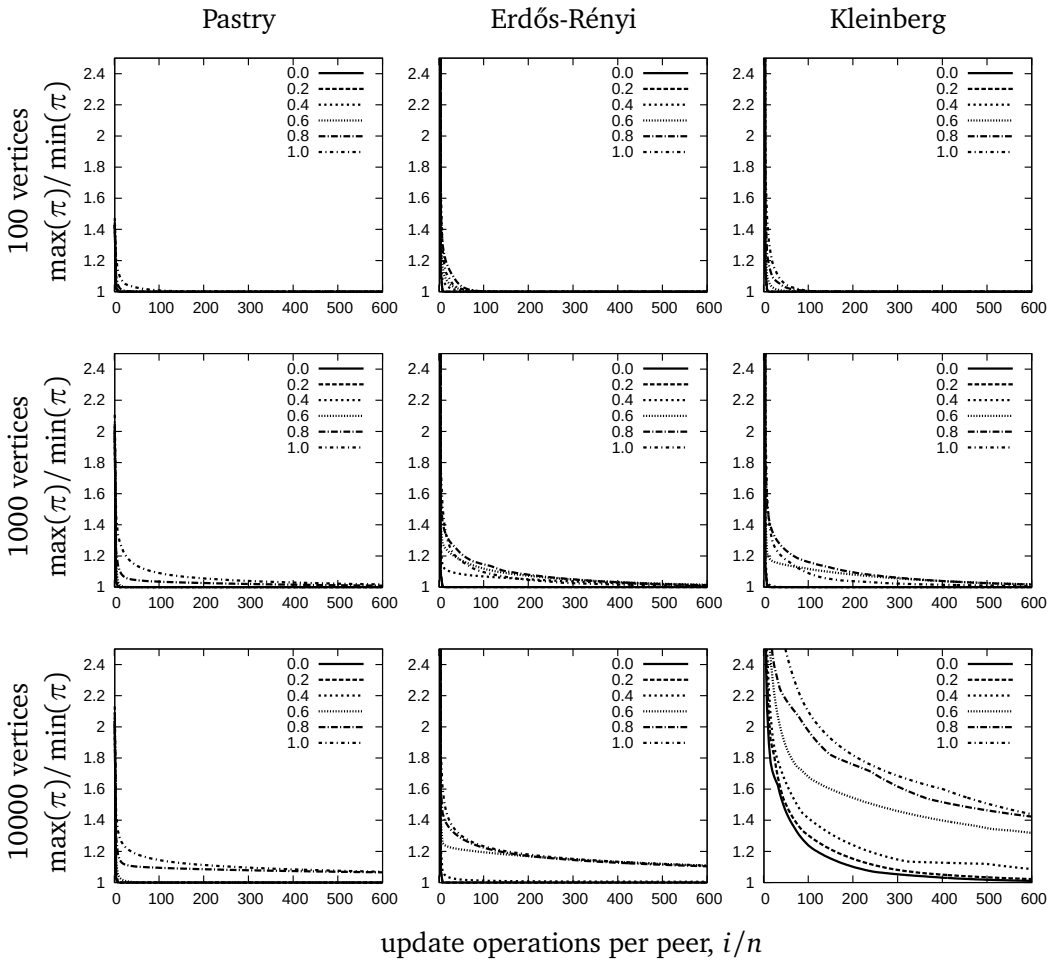


Figure 4.6. Uniformity ratio r after i/n updates for the standard norm, asymptotic feedback, and $\alpha = 0.6$.

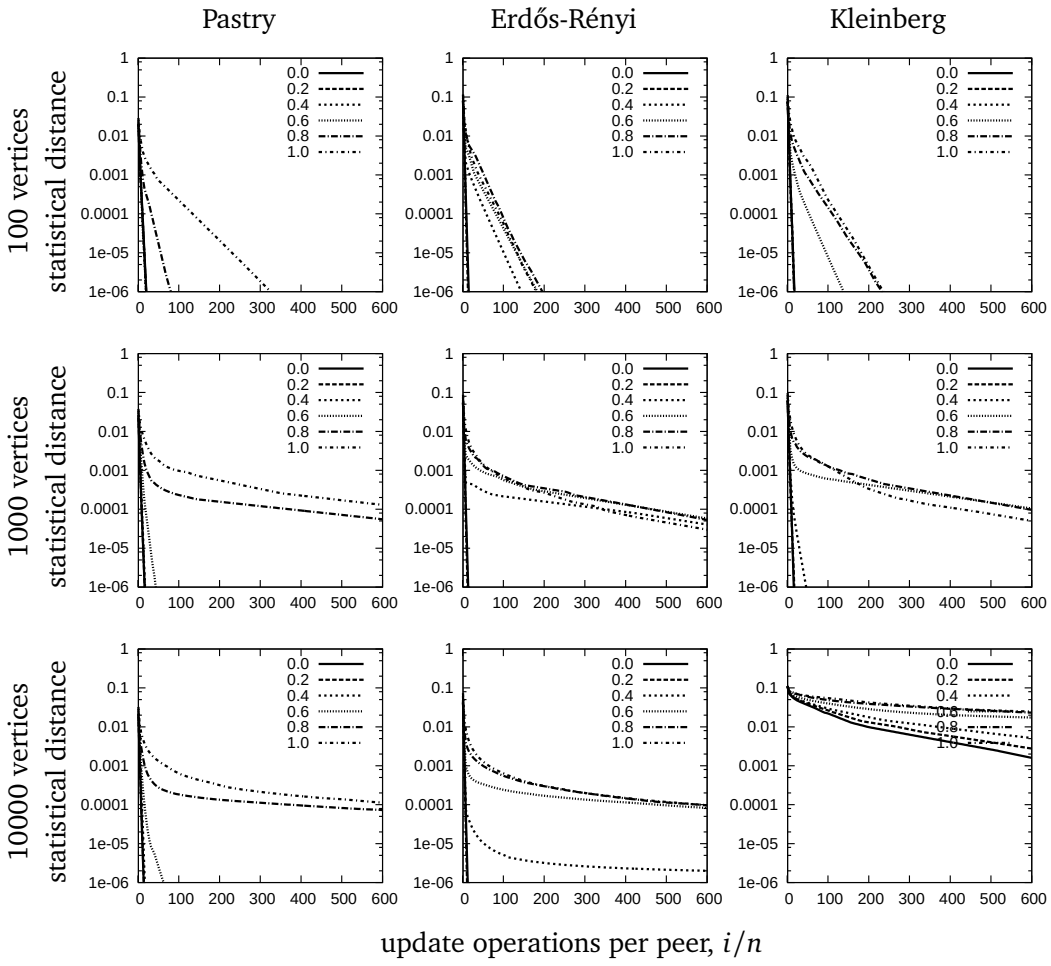


Figure 4.7. Statistical distance d after i/n updates for the standard norm, asymptotic feedback, and $\alpha = 0.6$.

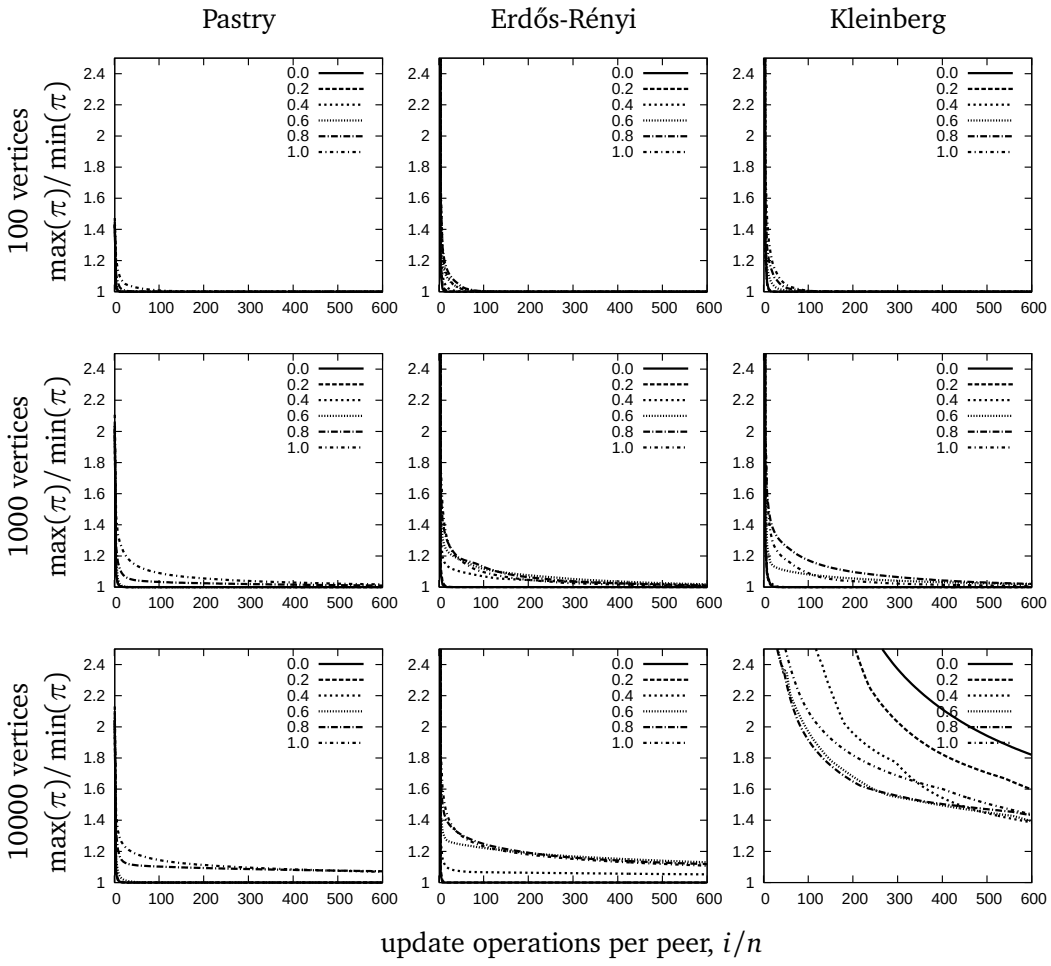


Figure 4.8. Uniformity ratio r after i/n updates for the differential norm, optimal feedback, and $\alpha = 1.0$.

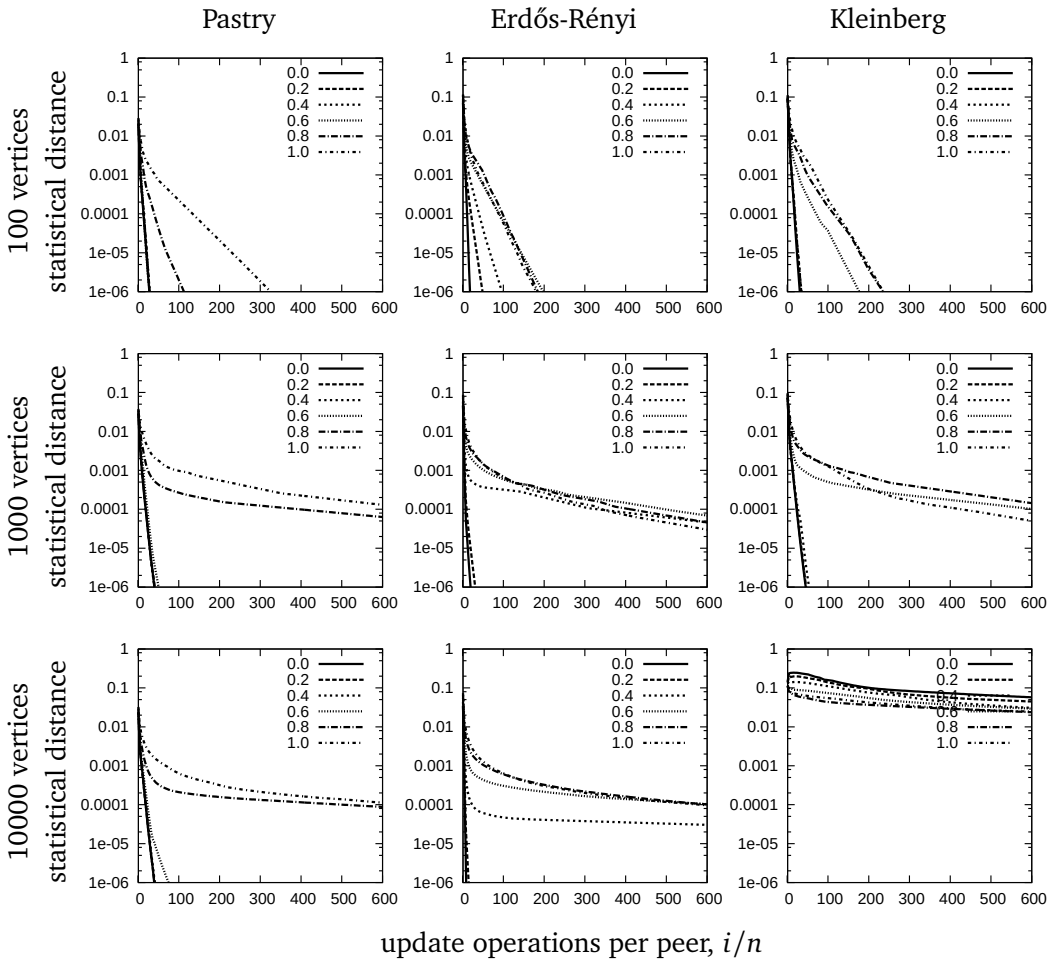


Figure 4.9. Statistical distance d after i/n updates for the differential norm, optimal feedback, and $\alpha = 1.0$.

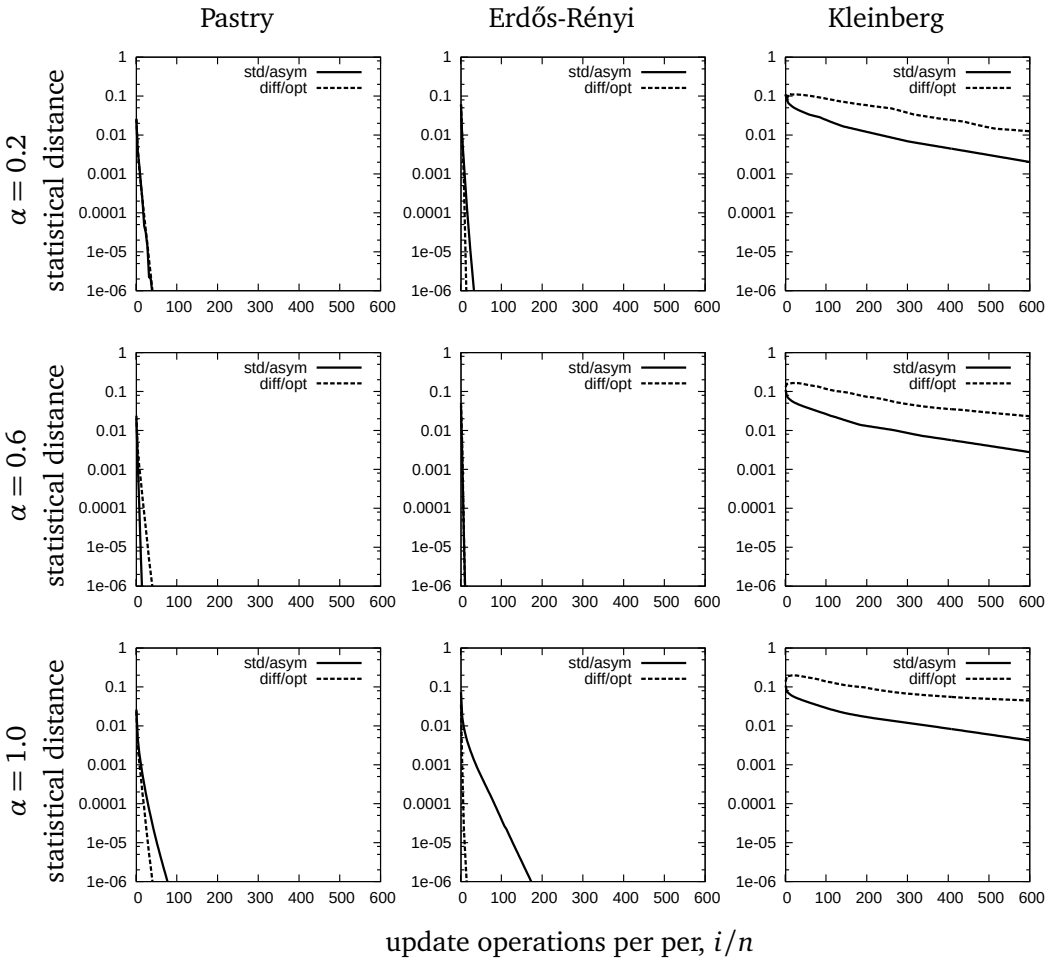


Figure 4.10. Statistical distance d after i/n updates for the standard norm with asymptotic feedback, and the differential norm with optimal feedback, for $\alpha \in \{0.2, 0.6, 1.0\}$ and $\rho = 0.2$.

combinations work much better. In particular, standard normalization with asymptotic feedback and differential normalization with optimal feedback converge the fastest of the four possible parings. As such, we focus on these two schemes for the rest of the evaluation.

A somewhat more intuitive, if conservative, metric of quality is the uniformity ratio r. Figures 4.6 and 4.8 show r for standard normalization with asymptotic feedback and differential normalization with optimal feedback, respectively. Figures 4.7 and 4.9 give the corresponding statistical distance plots. These plots only show convergence for the fastest two of the four possible parings of normalization and feedback. All four plots show convergence for different levels of directionality of the graphs, across different topology sizes.

The general trend of these plots is clear: the more bidirectional connections, the faster DSC converges. An exception to this is for highly directed topologies. For some topology types, completely directed graphs perform slightly better than graphs that are almost entirely directed. We do not have a complete explaination for this behavoir. It appears that for large Kleinberg topologies where DSC is using differential normalization with optimal feedback that the feedback process does more harm than good, such that highly directed graphs converge slightly more quickly.

Kleinberg topologies converge slowly. While Kleiberg topologies support greedy routing in $O(\log n)$ steps, they have very poor mixing properties. The non-lattice links are distributed proportionally to $1/d(u,v)^2$, where $d(u,v)$ is the lattice distance between peers v and u [49]. While this distribution assures that greedy routing is optimal [49], the distribution also means that non-lattice links typically point to peers close nearby in the lattice. This tends to lead to topologies with poor mixing. Since the underlying mixing-time of the network typically limits the speed at which DSC can converge, Kleinberg topologies perform poorly.

We mentioned at the top of the section that an exponential dampening factor of $\alpha = 0.6$ usually performs the best. Figure 4.10 shows the performance of different alpha-values for slightly directed graphs ($\rho = 0.2$) of size 10000. The plots clearly show that $\alpha = 0.6$ performs best for standard normalization with asymptotic feedback, while differential normalization with optimal feedback performs nearly identically with any value. This second fact indicates that differential normalization with optimal feedback does not suffer from probability oscillation, and does not need exponential dampening. However, since the dampening also doesn't appear to degrade it's performance, we show $\alpha = 0.6$ in most of our results, and for our FreePastry implementation (see Section 4.6.4).

4.6.3 Necessary Walk Length

Quick convergence indicates that the cost of balancing is low, but does tell us how many hops we will need to sample uniformly. The sampling cost is determined by the mixing-time, that is, the number of hops after which a random walk becomes independent of

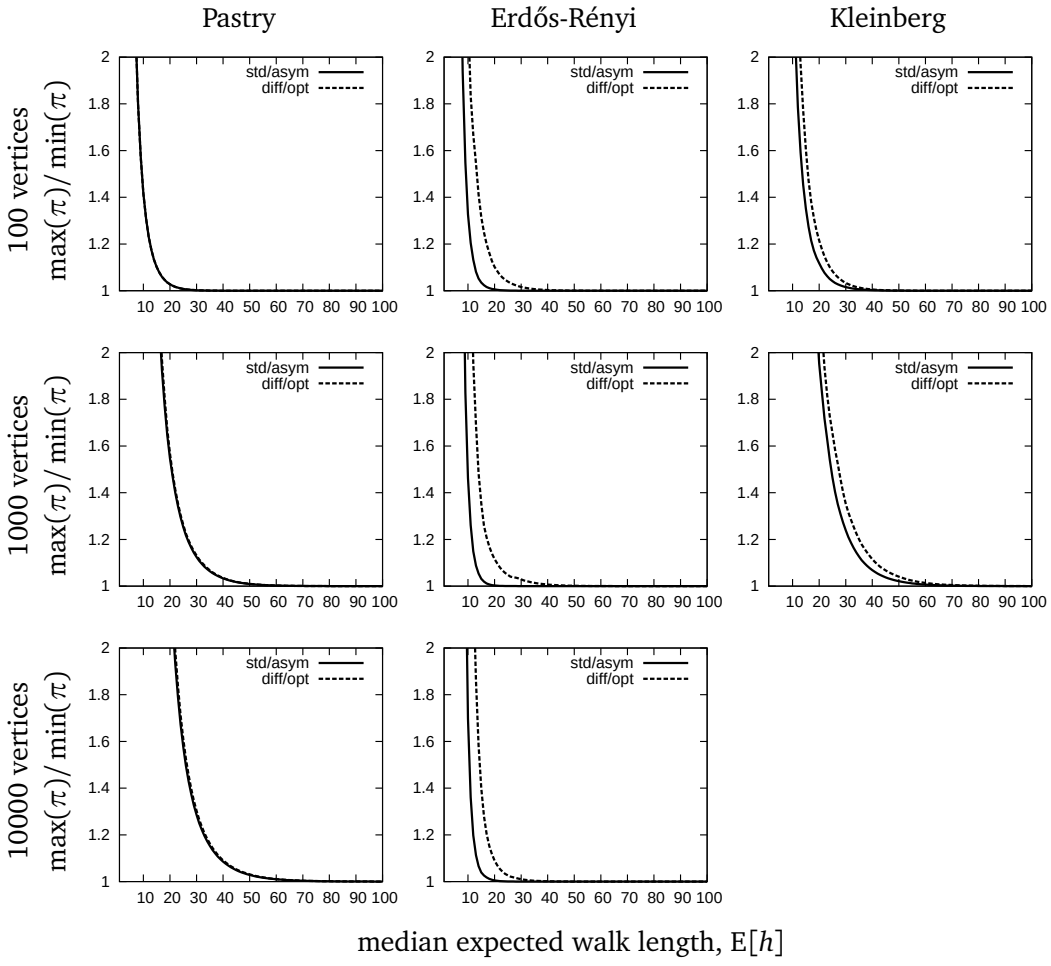


Figure 4.11. Uniformity ratio r as the expected median walk length $\mathrm{E}[h]$ increases, for both the standard norm with asymptotic feedback, and the differential norm with optimal feedback, for $\alpha = 0.6$ and $\rho = 0.2$.

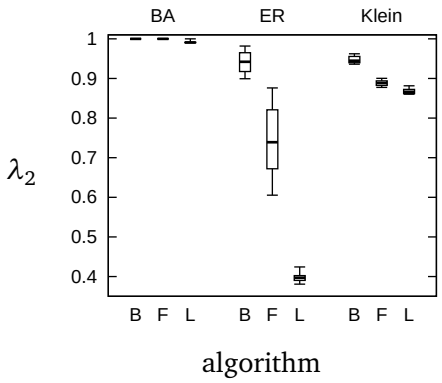


Figure 4.12. Second eigenvalue for basic DSC (B), full DSC (F), and linear-programming (L), for each topology type: Barabási-Albert (BA), Erdős-Rényi (ER), and Kleinberg (Klein).

its starting peer. We analyze this cost using the results from Markov-chain theory introduced in Chapter 3. After the convergence of each topology, we calculate the median expected walk-length $\mathrm{E}[h]$ for different values of r.

Figure 4.11 show the results of our analysis for of topology sizes, $\rho = 0.2$, and $\alpha = 0.6$. No plot is given for 10000 peer Kleinberg topologies, as the required walk length is far greater than 100 hops (indeed, greater than 5000 hops). We do not fully understand why the walk length explodes when scaling from 1000 to 10000 vertices for Kleinberg topologies. For the other topology types the walk length scales very well, and we see virtually no increase in necessary walk length as the topologies grow. Erdős-Rényi topologies typically have low mixing times, and DSC successfully maintains this property with a median sample length of 30 for all topology sizes. Pastry is "middle of the road" – not a great mixer, but not horrible either ($\lambda_2 \sim 0.8$), and DSC scales gracefully for a median sample length of 60 for the largest topologies.

In evaluating walk lengths in DSC, we would like to compare the results of DSC with the best achievable walk lengths for each topology. Unfortunately, we are not aware of a suitable optimization algorithm for topologies of the scale we considered (> 100 vertices) [9]. In order to obtain an estimate of the minimal walk length, we cast the approximate optimal solution as a linear-programming problem. We then compare DSC with the linear-programming solution using, as an indicator of necessary walk length, the second-largest eigenvalue modulus λ_2 of the transition matrix P. See Section 3.3.1 for more information on the relation of λ_2 to sample walk length.

Our objective function minimizes the self-loops, while trying to balance out-going

probability evenly across out-going edges:

$$\begin{aligned} \textbf{Minimize} \quad & w_0 \sum_{u \in V} p_{uu} + w_1 \sum_{u \in V} \max(|p_{uv} - p_{uw}|) \\ \textbf{s.t.} \quad & \forall u \in V : \sum_{v \in V} p_{uv} = 1, \\ & \forall u \in V : \sum_{v \in V} p_{vu} = 1 \end{aligned}$$

The second term of the objective function is linearized with an auxiliary variable. We do not list four additional constraint terms which force the resulting weighted topology to be ergodic. The objective function weights w_0 and w_1 were varied so as to obtain the solution with the best average behavior over all topologies. The weighting $w_0 = 1.0, w_1 = 0.5$ proved to be the best parameterization tried.

Since minimizing the mixing time is a non-linear problem, we expect the linear-programming solution to be sub-optimal. However, for many of the Erdős-Rényi, and a few of the Kleinberg topologies, the linear-programming solver found solutions with no self-loops, leading us to believe the weightings are likely close to optimal.

Figure 4.12 compares the ability of DSC to minimize walk length with the linear-programming solution, and graphs the second-largest eigenvalue λ_2 of P. We note that this data is from a separate set of experimental runs that included Barabási-Albert topologies instead of Pastry topologies, and did not include exponential dampening. We have found exponential dampening to reduce the mixing time of biased graphs, and therefore Figure 4.12 likely over-estimates the mixing time of DSC. Nevertheless, as the plot indicates, there is room for improvement in the way DSC assigns probability across edges. The linear-programming method minimizes λ_2 to a greater extent thanks to its global view of the network, whereas DSC operates in a completely decentralized manner.

4.6.4 FreePastry Simulations

In the second part of our evaluation, we study the behavior of DSC under churn by running it within FreePastry, an implementation of the fault tolerant, locality-aware Pastry DHT [66]. Our churn model follows the findings of Stutzbach and Rejaie and their study of a DHT based on Kademlia [57; 73]. We use FreePastry in simulation mode and look at 100 and 1000-peer runs, with the churn process starting as soon as the last peer has joined. After the initial join period is complete, samples of the uniformity ratio and statistical distance are collected for 5 minutes of simulated time. For all runs in FreePastry we used standard normalization with asymptotic feedback and an alpha-value of 0.6 for exponential dampening.

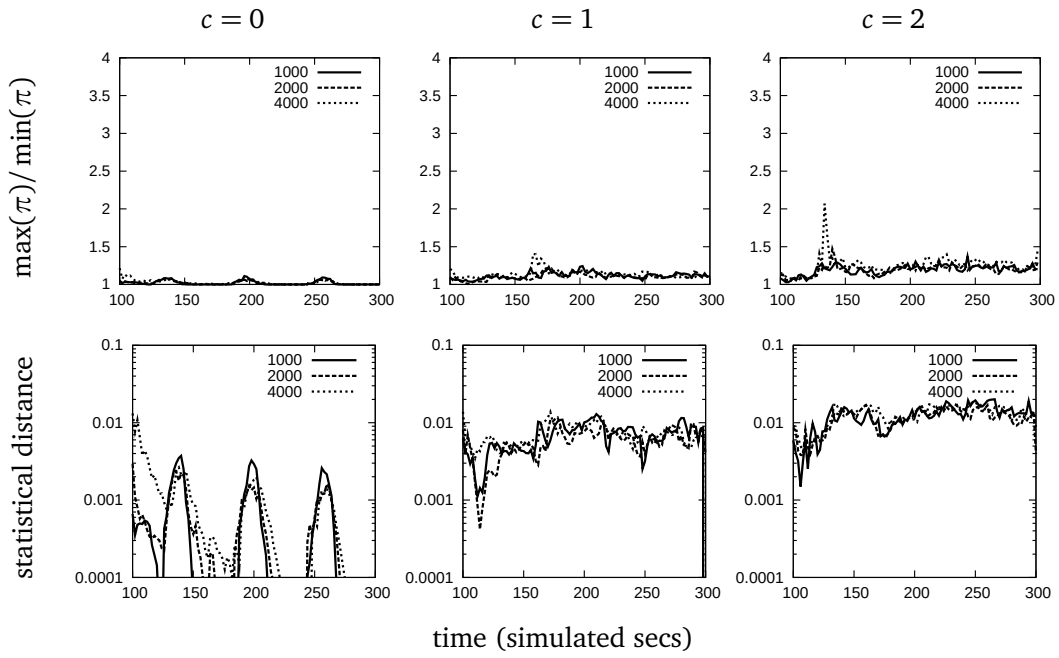


Figure 4.13. Uniformity ratio r under churn and statistical distance d for 100-peer FreePastry simulations, churn levels $c \in \{0, 1, 2\}$.

Convergence under Churn

The plots in Figure 4.13 show the uniformity ratio and statistical distance for different levels of churn c. Our churn model follows the findings of Stutzbach and Rejaie and their study of a DHT based on Kademlia [57; 73]. In particular, we model peer session time as random variates of a Weibull distribution, with the shape parameter $k_s = 0.4$ and the scale $\lambda_s = 30/c$. We refer to c as the *churn parameter*; the higher c, the shorter the average session length. Peer inter-arrival time is modeled by a second Weibull distribution, with $k_a = 0.65, \lambda_a = \mu_s/\left(N\Gamma(1+\frac{1}{k_a})\right)$, where μ_s is the mean session time, and $\Gamma(x)$ is the Gamma function. Both distributions model time in minutes. $c = 1$ indicates churn dynamics very similar to those measured by Stutzbach and Rejaie, while $c = 2$ indicates twice the level of churn events [73]. The session lengths that result from this configuration are given in Figure 4.14.

We ran multiple experiments in FreePastry with different frequency of DSC updates. The lines in Figure 4.13 show data for 1 second, 2 second, and 4 second update frequencies (labeled in milliseconds) for networks of 100 peers. All three timings do equally well at this small topology size.

In the $c = 0$ plots, there is a cyclic process that deconverges the network every 60 seconds. This is caused by the structural maintenance routines of FreePastry updating links in the network, such that the route tables of each peer points to the optimal set of

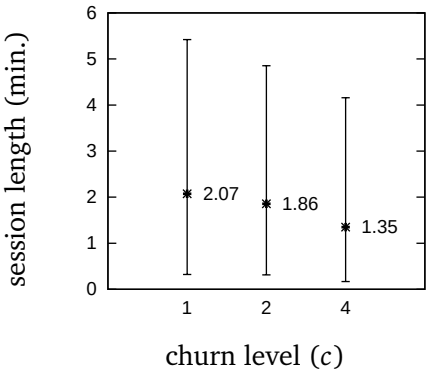


Figure 4.14. Session lengths (median, 1st and 3rd quartile)

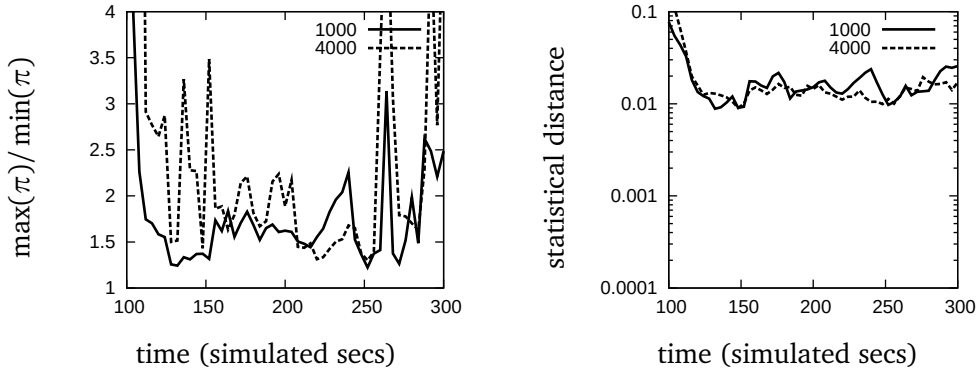


Figure 4.15. Uniformity ratio r under churn and statistical distance d for 1000-peer FreePastry simulations, churn level $c = 1$.

peers currently known. This process is present in the other graphs as well, but is masked by churn events.

Results for 1000 peer networks are given in Figure 4.15 for $c = 1$. While DSC initially biases and nearly converges, as time progresses it struggles to keep up with the churn. This was surprising until we looked at longer runs. Figure 4.16 shows a single long 1000 peer simulation, and plots the second eigenvalue and total sum of relaxation as well as statistical distance and uniformity ratio. As time progresses, DSC has a harder and harder time countering churn. After 1000 seconds of simulated time, DSC is nearly completely ineffective. The plot of λ_2 makes clear that relaxation is failing to maintain low self-loops. Indeed, manual inspection of the topology found most self-loops with values greater than 0.99.

As DSC loses control of the self-loops and the second eigenvalue approaches 1, the system becomes much more sensitive to perturbations. Small changes to the topology knock the stationary distribution far from uniform, and the precision of the transition

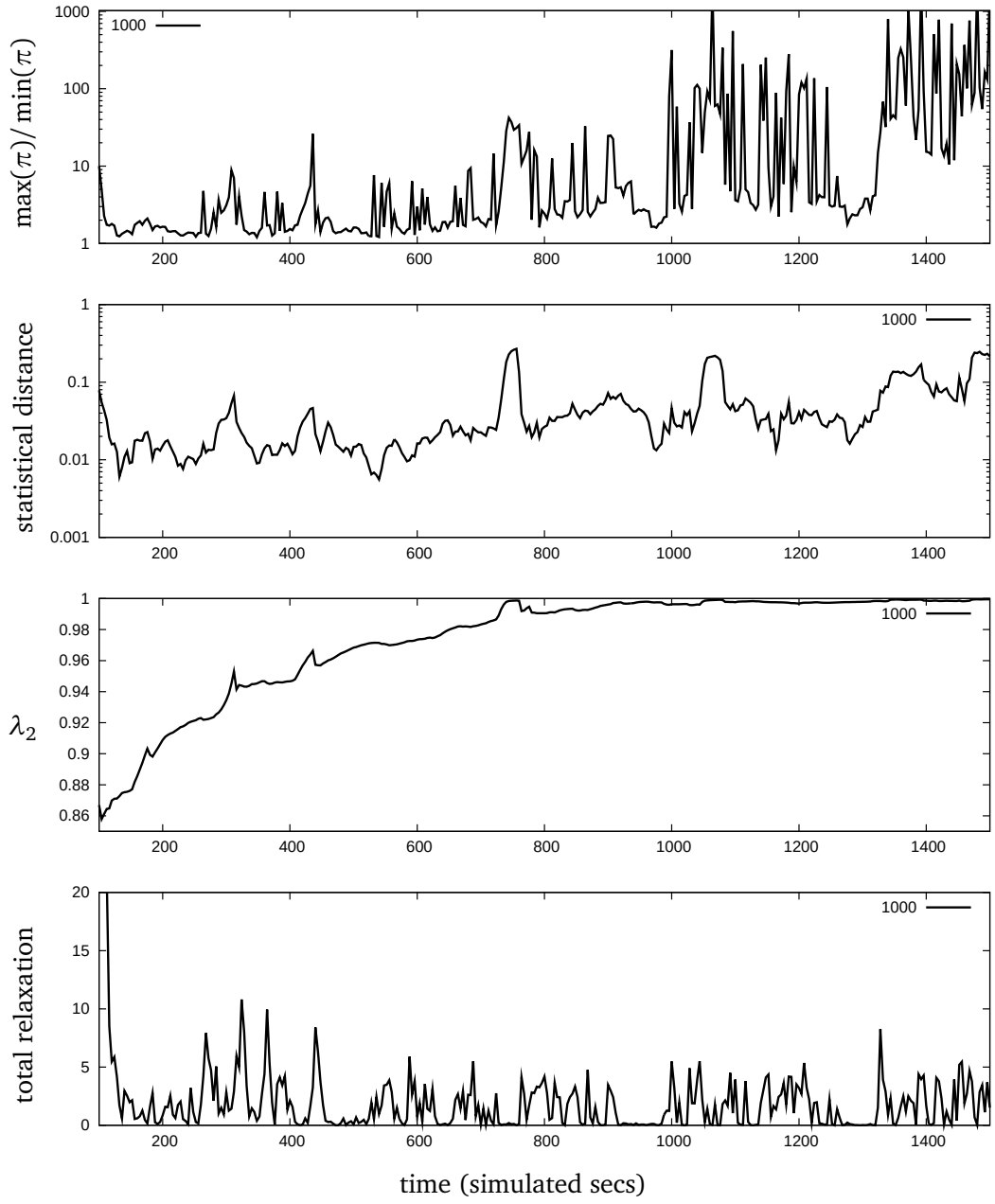


Figure 4.16. Uniformity ratio, statistical distance, the second eigenvalue, and the total sum of relaxation across time for a 1000-peer FreePastry simulation, churn level $c = 1$.

probabilities must be higher to converge back toward being doubly stochastic. This demonstrates the need for relaxation. Yet, as can be seen in Figure 4.16, relaxation does not react sufficiently to this situation. The defualt relaxation process is only triggered by local events, and assures that the sum of in-probability never drops below 1. This does not appear to be sufficient. Our hypothesis is that as churn changes the topology, peers over-converge, rather than adapting properly to their changing neighbors sets.

A relaxation process that purposely pushes the in-probability of peers below 1 would likely counteract this. By dumping probability back onto out-going links, downstream nodes will also be forced to relax, leading the network to rebalance. The challange is to design a scheme such that such over relaxation does not perminantly deconverge the weightings. We introduce three additional relaxation mechanisms to achieve this.

4.6.5 Improving Relaxation

Since relaxation on its own is not enough to stop self-loops from slowly climbing toward 1.0, we introduce three mechanisms to make further use of relaxation. The first of these is *over-relaxation*, which relaxes a multiple of γ more than the amount allowed by the basic version of relaxation. The second mechanism places a soft-threshold κ on the self-loop at each peer, and forces relaxation after a timeout if the threshold is crossed. We call this *self-loop thresholding*. Finally we *latch* the self-loop of a peer after it performs relaxation, and disallow any increase in its value for l latch rounds.

In normal relaxation, the self-loop is reduced by the amount $\min(p_{uu}, \text{In}(u) - 1)$ during the update process at any peer u where $u \in V^+$ and $p_{uu} > 0$. This sets the self-loop as close to 0 as possible. Over-relaxation simply extends this action by multiplying the decrease by a maximum of the parameter γ, such that the peer relaxes the self-loop by $\min(p_{uu}, \gamma\text{In}(u)-1)$. Unlike normal relaxation, if γ is greater than 1 the peer may end its update with its in-probability no longer balanced, such that $\text{In}(u) < 1$ and $u \in V^-$.

The intuition behind over-relaxation is that DSC converges such that the weightings it produces over time do not change enough over time, after a number large of churn events. Only immediate neighbors of new or removed nodes see even moderate changes in their weightings over time, while peers that are farther away are more or less stay stable. However, as the topology of the network changes, such stable peers may no longer have a particularly optimal set of weights. In particular, self-loops remain too high, limiting the ability of peers to provide enough in-probability to their downstream neighbors.

A similar intuition guides the introduction of self-loop thresholding. Any particular peer-to-peer network (and the graph that represents it) has a multitude of doubly stochastic weightings. Although we may be able to balance a peer with a high self-loop, it is likely that there is an alternative global weighting that would provide the peer with a lower self-loop. To try and find this weighting, self-loop thresholding performs a two step-process using relaxation. First, a peer converges normally, and the self-loop threshold κ is ignored for a set time-period (in our simulations, a fixed 30 seconds). If at any

time after the initial period the self-loop exceeds κ, it is cut in half such that $p_{uu} = p_{uu}/2$ and a new period in which the threshold is ignored begins.

Imagine a network that updates synchronously. After an initial period, it is likely that some peers have self-loops above κ and others below. Those above will relax their self-loop by half, causing a cascade of relaxation across the network. All peers will see reductions in their self-loop, and those peers where p_{uu} is still greater than κ will likely drop below. After several iterations the network will likely find a weighting where all self-loops are below κ, if such a weighting exists. From both the experiments in Section 4.6.3 with linear programming, and the simulation results, it appears that such weightings almost always exist for the topologies we have experimented with, even for very low κ values (e.g., $\kappa = 0.1$).

Latching the self-loop after relaxation amplifies the affects of the shifting probability to the out-edges of a peer. If a peer u relaxes the self-loop enough to bring the peer into V^-, then the next update round will merely increase p_{uu} such that $u \in V^=$. While eventual increases in incoming probability from the ergodic propagation of the relaxation may trigger a persistent amount of relaxation, immediately re-increasing the self-loop degrades the effect of relaxation. The re-increase effectively creates two interfering "waves" of probability changes, one decreasing self-loops, and the second increasing them. These two waves interfere, lessening the effectiveness of the first. If the initial reduction is allowed to fully or partially propagate to the peer who initially relaxed, it may either not have to increase its self-loop at all, or increase it much less.

Analysis Techniques

We carried out additional experiments with over-relaxation and self-loop thresholding implemented. Since five minute long simulation runs were insufficient to notice the degradation in the quality of the biasing produced by DSC, we lengthened the simulation runs to 15 minutes. Figure 4.16 indicates that this amount of time is sufficient to see a noticeable rise in both the convergence ration and the average statistical distance.

Our analysis of the new simulations attempts to determine if the statistical distance is stable over time. Each run is first cut into two windows: the beginning window (B), from 150 to 250 simulated seconds, and the ending window (E), from 800 to 900 seconds. The beginning window is chosen such that our fast network startup process has completed. Each window is then analyzed separately by first building the discrete cumulative distribution (CDF) of the statistical distance samples within it, and then building a profile from the CDF.

The profiles of the beginning and ending windows are compared using two sets of statistics. The first is a granular discrete CDF covering statistical distance values from 0 to 0.05. Over 99% of all statistical distance samples fall withing this range, even though the statistical distance can range between 0 and 1. These values are used to roughly sketch a picture of the dynamics of the statistical distance during a particular time period. Second, the area under the curve (AUC) is calculated for the full CDF

between the same range, 0 to 0.05. Since the AUC measures the area under the CDF curve, it gives a clear picture of how fast the CDF converges to 1, with higher values indicating better convergence of the weighting of DSC.

If the CDF stays the same or increases between the beginning and ending window, we consider DSC to have successfully maintained the probability bias through the simulation. This is then confirmed by looking at the granular CDF values for the ending window, and confirming that they are either close to, or better than, those of the starting window. In the following tables we highlight such cases in bold typeface.

Results of Over-relaxation and Self-loop Thresholding

Tables 4.1 and 4.2 give the results of this analysis for FreePastry networks of 100 peers, for churn level 1 and 2, respectively, while Table 4.3 lists results for networks of 1000 peers at churn level 1. Each row gives aggregate results over 10 independent runs of FreePastry and DSC with the same parameter settings.

Several phenomena are clear across all three tables. First, the distribution of statistical distance samples has higher variance when the cycle time of DSC is set aggressively at $q = 1000$. This means there are both more low samples (< 0.0075) indicating good convergence, and high samples (> 0.025) indicating poor convergence. This is a direct result of converging faster. Since a less converged weighting changes less during a churn event, a less aggressive cycle timing naturally produces less variance in statistical distance across time. Depending on the sampling situation, an either an aggressive or non-aggressive parameterization may be appropriate. This pattern regarding the sample variance is noticeably less pronounced in Table 4.3 for the 1000 peer topologies, where the slower cycle time of 4000 is less able to keep up with changes in the underlying network and the distribution shifts rightward to higher values.

Another clear pattern across all three tables is that a low self-loop threshold (κ) tends to result in a weighting that produces either a stable or improving statistical distance distribution. Higher latch times show the same trend. These two phenomena are directly related. In our experience, lower self-loops result in a network with a lower mixing-time. Further, in studying the data that was used to generate Figure 4.16 and related data runs, it appears that a lower mixing time decreases the divergence of the stationary distribution when holding the total amount of divergence from having a doubly stochastic transition matrix equal. As such, a small κ is expected to be beneficial.

When the latch time is too small, or no latch is used ($l = 0$), the additional relaxation caused by the low self-loop threshold is poorly absorbed, and results in unnecessary alternating rounds between relaxation and convergence. This inefficiency stops DSC from reaching a particularly good weighting, an effect particularly noticeable when inspecting rows for $\kappa = 0.1$ in Table 4.3.

Figure 4.17 reports the amount of relaxation for a single round of updates (i.e., each peer updates once) and demonstrates this problem. When κ is 1.0, relaxation remains lows, as relaxation only takes place when the self-loop can be reduced due to

q	γ	κ	l		% Samples Below Threshold 0.0075	0.01	0.015	0.02	0.025	0.05	AUC (0–0.05)
1000	2.0	0.4	8	B	0.636	0.780	0.912	0.964	0.988	1.0	0.04344
				E	0.528	0.660	0.908	0.956	0.972	1.0	0.04221
			4	B	0.604	0.744	0.908	0.960	0.98	0.992	0.04313
				E	0.488	0.656	0.860	0.916	0.964	1.0	0.04144
		0.1	8	B	0.640	0.776	0.936	0.968	0.984	1.0	0.04374
				E	**0.552**	**0.720**	**0.924**	**0.996**	**1.0**	**1.0**	**0.04314**
			4	B	0.588	0.736	0.896	0.948	0.972	0.996	0.04301
				E	0.508	0.632	0.844	0.944	0.980	1.0	0.04184
	3.0	0.4	8	B	0.632	0.752	0.904	0.956	0.996	0.996	0.04330
				E	0.508	0.668	0.876	0.964	0.992	1.0	0.04213
			4	B	0.616	0.764	0.912	0.976	0.992	0.996	0.04333
				E	0.488	0.648	0.848	0.944	0.964	1.0	0.04154
		0.1	8	B	0.652	0.816	0.944	0.968	0.980	1.0	0.04398
				E	**0.576**	**0.728**	**0.944**	**0.984**	**1.0**	**1.0**	**0.04328**
			4	B	0.608	0.760	0.900	0.952	0.980	0.996	0.04334
				E	0.536	0.632	0.848	0.948	0.984	1.0	0.04222
4000	2.0	0.4	8	B	0.412	0.588	0.880	0.972	0.992	1.0	0.04105
				E	**0.488**	**0.660**	**0.920**	**1.0**	**1.0**	**1.0**	**0.04207**
			4	B	0.412	0.592	0.872	0.956	0.988	1.0	0.04091
				E	**0.472**	**0.644**	**0.900**	**0.992**	**1.0**	**1.0**	**0.04188**
		0.1	8	B	0.428	0.628	0.880	0.972	0.992	1.0	0.04121
				E	**0.508**	**0.744**	**0.948**	**1.0**	**1.0**	**1.0**	**0.04252**
			4	B	0.440	0.640	0.892	0.972	0.996	1.0	0.04152
				E	**0.516**	**0.704**	**0.936**	**1.0**	**1.0**	**1.0**	**0.04248**
	3.0	0.4	8	B	0.420	0.624	0.884	0.964	0.992	1.0	0.04121
				E	**0.444**	**0.676**	**0.932**	**1.0**	**1.0**	**1.0**	**0.04212**
			4	B	0.424	0.600	0.868	0.948	0.984	1.0	0.04099
				E	**0.552**	**0.736**	**0.944**	**0.988**	**1.0**	**1.0**	**0.04321**
		0.1	8	B	0.436	0.632	0.888	0.968	0.992	1.0	0.04135
				E	**0.520**	**0.736**	**0.948**	**1.0**	**1.0**	**1.0**	**0.04257**
			4	B	0.436	0.616	0.896	0.976	0.992	1.0	0.04151
				E	**0.512**	**0.720**	**0.936**	**1.0**	**1.0**	**1.0**	**0.04253**
q	γ	κ	l		0.0075	0.01	0.015	0.02	0.025	0.05	

Table 4.1. Cumulative statistical distance percentiles for $N = 100$ FreePastry networks, churn level 1.0. The first four columns are q, the DSC update rate, γ, the relaxation multiplier, κ, the self-loop threshold, and l, the self-loop latch-time. Column four marks whether a set of values is from the beginning (B), after all peers have joined, or the end (E) of the runs. The next six columns list the percentage of statistical distance samples below the given threshold. The final column gives the area under the curve (AUC) of the discrete cumulative distribution function (CDF) between the value 0 and 0.05; the larger the AUC, the more samples that fall to the left of the CDF. Bold values in E rows indicate the distribution of statistical distance samples has not deteriorated.

q	γ	κ	l		% Samples Below Threshold 0.0075	0.01	0.015	0.02	0.025	0.05	AUC (0–0.05)
1000	2.0	0.4	8	B	0.360	0.456	0.660	0.796	0.876	0.992	0.03688
				E	0.208	0.256	0.472	0.592	0.716	0.952	0.03168
			4	B	0.380	0.484	0.640	0.772	0.856	0.984	0.03634
				E	0.132	0.200	0.408	0.616	0.712	0.908	0.02932
		0.1	8	B	0.364	0.464	0.696	0.824	0.908	0.992	0.03750
				E	0.144	0.196	0.428	0.620	0.748	0.976	0.03168
			4	B	0.328	0.404	0.608	0.736	0.812	0.976	0.03528
				E	0.128	0.168	0.340	0.516	0.620	0.916	0.02744
	3.0	0.4	8	B	0.352	0.456	0.676	0.812	0.900	0.996	0.03756
				E	0.128	0.208	0.408	0.628	0.756	0.968	0.03083
			4	B	0.340	0.460	0.624	0.776	0.884	0.988	0.03677
				E	0.236	0.304	0.476	0.672	0.764	0.940	0.03232
		0.1	8	B	0.368	0.520	0.724	0.844	0.896	0.996	0.03822
				E	0.136	0.192	0.464	0.620	0.760	0.984	0.03200
			4	B	0.340	0.440	0.636	0.756	0.860	0.988	0.03661
				E	0.140	0.192	0.368	0.540	0.668	0.956	0.02967
4000	2.0	0.4	8	B	0.172	0.336	0.584	0.784	0.924	1.0	0.03588
				E	0.108	0.156	0.324	0.568	0.836	1.0	0.03192
			4	B	0.168	0.324	0.616	0.812	0.940	1.0	0.03617
				E	0.100	0.124	0.296	0.540	0.736	0.948	0.02940
		0.1	8	B	0.176	0.316	0.632	0.848	0.972	1.0	0.03666
				E	0.108	0.160	0.396	0.696	0.852	1.0	0.03305
			4	B	0.184	0.332	0.628	0.848	0.968	1.0	0.03674
				E	0.112	0.152	0.376	0.628	0.832	1.0	0.03233
	3.0	0.4	8	B	0.188	0.340	0.604	0.816	0.912	1.0	0.03602
				E	0.108	0.180	0.388	0.620	0.816	1.0	0.03244
			4	B	0.184	0.320	0.620	0.824	0.904	1.0	0.03594
				E	0.116	0.176	0.348	0.536	0.736	0.976	0.03033
		0.1	8	B	0.192	0.344	0.636	0.848	0.956	1.0	0.03675
				E	0.108	0.172	0.424	0.772	0.860	1.0	0.03372
			4	B	0.204	0.336	0.636	0.848	0.956	1.0	0.03684
				E	0.124	0.180	0.392	0.640	0.856	0.992	0.03286
q	γ	κ	l		0.0075	0.01	0.015	0.02	0.025	0.05	

Table 4.2. Cumulative statistical distance percentiles for $N = 100$ FreePastry networks, churn level 2.0. Refer to Figure 4.1 for a full description of the table fields.

q	κ	l		% Samples Below Threshold 0.0075	0.01	0.015	0.02	0.025	0.05	AUC (0–0.05)
1000	1.0	8	B	0.100	0.268	0.873	1.0	1.0	1.0	0.03831
			E	0.0	0.065	0.353	0.635	0.853	1.0	0.03166
		4	B	0.170	0.395	0.780	0.960	1.0	1.0	0.03816
			E	0.028	0.075	0.220	0.448	0.705	1.0	0.02845
		0	B	0.180	0.360	0.740	0.893	0.949	1.0	0.03750
			E	0.007	0.042	0.138	0.273	0.489	0.940	0.02255
	0.4	8	B	0.100	0.308	0.775	0.893	0.963	1.0	0.03726
			E	0.058	0.185	0.565	0.878	0.990	1.0	0.03566
		4	B	0.060	0.220	0.710	0.948	1.0	1.0	0.03688
			E	0.040	0.128	0.510	0.918	1.0	1.0	0.03530
		0	B	0.111	0.358	0.740	0.940	1.0	1.0	0.03773
			E	0.060	0.109	0.456	0.767	0.953	1.0	0.03413
	0.1	8	B	0.058	0.195	0.785	0.988	1.0	1.0	0.03745
			E	**0.115**	**0.370**	**0.818**	**0.985**	**1.0**	**1.0**	**0.03851**
		4	B	0.070	0.210	0.623	0.888	0.970	1.0	0.03610
			E	0.010	0.095	0.335	0.653	0.850	1.0	0.03192
		0	B	0.044	0.176	0.470	0.810	0.926	1.0	0.03396
			E	0.030	0.050	0.168	0.424	0.618	1.0	0.02714
4000	1.0	8	B	0.008	0.048	0.452	0.848	0.966	1.0	0.03399
			E	**0.032**	**0.142**	**0.510**	**0.850**	**0.952**	**1.0**	**0.03485**
		4	B	0.0	0.035	0.455	0.843	0.938	1.0	0.03376
			E	0.0	0.025	0.233	0.608	0.838	1.0	0.03083
		0	B	0.026	0.146	0.603	0.929	0.994	1.0	0.03586
			E	0.034	0.111	0.423	0.746	0.926	1.0	0.03348
	0.4	8	B	0.004	0.087	0.471	0.884	0.982	1.0	0.03477
			E	**0.042**	**0.156**	**0.647**	**0.940**	**1.0**	**1.0**	**0.03621**
		4	B	0.004	0.073	0.478	0.916	0.993	1.0	0.03474
			E	0.007	0.047	0.313	0.729	0.922	1.0	0.03256
		0	B	0.031	0.117	0.566	0.797	0.917	1.0	0.03447
			E	**0.006**	**0.094**	**0.437**	**0.837**	**0.977**	**1.0**	**0.03429**
	0.1	8	B	0.020	0.130	0.455	0.840	0.953	1.0	0.03425
			E	**0.003**	**0.123**	**0.563**	**0.965**	**1.0**	**1.0**	**0.03563**
		4	B	0.008	0.105	0.523	0.895	0.985	1.0	0.03494
			E	**0.0**	**0.063**	**0.475**	**0.908**	**0.993**	**1.0**	**0.03476**
		0	B	0.024	0.091	0.442	0.898	0.964	1.0	0.03447
			E	0.002	0.060	0.273	0.700	0.922	1.0	0.03243
q	κ	l		0.0075	0.01	0.015	0.02	0.025	0.05	

Table 4.3. Statistical distance cumulative percentiles for 1000 peer FreePastry workloads. Note that, unlike in Figures 4.1 and 4.2, there is no column for the relaxation amount γ. Instead, $\gamma = 2$ for all rows.

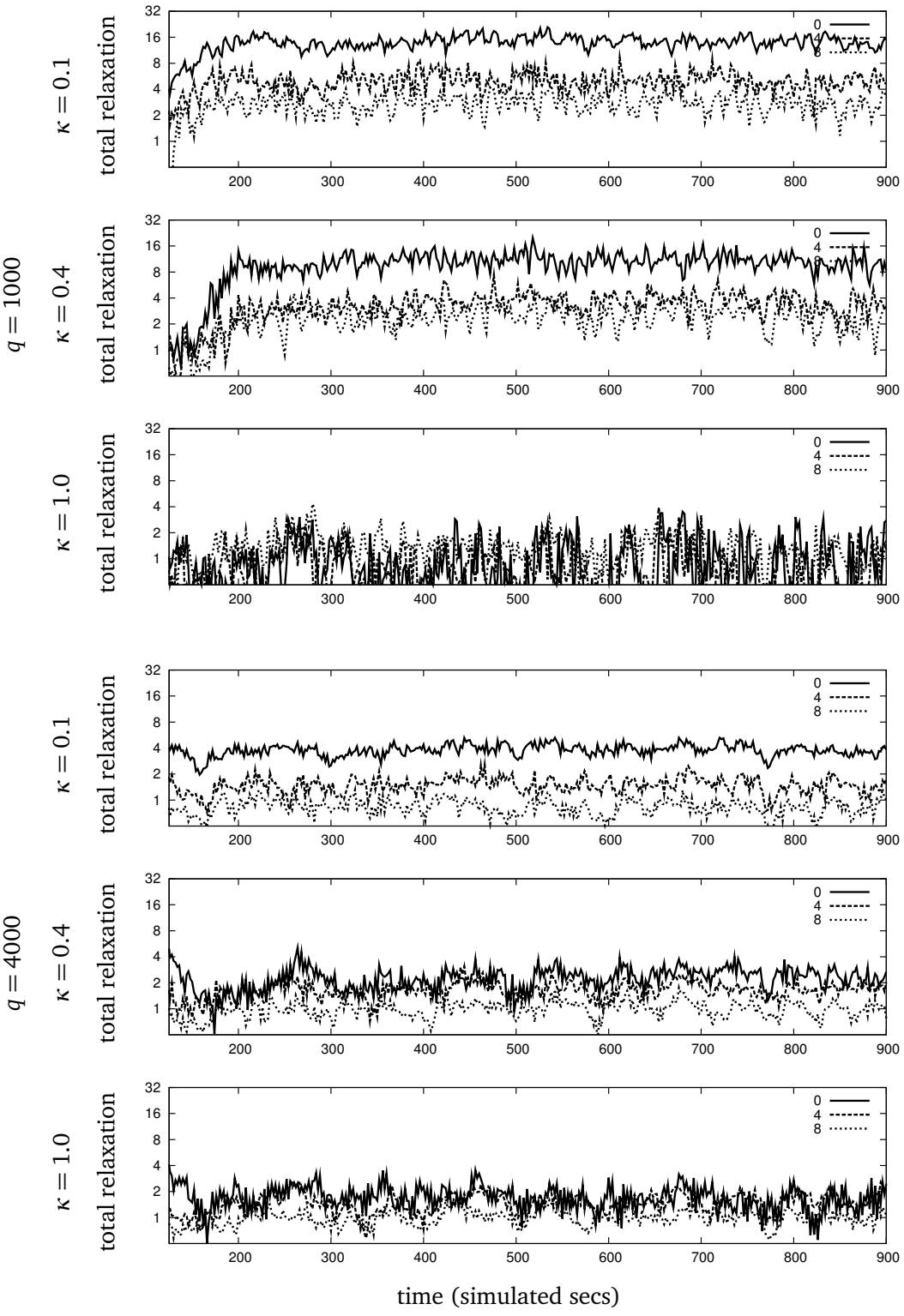


Figure 4.17. Amount of relaxation per round of updates across time for a 1000 peer FreePastry simulations, churn level $c = 1$, $\gamma = 2$, for various values of κ and q. Lines represent different latch times, l.

new incoming probability. As κ decreases to 0.4 and then 0.1, the amount of relaxation first doubles and then doubles again. However, it is clear that using higher latch times significantly reduces this increase, as relaxation is absorbed before starting to converge again. This, in turn, helps to keep DSC well balanced and stable.

Figure 4.18 shows the mixing time for 1000-peer networks for various parameterizations. The expected decrease in λ_2 with smaller values for κ is apparent, particularly with higher latch times. This coincides with the best parameterizations for statistical distance, as seen in the above tables.

Expected Walk Length

We also study the mixing-times in the Pastry experiments, with the same methodology as was used for the static topologies. Figure 4.19(a) displays the relation between the stationary distribution and the walk lengths under varying levels of churn, and shows that DSC achieves good uniformity with short sampling walks. In Figure 4.19(b), 1000-peer Pastry networks show a very similar median walk length as the static topologies presented in Section 4.6.3. The value of λ_2 in Figure 4.18 would suggest that $q = 4000$ would result in a slightly shorter median expected walk length. However, Figure 4.19(b) indicates that a cycle time of 1000 not only produces weightings that converge at the same rate as a cycle time 4000, but because the statistical distance of the resulting weighting is lower, the walks can sample more accurately for the same walk length.

Figure 4.19(c) compares the walk length of 100 and 1000-peer topologies. Both show a comparable walk length to the static results given in Section 4.6.3, with the 100-peer networks significantly out performing the static case. Figure 4.19(c) also appears to show a sub-linear growth of the length of the walks as a function of the size of the network.

4.7 Conclusion

We have presented DSC, a distributed algorithm that balances the transition probabilities of peers in directed peer-to-peer topology so that random walks can uniformly sample the network. DSC quickly converges of topologies with good mixing dynamics, and manages to counteract realistic levels of churn. An enhanced relaxation scheme successfully keeps the statistical distance and mixing time minimized under realistic conditions of churn, resulting in reasonable walk lengths for both 100 and 1000 peer networks. Chapter 6 details the future research directions we see for DSC.

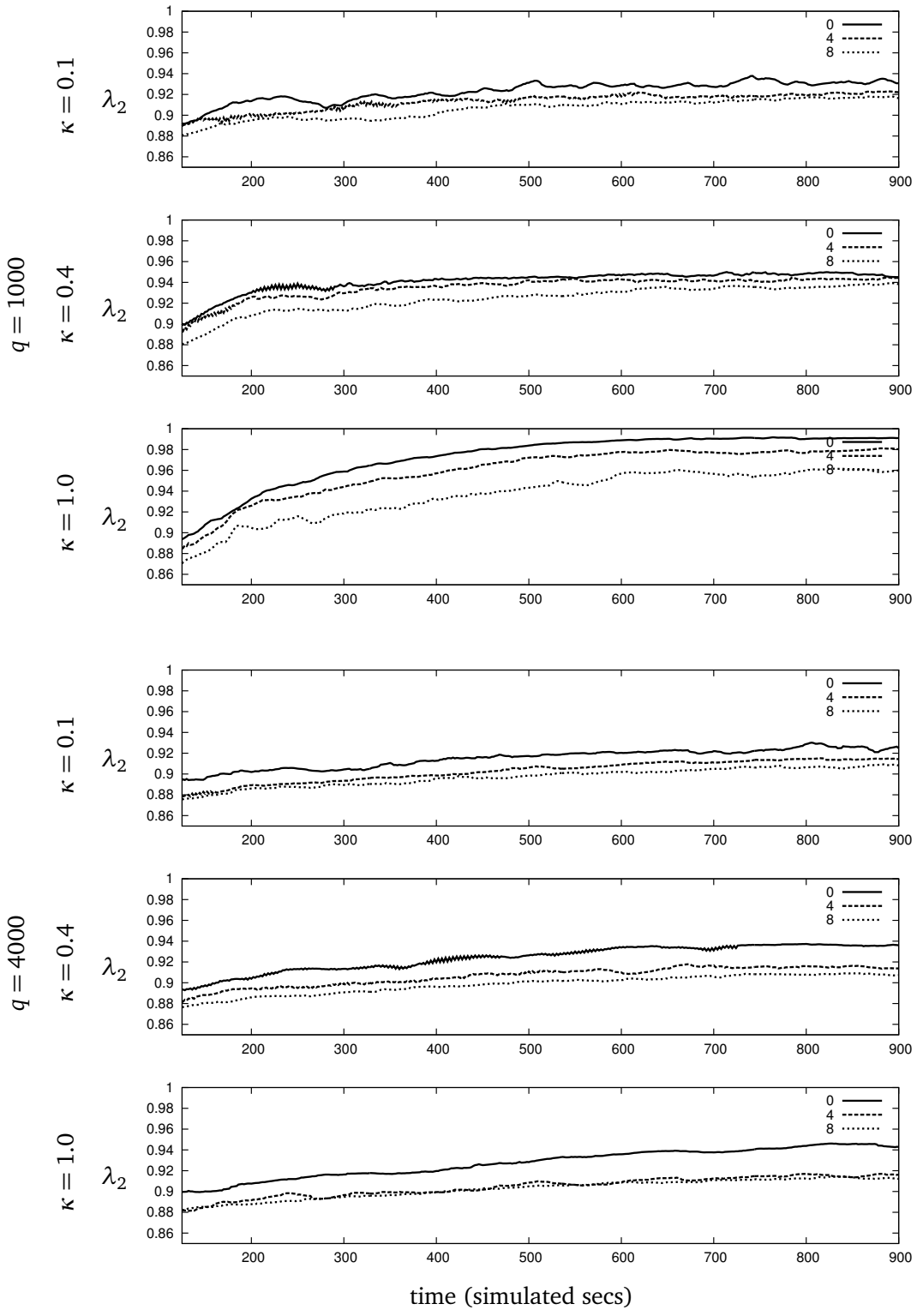


Figure 4.18. Amount of relaxation per round of updates across time for a 1000 peer FreePastry simulations, churn level $c = 1$, $\gamma = 2$, for various values of κ and q. Lines represent different latch times, l.

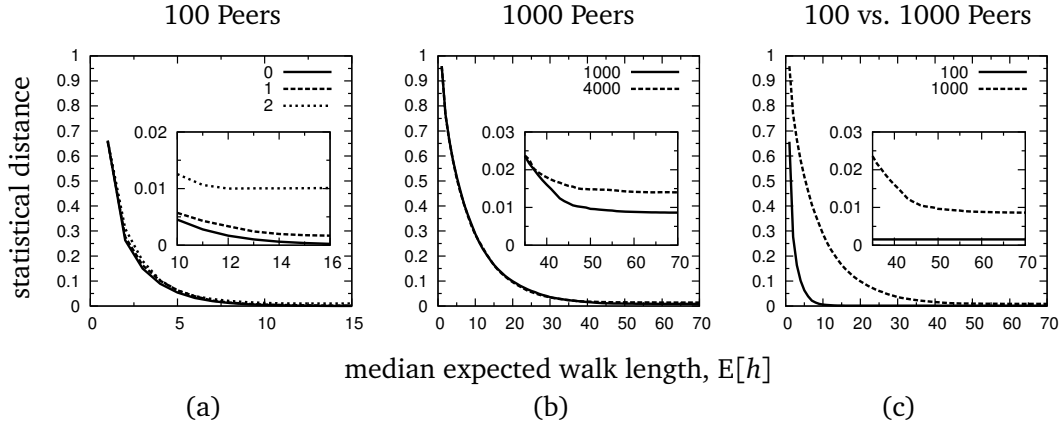


Figure 4.19. Statistical distance and the median expected walk length. (a) Expected hops for churn level 0, 1, and 2, $N = 100$; (b) Expected hops for both $q = 1000$ and $q = 4000$, churn rate 1, and $N = 1000$; (c) Comparison between expected hops for 100 and 1000 peer networks, churn level 1. For all lines, $\kappa = 0.1$, $l = 8$, and $\gamma = 2.0$. For figures (a) and (c), $q = 1000$.

Chapter 5

Distributed Spectral Estimation

Most peer-to-peer systems require a way to accurately estimate system parameters such as network order, diameter, or the necessary length of random-walks. The spectrum of a peer-to-peer network, defined as the eigenvalues of the peer-to-peer graph modeled as a Markov chain, leads to direct estimates or approximation bounds for many network parameters. In this chapter we present a fully distributed algorithm to estimate the dominant eigenvalues in the spectrum of a peer-to-peer network. The algorithm operates in two phases, and extends the Markov-chain model of peer-to-peer networks to treat the network as a linear dynamic system. The first phase amounts to a distributed computation of the impulse response of the linear system. The second phase uses the Ho-Kalman method and the impulse response at each peer to identify a reduced-order system with spectral properties that approximate those of the much larger peer-to-peer network. Since the impulse response forms the core of the algorithm, we call the algorithm *PeerImpulse*.

Our evaluation shows that the estimation is extremely accurate in the absence of churn, and that churn can be handled by truncating the impulse-response traces before performing system identification. Further, we formulate a simple estimate of necessary walk length based off the estimated spectrum, and show that using the distributed estimation algorithm can help to accurately parameterize unstructured search algorithms during network deployment.

5.1 Introduction

Most peer-to-peer network algorithms require *a priori* or *in situ* parametrization in order to perform efficiently. Parameters can be algorithmic constants, such as the size of the ID-space of a distributed hash table, or variables based on the local or global state of a network. Examples of parameters based on global state include the length of random walks, often based on the expected network diameter or an estimate of network size, and the frequency of replica maintenance, often based on measured rates of failure.

The need to estimate parameters based on global information has lead to much research in measurement and information-diffusion techniques. However, some parameters can also be estimated indirectly through other properties of the peer-to-peer network. One such property is the *spectrum* of the network [16]. The spectrum of a peer-to-peer network can be generally defined as the set of eigenvalues and eigenvectors of a matrix associated with the graph representing the network. Typically, the matrix is the adjacency matrix or the Laplacian matrix, or a Markov matrix corresponding to some assignment of transition probabilities for each edge in the graph. The spectrum of a peer-to-peer network can reveal important properties of the network. For example, in the case of the Markov matrix, the first eigenvector gives the probabilities to reach each peer at the end of a sufficiently-long random walk, while the second eigenvalue gives an upper bound on the necessary length of those walks.

The spectrum of a graph can easily be computed using standard techniques from numerical linear algebra, provided the entire matrix is known. However, this is equivalent to having complete knowledge of the peer-to-peer topology, and is therefore undesirable, if at all feasible, for large peer-to-peer systems. In this chapter, we propose a fully distributed algorithm that allows each peer to estimate the dominant components of the spectrum of a peer-to-peer network while maintaining only a local view of the network. We initially developed this algorithm to analyze the Markov chain associated with a peer-to-peer system, and in particular to estimate its second-largest eigenvalue modulus (SLEM). Therefore, in this chapter we describe and evaluate the computation of the SLEM of the Markov matrix of the peer-to-peer network. However, we note that the technique may be generally applicable to more extensive characterizations of the spectrum as well.

The main intuition behind the algorithm is that the graph representing a peer-to-peer network, and interpreted as a Markov chain, can also be interpreted as a large, discrete-time linear dynamic system. The algorithm proceeds by computing a truncated impulse response of this large system, which can be done in a fully decentralized way with a series of small local messages. The truncated impulse response is then used with a variant of the standard Ho-Kalman identification and realization technique to compute, at each peer, a model of a reduced dynamic system that approximates the dynamic behavior of the entire network. The dominant spectral properties of the entire network are then approximated using the dominant spectral properties of the reduced system.

As it turns out, this identification technique is accurate for a large majority of the peers, but not for all. Therefore, the algorithm concludes the estimation with a one-hop gossip/voting round in which each peer exchanges its estimated SLEM with its neighbors, and computes its final estimate by taking the median of all the values seen from itself and its neighbors. Additional improvement is likely with a multi-round gossip process.

Related work is briefly touched on in Section 5.2. We detail the algorithm in Sec-

tion 5.3, and then present initial results and a method to deal with churn in Section 5.4. Section 5.5 concludes the chapter.

5.2 Related Work

Kempe and McSherry compute the top-k eigenvectors of a matrix distributed across a network using a fully distributed version of the traditional orthogonal iteration algorithm [47]. Instead of using QR decomposition to orthonormalize in each round, Kempe and McSherry use Push-Sum [46] to estimate a matrix K related to the system, which is then factored by each peer to find the orthonormalization matrix. The algorithm halts by detecting convergence in a distributed fashion. Since it computes the eigenvectors of a distributed matrix, the algorithm of Kempe and McSherry is clearly more powerful than the PeerImpulse algorithm we propose in this chapter, which only computes the top-k eigenvalues. However, it is also slower, roughly by a factor of at least $\log^2 n$, and significantly more expensive in terms of message traffic.

EigenSpeed and EigenTrust both use distributed power iteration to find the first eigenvector of a matrix [44; 71]. Both EigenSpeed and EigenTrust approach the problem of malicious peers, which we do not address here, but do not analyze their results under churn. While the power method can be used to find more than just the first eigenvector, each additional eigenvector must be run sequentially, making the process slow and expensive compared to our estimation process. However, PeerImpulse does not return a full-rank system, such that the estimated eigenvectors cannot be directly used to estimate stationary distribution. It remains future work to investigate the relation between estimated eigenvectors our algorithm produces and those of the full system.

5.3 Distributed Spectral Estimation with PeerImpulse

As in Chapter 3, we model a peer-to-peer network as a directed graph $G = (V, E)$ with $n = |V|$ vertices. We associate a Markov "transition" matrix P describing a random walk with G.[1] In particular, $P \in [0,1]^{n \times n} = (p_{vu})$ where p_{vu} is the probability of transitioning from u to v in a random walk, therefore $p_{vu} \neq 0 \Rightarrow (u, v) \in E$. Since we are interested in characterizing random walks over the peer-to-peer system, our goal is to compute the dominant components of the spectrum of P. Notice that we assume a completely decentralized computation, which means that P as a whole is not known by any peer. In fact, the problem is precisely to discover useful properties of P without having access to P as a whole.

[1] The matrix P associated with G does not necessarily need to describe a random walk. The algorithm makes no assumptions beyond that the matrix is stochastic and would converge under power iteration. However, as our immediate application deals with the Markov matrix, we will consistently use it in our exposition of the algorithm.

Considering a random walk over the peer-to-peer network, let $x(t) \in [0,1]^n$ be a column vector of probabilities where $x_u(t)$ is the probability that, at step t, the walk visits peer u. (In the absence of qualifications, all vectors should be considered column vectors.) For a random walk starting at peer s, we set $x_s(0) = 1$ and $x_u(0) = 0$ for all other peers $u \neq s$. Then, the probability of a walk being at peer v at time $t+1$ is the sum over all peers u of the probability of the walk being at peer u at time t multiplied by the probability p_{vu} of transitioning from u to v. In essence, a step in the random walk is a linear transformation of x by P with $x(t+1) = Px(t)$.[2]

We reinterpret the Markov-chain model as the model of a linear, deterministic, time-invariant, discrete-time, single-input single-output, dynamic system whose vector of state variables is $x(t)$, and whose state-space model is generically written as

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \qquad\qquad t = 0, 1, \ldots \end{aligned} \tag{5.1}$$

A is the state transformation, $u(t)$ is a scalar input signal, B is a vector that defines how the input signal affects the state, $y(t)$ is a scalar output signal, C is a row vector defining a state-to-output map, and D is a scalar defining an input-to-output map. In our case, A is exactly equal to P, the input and output transformations B and C have a specific role in our algorithm (discussed below), and D is always 0. Apart from the introduction of the input and output signals and transformations, the main difference with the standard Markov-chain model is that $x(t)$ is not restricted to representing probabilities, but is instead allowed to take any value from $\mathbb{R}^n$. On the basis of this model we compute k steps of the *impulse response* of the system, and use it to realize a reduced-order system $\hat{A}$ at each peer. $\hat{A}$ is then used as a surrogate of the original system to compute the dominant spectral components.

We now overview the four high-level steps of this process, discuss their assumptions, and then detail each.

1. *Initialization:* We first construct a suitable input transformation B. Since we are going to compute responses to the unit impulse, which means that $u(0) = 1$ and $u(t > 0) = 0$, the role of B is to define an initial (non-null) state $x(1) = B$. Algorithmically, B is the result of our initialization procedure (Section 5.3.2).

2. *Impulse response:* For k steps, with $t = 1, 2, \ldots, k$, we compute the main state transformation $x(t+1) = Ax(t)$. This is done by having each peer v maintain and update its portion $x_v(t)$ of the state vector. Each peer also records an output trace $y(t) = x_v(t)$ for $t = 1, 2, \ldots k$. This is v's impulse response.

[2] In traditional Markov-chain models, the probability vector x and the transition-probability matrix P are transposed, so x is a row vector, and $P = (p_{uv})$ is such that p_{uv} is the probability of going from u to v, and therefore $x(t+1) = x(t)P$. However, here we adopt the notation used in the dynamic-systems literature, where the state of a system is a *column* vector x, and the dynamic evolution of the system is written as $x(t+1) = Ax(t)$.

Notice that the impulse response is likely to be different for every peer when a graph does not exhibit symmetries. Indeed, each peer v computes the response *of a different system*, that in terms of the general model of Equation 5.1 has state-space equations $x(t+1) = Ax(t) + Bu(t)$; $y(t) = C^v x(t)$, where C^v is a peer-specific output map defined as follows: $C^v_i = 1$ when $i = v$ and $C^v_i = 0$ when $i \neq v$, where i is the index of the vector (Section 5.3.3).

3. *System identification:* Each peer v uses its recorded impulse response $x_v(1), x_v(2), \ldots, x_v(k)$ to realize a reduced system with state-transformation $\hat{A}_v$. In particular, each peer v uses a well-known extension of the Ho-Kalman method [40; 52]. A peer then computes the spectrum of $\hat{A}_v$ locally, using standard techniques (Section 5.3.3).

4. *One-hop vote:* Each peer v exchanges its estimated spectrum with its immediate neighbors. The final estimate for each dominant component of the spectrum at peer v is computed as the median of the corresponding values computed by v and those gathered from the neighbors of v (Section 5.3.4).

5.3.1 Assumptions and Justification

The algorithm described above and detailed below is based on a crucial assumption: at each peer v we can meaningfully approximate the dominant dynamics of a large peer-to-peer system A with a much smaller system $\hat{A}_v$, or at least we can do so for most peers. In other words, we claim that the dominant components of the spectrum of $\hat{A}_v$ are very close to those of A even when $\hat{A}_v$ is realized from a *partial* impulse response of $k \ll n$ steps.

Three assumptions seem to justify the adequacy of $\hat{A}_v$ as an approximation of A: (1) the entire peer-to-peer system is *ergodic*, (2) the peer-to-peer network has low diameter $d < k \ll n$, and (3) most of the peers-specific systems (i.e., $\hat{A}_v$) are observable. Together, these assumptions mean that the state $x_v(t)$ at peer v is affected by every other peer in the network, in the sense that $x_v(t)$ contains pieces of the state $x_u(t-d)$ of every other peer u at a time at most d steps in the past. Since state flows from any peer to any other peer (assumption 1), the impulse response at each peer contains information about the dynamics of the entire network. Second, since the network has a low diameter d (assumption 2), the step at which the impulse response contains information about the entire network is reached quickly. As such, after $k > d$ steps enough information has accrued in the impulse response to perform system identification. Therefore, if most of the estimated systems are observable (assumption 3), the partial impulse-response will contain enough information to allow the realization of a good approximation of A at most peers. The use of a final gossip step will correct those few peers that could not observe important dynamics in the system.

The weakest of the three assumptions is the last. As we discussed in Chapter 4, for a peer-to-peer system to be useful, such that information can flow from any peer to any

other peer, it must be ergodic. This makes the first assumption more or less a given. The second assumption is similarly likely to hold for reasons of performance. A peer-to-peer network with a high network diameter would likely be poorly connected, have poor resilience to churn, and slow search times. Existing peer-to-peer algorithms avoid building high diameter networks for these reasons, and we see no reason to presume that future networks will be different. The final assumption, that most peer-specific systems are observable, is only an intuition, and is not manifestly supported by any common property of peer-to-peer systems that we are aware of. However, importantly, the experimental results presented in Section 5.4 corroborate the assumption.

5.3.2 Initialization

The algorithm we propose computes the response to a unit impulse $u(t) = 1, 0, 0, \ldots$ In practice, this means that an initialization process must define the initial state $x(1) = B$. We implement this initialization so as to satisfy two high-level conditions: first, we want a valid initialization to produce an impulse response usable with the Ho-Kalman realization method; second, we already know that A admits a *stationary distribution* π represented by its first eigenvector, corresponding to an eigenvalue of 1, so we want to create an impulse response that *does not* include the stationary component of A, and therefore *excludes* eigenvalue 1. Together, these two properties mean we want a vector B such that $\lim_{t\to\infty} x(t) = \lim_{t\to\infty} A^t B = 0$.

Since A is a Markov matrix (i.e., non-negative and stochastic) and is also assumed ergodic, Perron-Frobenius theory guarantees that A has one eigenvalue $\lambda_1 = 1$, and that all other eigenvalues are less than 1 in modulus. Let π be the stationary distribution, such that $A\pi = \pi$. A^t will converge to a matrix where all columns equal π, meaning $A^\infty = \pi[1\ 1\ \cdots\ 1]$, where $[1\ 1\ \cdots\ 1]$ is a row vector of n ones. Correspondingly, $x(\infty) = A^\infty B = \pi[1\ 1\ \cdots\ 1]B$. We therefore set $B \neq 0$ such that $[1\ 1\ \cdots\ 1]B = 0$, which makes $x(\infty)$ go to 0.

In practice, the initialization is such that the sum of the initial values $x_v(1)$ over all peers v is 0 and $x(1) \neq 0$ with high probability. This initialization is carried out using a simple, one-round randomized algorithm: each peer v picks a random value r_v, then v chooses one of its neighbors u at random, and sends a message with value $-r_v$ to u; v then sets its initial value $x_v(1) \leftarrow r_v + r_{w_1} + r_{w_2} + \cdots$ where $r_{w_1}, r_{w_2}, \ldots$ are the values received from its neighbors.

5.3.3 Impulse Response and System Identification

After initialization, each peer executes k rounds calculating the impulse response. At round t, peer v holds the value $x_v(t)$. Also, at all times, peer v holds the values A_{iv}, representing the probabilities to transition from v to every other peer i in the network. These probabilities amount to the weights of the outgoing edges of v. For each outgoing edge $(v, w) \in E$ such that $A_{wv} > 0$, peer v sends a message to w containing the value

$(A_{wv})x_v(t)$. For most w, $A_{wv} = 0$, making communication cheap. At the end of round t, each peer v sets its next value $x_v(t+1)$ to the sum of all the values received from its neighbors (and potentially itself) during round t. This process amounts to a distributed computation of the matrix multiplication $x(t+1) \leftarrow Ax(t)$.

Each peer v records the k values of the impulse response, $x_v(1), x_v(2), \ldots, x_v(k)$, and with that time sequence, at the end of the k rounds, realizes a surrogate system with state-transformation $\hat{A}_v$ using Kung's algorithm [52]. Peer v then proceeds to locally compute the spectrum of $\hat{A}_v$ using a standard algorithm such as the eigendecomposition.

In practice, k is a configuration parameter, or can be determined dynamically by calculating the residual ε between the estimated spectrum of $\hat{A}_v$ at round t and $t+1$. When ε drops below a threshold, one can consider the impulse response complete.

5.3.4 Voting Among Immediate Neighbors

The realizations from the partial impulse responses are not always accurate proxies for the entire peer-to-peer network (see Section 5.4.2). Nevertheless, the estimation of the dominant eigenvalue moduli is quite accurate for a large majority of the peers. Therefore, the estimation algorithms finishes with a single round of gossip in which each peer sends its estimated dominant eigenvalue moduli to all its neighbors. For each position in the spectrum, the final estimates are computed as the median of the corresponding moduli computed locally and received from immediate neighbors. Additional rounds can be used to further lower the variance of the estimates.

5.4 Evaluation

We approach the initial evaluation of the algorithm in three phases. First we look at the effectiveness of the estimation in an ideal network (i.e., in the absence of churn). In this case, we test whether and how the initialization algorithm affects the estimates; then we test the accuracy of the estimates as a function of the length k of the impulse response; and then we test whether and how much the one-hop gossip reduces estimate variance.

Second, we consider the effects of churn. In particular, we investigate the behavior of the algorithm when a single failure occurs during the generation of the impulse response.

Finally, we investigate one use of the estimated network spectrum, and estimate the necessary length of a random walk to reach the stationary distribution with a given error tolerance.

5.4.1 Setup

In order to confirm that the algorithm works on a wide range of network topologies, we run it over three diverse types of graphs:

- *Pastry*: Vertices are connected with immediate neighbors in a lattice (the leaf-set). Additional edges emulate the Pastry route table [66].
- *Erdős-Rényi*: Edges are selected independently with a given probability p.
- *Barabási-Albert*: Added vertices are connected preferentially with existing high-degree vertices, creating topologies with power-law degree distributions. We use a preferential attachment factor of $1/2$.

The Pastry graphs are statically calculated, with each graph representing one possible converged Pastry topology for a randomly generated set of node ids. Both the Barabási-Albert and Erdős-Rényi models are described in an excellent survey [2]. Graphs are strongly connected, such that the resulting system is ergodic with high probability. Each graph has (as close as possible) the same number of vertices $n \in \{1000, 10000\}$ and edges $|E| = n \log n$. This is a similar node-edge ratio as to what is found in many deployed peer-to-peer networks.

For each topology we consider two ways to initialize the weights across the edges, which we refer to as *biases*:

- *Degree*: All edges departing from v have the same transition probability $1/d(v)$, where $d(v)$ is v's out-degree. The stationary distribution is non-uniform.
- *Metropolis-Hastings*: An edge between peer u and v has probability $\min(\frac{d(u)}{d(v)}, 1)$. Edges are assumed to be bidirectional, such that $d(v)$ is the degree of v. A self-loop is added at v with transition probability equal to $1 - \sum_{(u,v)\in E} p_{uv}$ [13; 75]. This weighting results in a uniform stationary distribution. See Chapter 3 for more details.

In both cases the probabilities can be calculated through a simple local process. Ten graphs of each type-bias pair are used for the following experiments.

5.4.2 Static Networks

Initialization Error

We first test the stability of the algorithm with respect to the initialization procedure. In particular, we study the effect of 30 different values of B on the estimated second eigenvalue $\hat{\lambda}_2$. To do that, we use long impulse-response traces with fully converged estimations. In Figure 5.1 we plot the difference between the maximum and minimum median estimation error (median over all vertices in a graph) over all initialization vectors, such that each graph is represented as a single point. Each column shows a fairly tight clustering, with no initialization vector significantly worse than any other. While there are bad initialization vectors (e.g., a vector nearly orthogonal to the second eigenvector), they appear to be rare.

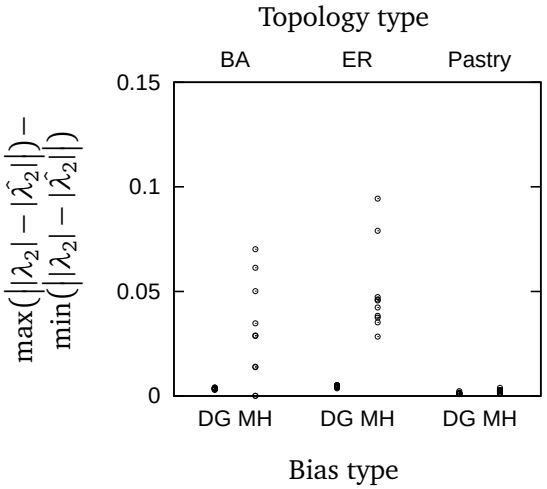


Figure 5.1. Characterization of error caused by choice of initialization vector for degree biased (DG) and Metropolis-Hastings biased biased (MH) graphs of each type: Barabási-Albert (BA); Erdős-Rényi (ER); and Pastry.

Curiously, the degree biased graphs and Pastry networks display much less variation in the resulting estimations based on initialization vector. Intuitively, the degree bias typically results in a fairly low mixing time, however the length of the impulse response was long enough for complete convergence. Another possibility is that the Metropolis-Hastings bias can induce components in the graph with similar behavior, such that some spectral components are not observable at some peers. In particular, since Metropolis-Hastings biases the graph such that the stationary distribution is uniform, it is intuitive that graph components that are connected by only one or two edges can appear to have similar dynamics. The Ho-Kalman algorithm may be unable to distinguish the contributions to the impulse response coming from such "barely connected components" for poorly chosen initialization vectors. Both Erdős-Rényi and Barabási-Albert topologies can exhibit such barely connected components, while our Pastry topologies cannot. Therefore, this hypothesis fits our result nicely, but it remains to be properly tested.

Effect of Impulse Length

Next we look at the effect of the length of the impulse response, k. For each topology, bias, and initialization vector, we estimate the second-largest eigenvalue $\hat{\lambda}_2$ using impulse responses of varying lengths k between 15 and 50. The results are shown in Figure 5.2 and Figure 5.3, respectively for 1000 and 10000 vertex graphs. We plot the estimation error $\left||\lambda_2| - |\hat{\lambda}_2|\right|$ for all three topology types and both biases. Both graphs give the 10%, 50%, and 90% percentile of the estimation error across peers and different topologies. While both Barabási-Albert and Pastry topologies show good convergence and fairly tight variance, Erdős-Rényi topologies do not converge as well for the degree

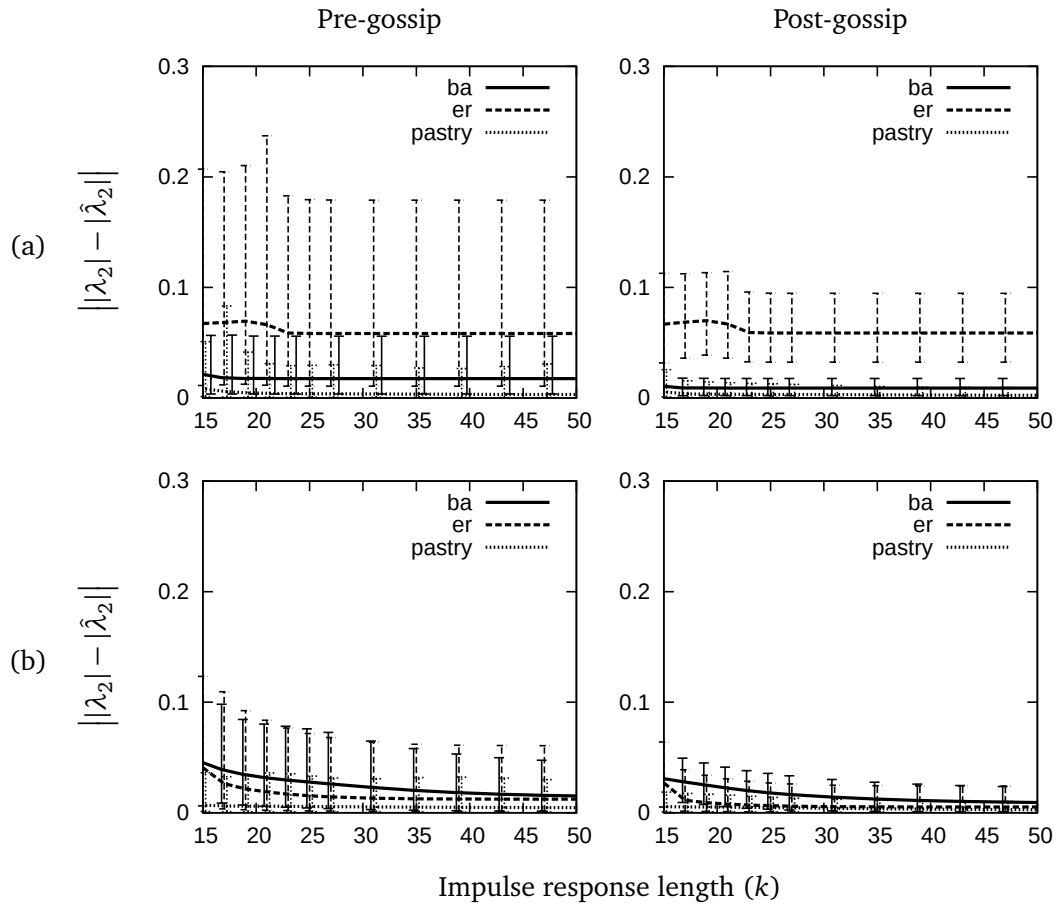


Figure 5.2. Error by trace length, pre- and post-gossip for (a) degree biased graphs and (b) Metropolis-Hastings biased graphs of size 1000.

bias. The results are better for the Metropolis-Hastings bias for all three types of graphs.

Figures 5.2 and 5.3 also contrast the estimation before and after the final one-round gossip step, in the first and second column, respectively. The gossip round reduces the median error and the variance across peers and topologies for all combinations of topology and bias, although this effect is noticeably weaker for degree-biased Erdős-Rényi graphs. The gossip round does not decrease the number of rounds necessary for the estimation to converge, as convergence is purely a function of the length of the impulse response. Nonetheless, the estimation converges relatively quickly. In particular, for all graph types and biases, 20 steps of impulse response are sufficient to find a good estimate of the second eigenvalue. This equates to 20 or less seconds of time for a deployed peer-to-peer network. Degree biased graphs appear to see no benefit after 20 steps. This is mostly likely due to their lower mixing time. The degree biased graphs also scale better from 1000 to 10000 vertices than Metropolis-Hasting biased graphs; again, this

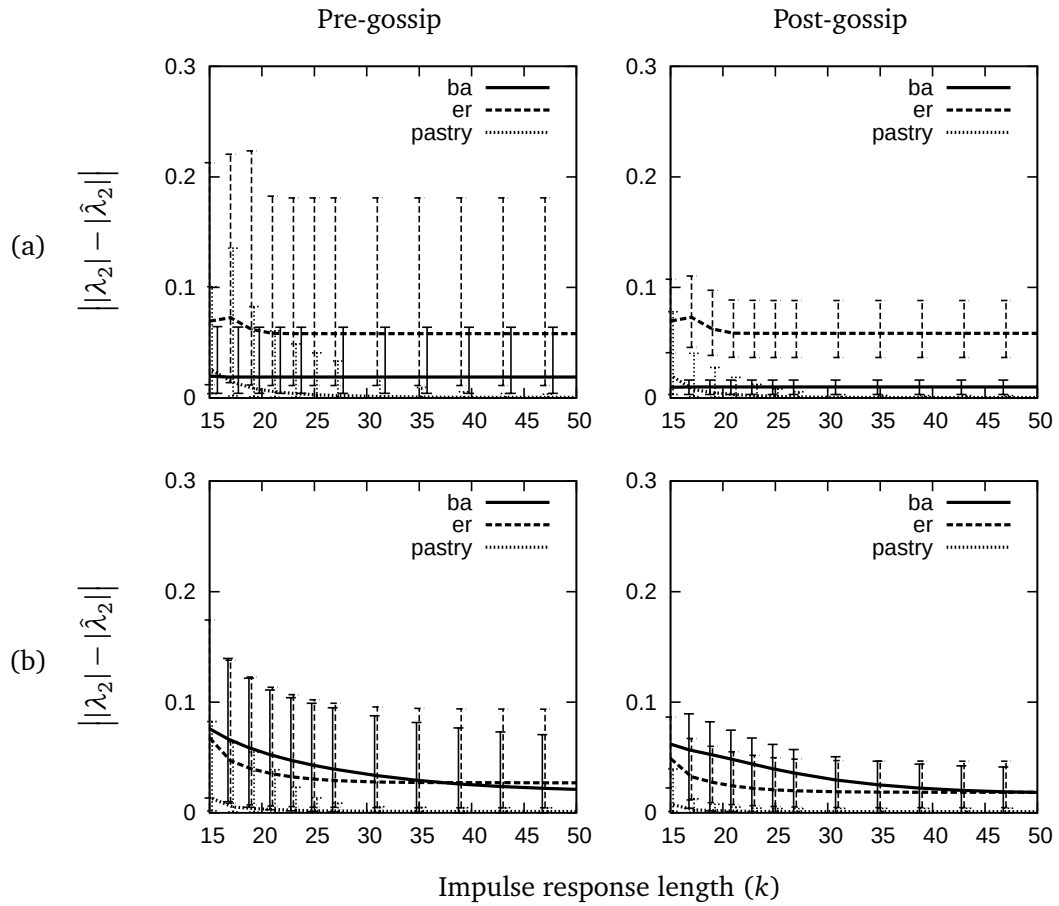


Figure 5.3. Error by trace length, pre- and post-gossip for (a) degree biased graphs and (b) Metropolis-Hastings biased graphs of size 10000.

is due to quicker mixing.

It is not clear why the degree-biased Erdős-Rényi graphs converge with higher error. Figure 5.1 suggests the variance in error induced for this topology-bias pairing is very low, yet the absolute error appears to consistently be the highest. This indicates that most initialization vectors are equally problematic for this topology-bias pairing.

5.4.3 Truncated Impulse Response and Churn

Churn could reasonably be expected to create problems for our estimation scheme. The system-identification techniques we use are intended to work with time-invariant systems. However, churn events (peers joining and leaving the network) clearly make a peer-to-peer network *non time-invariant*, as the structure of the system matrix A changes during the creation of the impulse response. New peers and links can simply be ignored until the termination of the current estimation run. However, peers that leave the net-

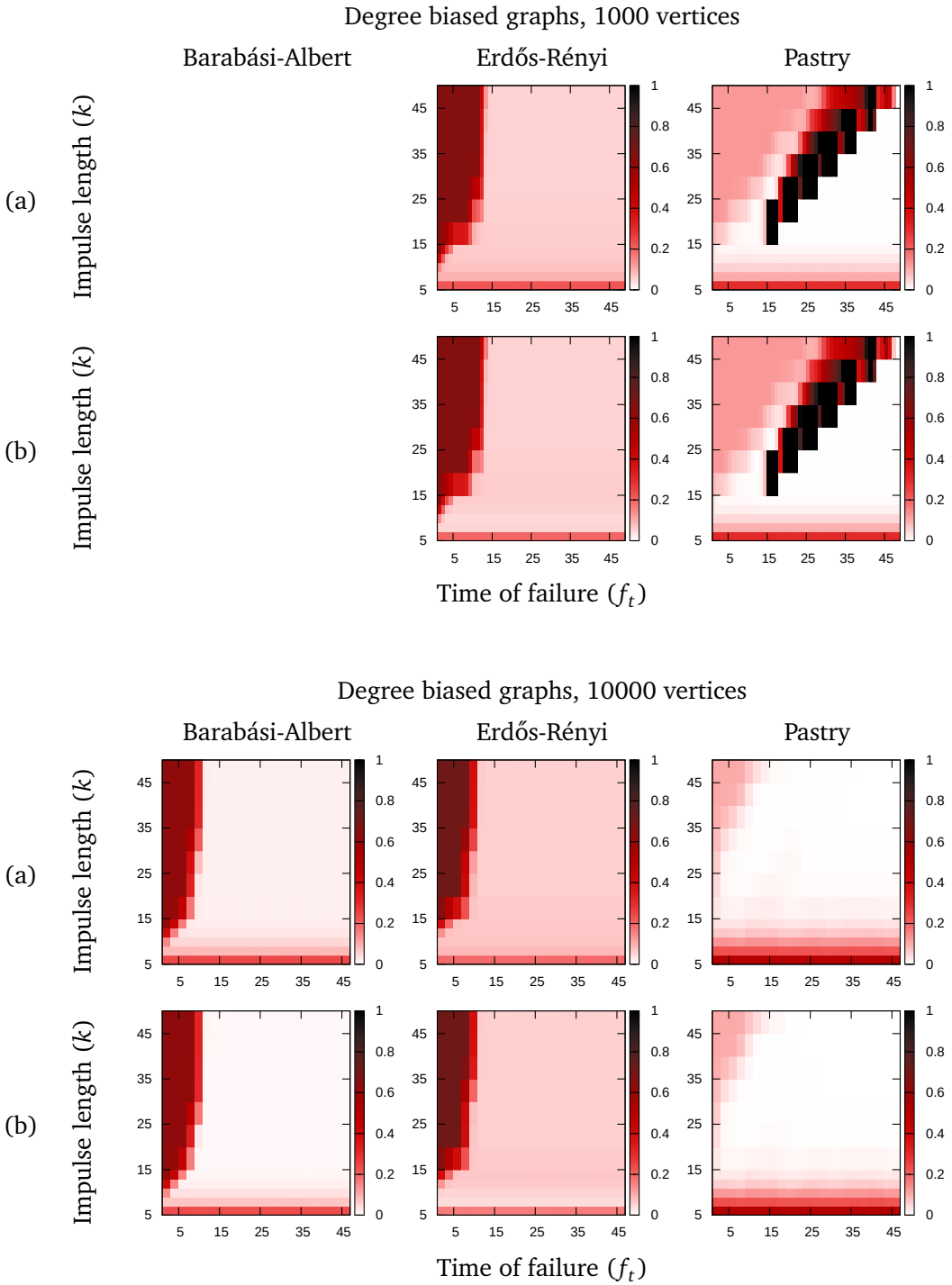
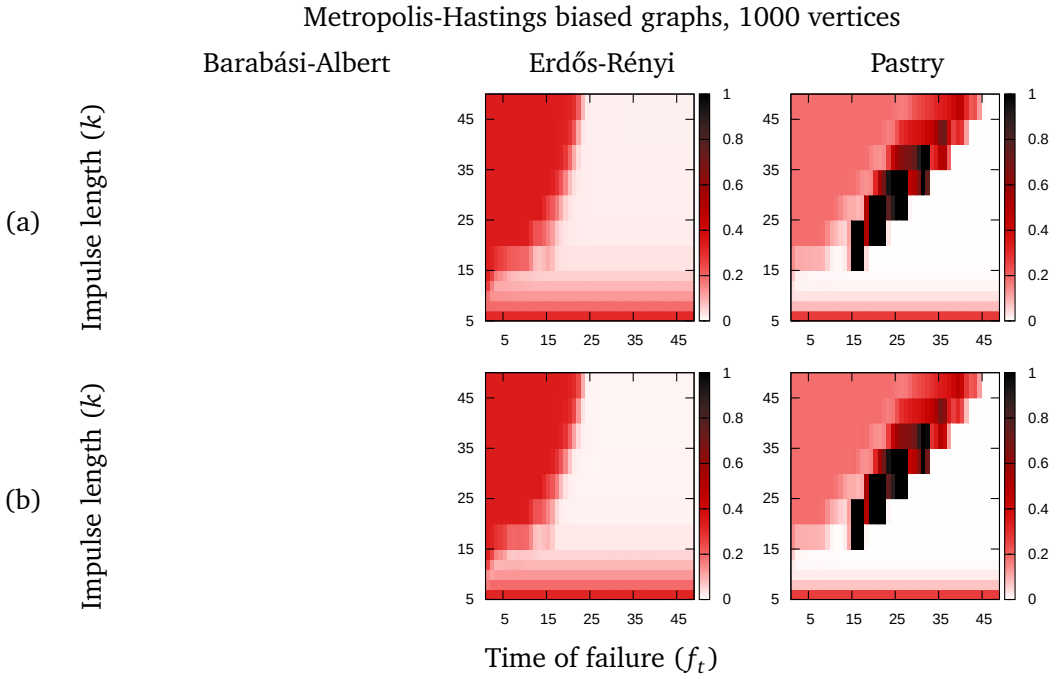


Figure 5.4. (a) Pre-gossip and (b) post-gossip estimation error $\left||\lambda_2| - |\hat{\lambda}_2|\right|$ for varying failure times and trace lengths, for 1000 and 10000 vertex degree biased graphs.



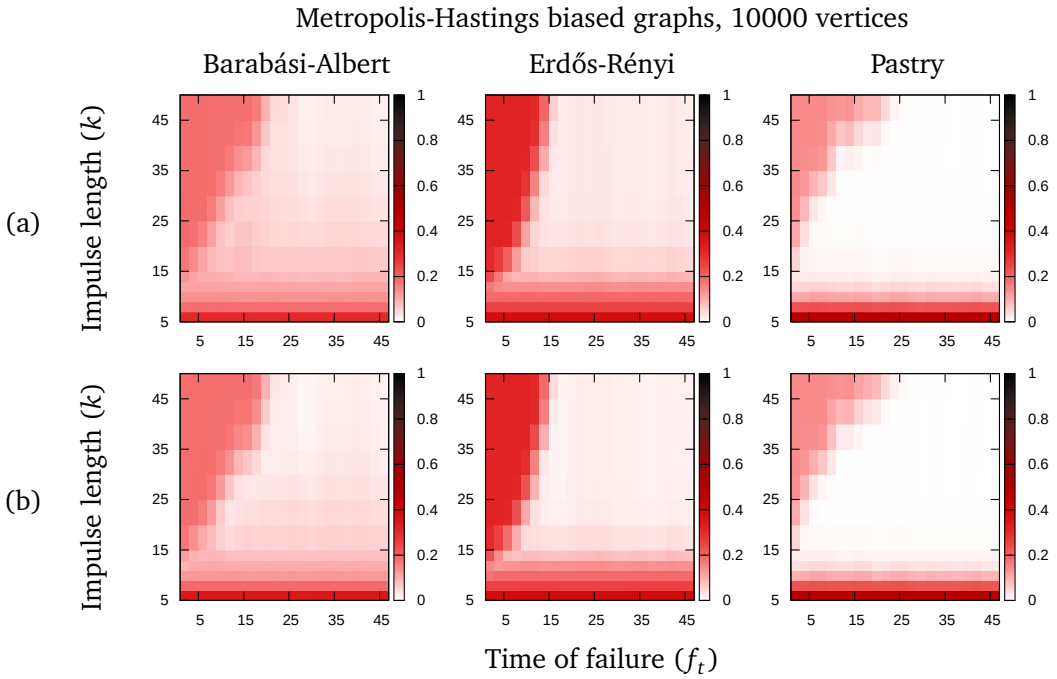


Figure 5.5. (a) Pre-gossip and (b) post-gossip estimation error $\left||\lambda_2| - |\hat{\lambda}_2|\right|$ for varying failure times and trace lengths, for 1000 and 10000 vertex Metropolis-Hastings biased graphs.

work in the middle of the computation of the impulse response cannot be ignored, as they fundamentally change the peer-to-peer topology and the underlying Markov matrix.

Our current implementation attempts to compensate for the departure of a neighbor u of peer v by redistributing u's portion of the state (which can be thought of as a probability mass) to v's other immediate neighbors. However, regardless of the compensation for the change in the state vector, the churn event clearly changes the transition matrix A. In simulations, we induced a single failure at a given time f_t and recorded its effect in term of estimation error. The results of this experiment are shown in Figure 5.4 and Figure 5.5, respectively for degree biased and Metropolis-Hastings biased topologies. The plot shows the error at a given impulse length (y-axis) when a single peer fails at time f_t (x-axis), where the darker cells indicate higher estimation error for the second-largest eigenvalue, $\left||\lambda_2| - |\hat{\lambda}_2|\right|$. 1000 vertex Barabási-Albert graphs are not shown due to missing data.

The results show that, while failure early in the process is quite detrimental for long impulse response trace lengths, early truncation of the impulse response results in good spectral estimation even after churn. This effect is particularly pronounced in the plots for the 1000 vertex Pastry graphs. Individual peers should be able to use the residual ε (see Section 5.3.3) to detect failures and properly truncate their impulse response trace, although evaluating this idea remains future work.

We note that failure toward the end of the impulse response convergence is less problematic. In this regard, the effect of mixing time can once again be seen by comparing the Metropolis-Hastings and degree biased graphs. The area of the plot affected by failure moves further to the right in the Metropolis-Hastings biased plots, as the impulse response takes longer to converge due to higher mixing times, and is therefore vulnerable to churn for a longer period of time. If failure can be detected by use of the residual, accurate estimates can be obtained for the degree biased graphs if failure occurs after the first 5 rounds of impulse generation. For Metropolis-Hastings the necessary number of rounds without failure is more variable: 10 rounds for Barabási-Albert and Erdős-Rényi graphs, and 5 rounds for Pastry graphs.

5.4.4 Walk-length Estimation

In order to substantiate our claim that the largest-magnitude eigenvalues are useful for in situ parametrization, we estimate the Markov spectrum of our biased graphs, and then use $\hat{\lambda}_2$ to estimate the walk length necessary to sample a peer v from the network with probability $\pi_v \pm \epsilon$. For our experiments, we use networks of 10000 vertices and a maximum sampling error of $\epsilon = 10^{-6}$. To estimate the necessary walk length h we find $\hat{\lambda}_2$ and then calculate the estimate $\hat{h} \leftarrow \omega \log_{\hat{\lambda}_2} \epsilon$, where ω is the ratio between the upper bound $\log_{\lambda_2} \epsilon$ and the actual walk-length estimated from one-third of the topologies. The parameter ω is intended to compensate for the pessimistic estimation given by $\log_{\lambda_2} \epsilon$ (see Chapter 3, Section 3.3.1). This ratio ω is fairly stable across different

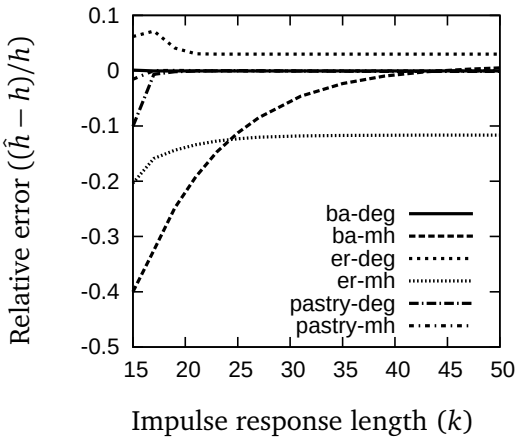


Figure 5.6. Relative error of the estimated hop length.

topologies of the same type/bias, and can be calculated a priori using simulation or from theoretical analysis.

The results are given in Figure 5.6, which shows the median value taken over all vertices and graphs of each type. The figure shows that the estimation works well for its simplicity, with a relative error well below 10% for all topology-bias pairs except Metropolis-Hastings biased Erdős-Rényi graphs.

5.5 Conclusion and Future Work

We have presented a fully distributed algorithm to estimate the largest-magnitude eigenvalues of a peer-to-peer network. Our evaluation shows that the distributed initialization algorithm produces impulse responses usable by Ho-Kalman realization, and that the estimation is quite accurate for the second largest eigenvalue.

This work is preliminary and we continue to work on refining the technique. Other identification and realization models besides Ho-Kalman may give better results, and we plan to try additional models. Another option is to calculate the impulse response to more than a single initialization vector simultaneously. Our initial results have found that nodes with poor observability are sensitive to the initialization vector, so calculating the impulse response for two or three simultaneously should further reduce error. Similarly, using a small cascade of temporally off-set calculations may help to deal with churn.

It is clear that churn invalidates time-invariance, the core assumption of system identification. We would like to look for techniques that partially mitigate this problem. As this problem relates to the essence of the model, a different theoretical model and analysis might provide to be a fruitful approach. We discuss this future work further in the next chapter as we conclude the dissertation.

Chapter 6

Conclusion

In this chapter we conclude the dissertation. First we summarize the thesis and contributions of this work. Then we look at possible future directions of research for both Doubly Stochastic Convergence and PeerImpulse, and peer sampling in general.

6.1 Summary of Work

In this dissertation we have presented two algorithms related to peer sampling in peer-to-peer networks. The first, Doubly Stochastic Convergence, biases the edges in a directed peer-to-peer network such that a random walk uniformly samples peers. The second, PeerImpulse, estimates the largest magnitude eigenvalues of a peer-to-peer network using system identification on a generated impulse response. The estimation of the second largest eigenvalue is then used to estimate the necessary random-walk length.

We have shown, primarily using simulations, that both algorithms are promising. In particular, DSC appears resilient to realistic levels of churn even though it requires multiple rounds to converge. By exploiting bidirectional links when they are available, DSC successfully reduces the number of update rounds necessary by balancing probability advantageously. Our simulations of PeerImpulse demonstrate that good estimations of the largest magnitude spectrum are possible with a truncated and incomplete impulse response, indicating that the methods could prove useful in low-churn environments.

Our algorithms are designed to work under the properties found in wide-area peer-to-peer networks. In particular, we have studied an algorithmic design space where edges between peers can be directed, such that messages may flow in only one direction. We believe this assumption is quite realistic (unfortunately), partly due to the cost and failure rate imposed by network middle-boxes on the creation of bidirectional connections. Moreover, the additional design, implementation, and message costs imposed by the addition of edge tracking on existing networks may be undesirable. We hope the contribution of algorithms that do away with the bidirectional assumption will encourage others to investigate this interesting and largely unexplored space of peer-to-peer

algorithms.

This dissertation is, of course, not exhaustive, either about the algorithms presented, or the areas in general. There are numerous future directions for research on both Doubly Stochastic Convergence and PeerImpulse, but also on peer sampling in general. We now detail some of the more concrete ideas we would like to explore.

6.2 Future Directions

There are several general classes of future work that would be interesting in pursue. Theoretically, peer-to-peer networks are almost always modeled as static graphs, a clearly misleading model in the context of network churn. While there is a growing body of literature on dynamic graph algorithms [18; 51; 79], most work in peer-to-peer networking relies on simulation to verify algorithmic durability under churn. Both DSC and PeerImpulse would benefit if the theory that underlies their correctness could be adapted for dynamic graphs.

Second, both DSC and PeerImpulse would benefit from more real world experimentation. Up to this point, we have relied exclusively on simulation to generate quantitative results in our work, and we firmly believe that this has been the correct decision. However, we feel that DSC is very close to being ready for use in real deployments, and that such a deployment would be beneficial at this point.

6.2.1 Doubly Stochastic Convergence

Some clear directions of future evaluation of Doubly Stochastic Convergence exist. First, scaling up simulations to larger topologies ($n > 10,000$) would allow us to evaluate static convergence in networks closer to the size of many deployed peer-to-peer systems. This is likely to prove difficult without additional infrastructure work, particularly on eigendecomposition. Our initial experiments using MPI and distributed sparse orthonormalization algorithms have proven promising [39].

Another way to approach scalability would be to evaluate DSC in a deployed system, either across PlanetLab, or in a widely used peer-to-peer deployment. We are particularly interested in using infrastructure similar to that developed by David Choffnes for the Ono project [14], a plug-in for bit-torrent now used by close to a million users who have agreed to be monitored for academic purposes. By using centralized tracking, the results from sample walks using the weights generated by DSC could be checked against the known state of the system, allowing the determination of the quality of the biasing and walks. However, the nature of random walks means the level of participation in the network must be high enough that a large, strongly connected component is formed, and it is not clear that the Ono project meets this requirement. Other possible opportunities include Tor or Freenet [17; 25], although the social structure of Freenet makes centralized monitoring impossible.

Improving the mixing time in DSC is clearly crucial for many topology classes. One possible direction to explore to minimize mixing time is the use of distributed approximation of an iterative minimization of mixing time introduced by Boyd et al [9]. Methods like those presented by Awan et al in the Random Weight Distribution algorithm are unlikely to work for DSC, as symmetric trading of probability does not work for non time-reversible graphs [6]. However, techniques to merge DSC and Metropolis-Hastings (which has, in general, lower mixing times) may be possible. We have tentatively explored one such scheme, but it requires targeted information diffusion, such that a peer must request that a specifically named, non-neighboring peer either raise or lower their out-probabilities on a particular edge. More intelligent fusions of the two algorithms may exist.

We would also like to explore ways to reduce the perturbation of the stationary distribution under churn. The goal of such work would be to find graph weightings that produce stationary distributions that change less when edges or vertices are added or removed from the network. Unfortunately, some results in spectral theory indicate that this may be difficult for graphs with low average degree [69], a category which all peer-to-peer networks are likely to fall into, as typically the degree of a peer is much less than the size of the network.

6.2.2 Peer Sampling

More generally, there is a large area still to be explored in peer sampling. One direction we would like to explore is a theoretical analysis of random walks under churn. The commonly used theory of Markov chains in the analysis of random walks can only makes claims about walks on static topologies. Several questions are then in order: to what distribution do walks converge if the underlying topology changes during a walk? If the link biases change during a walk, but the stationary distribution remains virtually the same after each change, does a walk still converge to the stationary distribution? Evaluations random walk schemes can not currently answer any of these questions.

One obvious avenue of exploration are non time-homogeneous Markov chains for modeling a network under churn. Non-homogeneous Markov chains allow the transition probabilities to change over time, possibly as often as the state advances. Typically, such chains do not exhibit a stationary distribution. However, under certain conditions, some non-homogeneous Markov chains do converge to a stationary distribution [33; 80].

Another theoretical area of interest is the study of dynamic graphs, in which vertices and edges are added and removed over the lifetime of the graph [18; 51; 79]. However, the dynamic graph models we are currently familiar with are not appropriate for modeling peer-to-peer networks, as they typically make assumptions about the stability of either the vertex or edge set. This would make the initial challenge the need to develop a new theoretically tractable model that is relevant to churn as it is experienced in peer-to-peer networks.

The only existing sampling algorithm we are aware of that can sample from multiple target distributions is Metropolis-Hastings, which requires bidirectional links. We believe it is likely impossible to design a generic sampling algorithm that can sample from any given target distribution in a graph with directed links, at least without building global topological information. However, there may be other interesting classes of matrices with known stationary distributions that are interesting in the context of peer-to-peer sampling.

Other issues of interest include more extensive comparison between sampling with gossip and sampling with random-walks, and the reduction of the cost of biasing and sampling incurred by DSC and other random walk sampling algorithms via the use of generic aggregation schemes. We have conducted an initial exploration of this second idea and believe that utilizing the latency between update messages in biasing algorithms would give a generic aggregation algorithm the opportunity to aggregate well over half of the maintenance messages associated with link-bias maintenance.

6.2.3 PeerImpulse

Our work on PeerImpulse is more preliminary than the work on DSC. While this means there is more to be done, we also have less understanding of what is possible. However, some major areas of investigation naturally arise.

There is a clear mismatch between the assumption of time-invariance made by Ho-Kalman realization and the dynamic nature of peer-to-peer networks. While our results show that an incomplete, truncated peer impulse often results in a reasonable approximation of the largest magnitude eigenvalues, it is not clear that deployed peer-to-peer systems are stable enough to gather a peer impulse of sufficient length. There are several ways to address this question. First, the introduction of a more realistic churn model in simulation would provide a better analysis of the current algorithms performance under churn. However, it would probably be more productive to step back and try to build a deeper understanding of identification theory. In particular, there appear to be techniques to handle system transition in the middle of identification [41].

It would also be interesting to revisit what little prior work there is in the area, particularly Kempe and McSherry's use of distributed orthonormalization and Push-Sum to compute the complete spectral decomposition [46; 47]. Kempe and McSherry's algorithm has not been analyzed under churn, and we suspect that churn could elongate the time necessary for convergence. As such, another direction of work could be to engineer a version of Kempe and McSherry's algorithm that is hardened for deployment.

Appendix A

Doubly Stochastic Converge Pseudocode

This Appendix gives the pseudocode for the various components of DSC. Figure A.1 summarizes the global variables used by the proceedures. All other variables are local to each procedure. Comments in the pseudocode are right-justified, prefaced with the symbol ▷, and formatted in normal text typeface.

There are two special prodecures that are used in the pseudocode listings, neither of which are themselves listed. The procedure CurrentTime returns the current system time at the calling peer. The procedure Notify$\langle func, s, d, v\rangle$ sends an update message to a remote peer. It takes four arguments: *func*, the procedure which will handle the message; s, the source of the message; d, the destination of the message; and v, the value being passed. For example,

$$\textsc{Notify}\left\langle \textsc{FeedbackHandler}, v, u, p_{uv}(1/(s_t - p_{uu}) - 1)\right\rangle$$

attempts to pass the value $p_{uv}(1/(s_t - p_{uu}) - 1)$ to peer u, where the message will be handled by the FeedbackHandler procedure. The use of $\langle\ \rangle$ instead of $(\ )$ is to indicate that the procedure sends a message and does not wait for a response. Messages sent via Notify are assumed to use a best-effort communication channel, with no assurance of delivery.

The proceedure DscUpdate in Algorithm 1 is called every q seconds, as discussed at the end of Section 4.3.1. All other procedures are either called from DscUpdate or invoked as callbacks when a message is received from another peer.

The pseudocode does not handle many aspects of churn. For example, the pseudocode does not use a timer to invalidate old transition probabilities updates from upstream peers. If an upstream peer were to die, its most recent update would continue to be used indifinitely. For a complete implementation of DSC in Java that addresses such concerns, see the website of the author.[1]

[1] Use Google, as the URI has likely changed since this text was published.

Global variables	
f_{xy}	Feedback value sent to peer x from peer y.
l_v	The amount of latch time remaining at peer v.
p_{xy}	Link transition probability from peer x to peer y.
t_v	The last time self-loop thresholding was applied at peer v.
v	The id of the current peer.
Constant parameters	
α	Exponential feedback multiplier.
γ	Relaxation factor.
κ	Self-loop threshold.
l	Latch time after self-loop relaxation.
t_κ	Minimal amount of time between application of self-loop thresholding.

Table A.1. Table of global variables and parameters.

Algorithm 1 Main DSC Update Routine

```
1:  procedure DscUpdate
2:      s_t ← Σ_{u∈N⁻(v)} p_uv                ▷ Sum incoming probability, include self-loop
3:      sl ← p_vv
4:
5:      if p_vv > κ ∧ CurrentTime() − t_v > t_κ then        ▷ Check self-loop threshold
6:          p_vv ← p_vv/2
7:          t_v ← CurrentTime()
8:          l_v ← l
9:      else if s_t > 1 then                    ▷ Check if relaxation is possible
10:         r ← (s_t − 1.0)γ
11:         if p_vv > r then
12:             p_vv ← p_vv − r
13:         else
14:             p_vv ← 0
15:         end if
16:         l_v ← l
17:     else if s_t < 1 ∧ l_v ≤ 0 then          ▷ Perform normal self-loop update
18:         p_vv ← p_vv + (1 − s_t)
19:     end if
20:     l_v ← l_v − 1
21:
22:     for all u ∈ N⁻(v) \ v do                ▷ Send feedback to upstream peers
23:         if using asymptotic feedback then
24:             Notify ⟨FeedbackHandler, v, u, p_uv(1/s_t − 1)⟩
25:         else
26:             Notify ⟨FeedbackHandler, v, u, p_uv(1/(s_t − p_uu) − 1)⟩
27:         end if
28:     end for
29:
30:     if using standard normalization then    ▷ Normalize out-link probabilities
31:         StraightNormalize()
32:     else
33:         DifferentialNormalize(p_vv − sl)
34:     end if
35: end procedure
```

Algorithm 2 Differential Link-Out Normalization

procedure DIFFERENTIALNORMALIZE(d)
  $s_t \leftarrow \sum_{w \in N^+(v)} p_{vw}$ ▷ Total sum of probability
  $s_u \leftarrow \sum_{w \in \{x \in N^+(v) | f_{vx} < 0\}} f_{vw}$ ▷ Unwanted probability
  $s_w \leftarrow \sum_{w \in \{x \in N^+(v) | f_{vx} > 0\}} f_{vw}$ ▷ Wanted probability
  $s_n \leftarrow 0$ ▷ New sum of probability
  if $d < 0$ **then** ▷ Add difference in self-loop to appropriate sum
    $s_u \leftarrow s_u + |d|$
  else
    $s_w \leftarrow s_w + d$
  end if
  if $s_u \neq 0 \wedge s_w \neq 0$ **then**
    if $s_u < s_w$ **then**
      for all $w \in N^+(v) \setminus v$ **do**
        if $f_{vw} > 0$ **then** ▷ Check if we can fulfill feedback request
          $p_{vw} \leftarrow p_{vw} + f_{vw}(s_u / s_w)$ ▷ Partially apply feedback
        else
          $p_{vw} \leftarrow p_{vw} + f_{vw}$ ▷ Fully apply feedback
        end if
        $s_n \leftarrow s_n + p_{vw}$
      end for
    else ▷ Mirror of $s_u < s_w$ case
      for all $w \in N^+(v) \setminus v$ **do**
        if $f_{vw} < 0$ **then**
          $p_{vw} \leftarrow p_{vw} + f_{vw}(s_w / s_u)$
        else
          $p_{vw} \leftarrow p_{vw} + f_{vw}$
        end if
        $s_n \leftarrow s_n + p_{vw}$
      end for
    end if
  else
    $s_n \leftarrow s_t - p_{vv}$
  end if

  for all $w \in N^+(v) \setminus v$ **do** ▷ Normalize and notify downstream peers
    $p_{vw} \leftarrow p_{vw}\left((1 - p_{vv})/s_n\right)$
    NOTIFY⟨PROBUPDATEHANDLER, v, w, p_{vw}⟩
  end for
end procedure

Algorithm 3 Straight Link-Out Normalization

procedure STRAIGHTNORMALIZE
    $t \leftarrow 1 - p_{vv}$
    $s \leftarrow \sum_{w \in N^+(v) \setminus v} p_{vw} + f_{vw}$
    for all $w \in N^+(v) \setminus v$ **do**
        $p_{vw} \leftarrow (p_{vw} + f_{vw})t/s$
    end for
end procedure

Algorithm 4 Handle feedback from downstream peer; applies exponential smoothing.

procedure FEEDBACKHANDLER(w, v, f)
    $f_{vw} \leftarrow \alpha f + (1 - \alpha) f_{vw}$
end procedure

Algorithm 5 Set local record of transition probability from upstream peer.

procedure PROBUPDATEHANDLER(u, v, p)
    $p_{uv} \leftarrow p$
end procedure

Bibliography

[1] ABERER, K., AND HAUSWIRTH, M. An overview of peer-to-peer information systems. In *Proceedings of the 4th Worksop on Distributed Data and Structures (WDAS)* (2002), pp. 171–188.

[2] ALBERT, R., AND BARABÁSI, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics 74*, 1 (2002), 47–97.

[3] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. The case for resilient overlay networks. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS)* (2001), IEEE Computer Society.

[4] ANDERSON, D. P. BOINC: A system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID)* (2004), IEEE Computer Society, pp. 4–10.

[5] ASPNES, J., AND SHAH, G. Skip graphs. *ACM Transactions on Algorithms 3*, 4 (Nov. 2007).

[6] AWAN, A., FERREIRA, R. A., JAGANNATHAN, S., AND GRAMA, A. Distributed uniform sampling in unstructured peer-to-peer networks. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS)* (2006).

[7] BAKER, D., AND ET. AL. distributed.net, http://www.distributed.net. Online.

[8] BERMAN, A., AND PLEMMONS, R. J. *Nonnegative Matrices in the Mathematical Sciences*. Society for Industrial and Applied Mathematics, 1994.

[9] BOYD, S., DIACONIS, P., AND XIAO, L. Fastest mixing markov chain on a graph. *SIAM Review 46*, 4 (2004), 667–689.

[10] CAMARILLO, G., AND ET. AL. Peer-to-peer (P2P) architecture: Definition, taxonomies, examples, and applicability. *RFC 4844* (2009).

[11] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., AND ROWSTRON, A. Scribe: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on 20*, 8 (Oct. 2002), 1489–1499.

[12] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)* (2006), pp. 205–218.

[13] CHIB, S., AND GREENBERG, E. Understanding the Metropolis–Hastings algorithm. *The American Statistician 49*, 4 (1995), 327–335.

[14] CHOFFNES, D. R., AND BUSTAMANTE, F. E. Taming the torrent: a practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *Proceedings of the 2008 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)* (2008), pp. 363–374.

[15] CHU, Y.-H., RAO, S. G., SESHAN, S., AND ZHANG, H. A case for end system multicast. *IEEE Journal on Selected Areas in Communications 20*, 8 (Oct. 2002).

[16] CHUNG, F. R. K. *Spectral Graph Theory*. American Mathematical Society, 1997.

[17] CLARKE, I., SANDBERG, O., WILEY, B., AND HONG, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability* (2000), pp. 46–66.

[18] CLEMENTI, A. E. F., MACCI, C., MONTI, A., PASQUALE, F., AND SILVESTRI, R. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)* (2008), pp. 213–222.

[19] COHEN, B. Incentives build robustness in bittorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems* (2003).

[20] CROWCROFT, J., AND PRATT, I. Peer to peer: Peering into the future. In *Proceedings of Networking 2* (2002), Springer, pp. 1–19.

[21] DATTA, S., AND KARGUPTA, H. Uniform data sampling from a peer-to-peer network. In *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS)* (2007).

[22] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)* (2004), pp. 137–150.

[23] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)* (2007), pp. 205–220.

[24] DEERING, S. Host extensions for IP multicasting. *RFC 1112 (partially obsoleted)* (1989).

[25] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. F. Tor: The second-generation onion router. In *USENIX Security Symposium* (2004), pp. 303–320.

[26] DRUSCHEL, P., ROWSTRON, A., AND ET. AL. Freepastry, http://freepastry.rice.edu. Online.

[27] FANNING, S. Napster, http://www.napster.com (historical, current site unrelated). Online.

[28] FEATHER, C. D. Network news transfer protocol (NNTP). *RFC 3977* (2006).

[29] FORD, B., SRISURESH, P., AND KEGEL, D. Peer-to-peer communication across network address translators. In *Proceedings of the 2005 USENIX Annual Technical Conference* (2005), pp. 179–192.

[30] GANESH, A. J., KERMARREC, A. M., MERRER, E. L., AND MASSOULIÉ, L. Peer counting and sampling in overlay networks based on random walks. *Distributed Computing 20* (June 2007), 267–278.

[31] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)* (2003), pp. 29–43.

[32] GKANTSIDIS, C., MIHAIL, M., AND SABERI, A. Random walks in peer-to-peer networks. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM)* (2004).

[33] GRIFFEATH, D. Uniform coupling of non-homogeneous markov chains. *Journal of Applied Probability 12*, 4 (1975), 753–762.

[34] GRINSTEAD, C. M., AND SNELL, J. L. *Introduction to Probability*, 2nd ed. American Mathematical Society, 1997.

[35] GUMMADI, P. K., GUMMADI, R., GRIBBLE, S. D., RATNASAMY, S., SHENKER, S., AND STOICA, I. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the 9th Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (2003), pp. 381–394.

[36] HALL, C. P., AND CARZANIGA, A. Doubly stochastic converge: Uniform sampling for directed P2P networks. Tech. rep., University of Lugano, Lugano, Switzerland, 2009.

[37] HALL, C. P., AND CARZANIGA, A. Uniform sampling for directed P2P networks. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing* (2009), pp. 511–522.

[38] HASTINGS, W. K. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57*, 1 (1970), 97–109.

[39] HERNANDEZ, V., ROMAN, J. E., AND VIDAL, V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software 31*, 3 (Sept. 2005), 351–362.

[40] HO, B.-L., AND KALMAN, R. E. Effective construction of linear state-variable models from input/output functions. In *Proceedings of the 3rd Annual Allerton Conference on Circuit and System Theory* (Oct. 1965).

[41] HORENKO, I. On identification of nonstationary factor models and its application to atmospherical data analysis (to appear). *Journal of Atmospheric Science* (2010).

[42] JELASITY, M., AND BABAOGLU, O. T-Man: Gossip-based overlay topology management. In *Proceedings of the 3rd International Workshop on Engineering Self-Organising Applications (ESOA)* (2005), pp. 1–15.

[43] JELASITY, M., VOULGARIS, S., GUERRAOUI, R., KERMARREC, A.-M., AND VAN STEEN, M. Gossip-based peer sampling. *ACM Transactions on Computing Systems 25*, 3 (2007).

[44] KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International World Wide Web Conference (WWW)* (2003).

[45] KARGER, D., LEHMAN, E., LEIGHTON, T., PANIGRAHY, R., LEVINE, M., AND LEWIN, D. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th annual ACM Symposium on Theory of Computing (STOC)* (1997), ACM, pp. 654–663.

[46] KEMPE, D., DOBRA, A., AND GEHRKE, J. Gossip-based computation of aggregate information. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)* (2003), pp. 482–491.

[47] KEMPE, D., AND MCSHERRY, F. A decentralized algorithm for spectral analysis. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)* (June 2004).

[48] KING, V., AND SAIA, J. Choosing a random peer. In *Proceedings of the 23rd Symposium on Principles of Distributed Computing (PODC)* (2004), pp. 125–130.

[49] KLEINBERG, J. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the 32nd Symposium on Theory of Computing (STOC)* (2000), pp. 163–170.

[50] KLENSIN, J. C. Simple mail transfer protocol. *RFC 5321* (2008).

[51] KUHN, F., LYNCH, N., AND OSHMAN, R. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)* (June 2010).

[52] KUNG, S. Y. A new identification and model reduction algorithm via singular value decomposition. In *Proceedings of the 12th Asilomar Conference on Circuits, Systems and Computers* (1978).

[53] LOVÁSZ, L. Random walks on graphs: A survey. *Combinatorics, Paul Erdös is eighty 2* (1993), 1–46.

[54] LV, Q., CAO, P., COHEN, E., LI, K., AND SHENKER, S. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing (ICS)* (2002), pp. 84–95.

[55] MACCLUER, C. R. The many proofs and applications of Perron's theorem. *SIAM Review 42*, 3 (2000), 487–498.

[56] MAGNIEN, C., LATAPY, M., AND HABIB, M. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics 13* (2008).

[57] MAYMOUNKOV, P., AND MAZIÈRES, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-To-Peer Systems (IPTPS)* (2001), pp. 53–65.

[58] METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., AND TELLER, E. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics 21*, 6 (1953), 1087–1092.

[59] MILOJIČIĆ, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. Peer-to-peer computing. Tech. rep., HP Labs, Palo Alto, 2002.

[60] MOCKAPETRIS, P. Domain names - implementation and specification. *RFC 1035 (partially obsoleted)* (1987).

[61] NORRIS, J. *Markov Chains*. Cambridge University Press, 1997.

[62] RASTI, A. H., TORKJAZI, M., REJAIE, R., DUFFIELD, N. G., WILLINGER, W., AND STUTZBACH, D. Respondent-driven sampling for characterizing unstructured overlays. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)* (2009), pp. 2701–2705.

[63] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. A scalable content-addressable network. In *Proceedings of the 7th conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)* (2001), ACM, pp. 161–172.

[64] REYNOLDS, P., AND VAHDAT, A. Efficient peer-to-peer keyword searching. In *Proceedings of the 4th International Conference on Middleware (Middleware)* (2003), Springer-Verlag New York, pp. 21–40.

[65] ROSE, J., HALL, C. P., AND CARZANIGA, A. Spinneret: A log random substrate for P2P networks. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems (Hot-P2P '07)* (2007), IEEE Computer Society.

[66] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 2nd International Middleware Conference (Middleware)* (2001), pp. 329–350.

[67] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systems Journal*, 2 (2003), 170–184.

[68] SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P)* (2001), IEEE Computer Society.

[69] SENETA, E. Perturbation of the stationary distribution measured by ergodicity coefficients. *Advances in Applied Probability 20*, 1 (Mar. 1988), 228–230.

[70] SINCLAIR, A. Improved bounds for mixing rates of marked chains and multicommodity flow. In *Proceedings of the 1st Latin American Symposium on Theoretical Informatics (LATIN)* (1992), pp. 474–487.

[71] SNADER, R., AND BORISOV, N. EigenSpeed: Secure peer-to-peer bandwidth evaluation. In *Proceedings of the 8th International Workshop on Peer-to-Peer Systems (IPTPS)* (2009).

[72] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F. M., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 7th conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)* (2001), ACM, pp. 149–160.

[73] STUTZBACH, D., AND REJAIE, R. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th Internet Measurement Conference (IMC)* (2006), pp. 189–202.

[74] STUTZBACH, D., REJAIE, R., DUFFIELD, N., SEN, S., AND WILLINGER, W. On unbiased sampling for unstructured peer-to-peer networks. In *Proceedings of the 6th Internet Measurement Conference (IMC)* (2006), pp. 27–40.

[75] STUTZBACH, D., REJAIE, R., DUFFIELD, N. G., SEN, S., AND WILLINGER, W. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking 17*, 2 (2009), 377–390.

[76] TERPSTRA, W. W., KANGASHARJU, J., LENG, C., AND BUCHMANN, A. P. Bubblestorm: Resilient, probabilistic, and exhaustive peer-to-peer search. In *Proceedings of the 13th conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)* (2007), ACM, pp. 49–60.

[77] TÖLGYESI, N., AND JELASITY, M. Adaptive peer sampling with newscast. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing* (2009), pp. 523–534.

[78] TRAVERS, J., AND MILGRAM, S. An experimental study of the small world problem. *Sociometry 32*, 4 (1969), 425–443.

[79] VARIOUS. Special issue on dynamic graph algorithms. *Algorithmica 22*, 3 (1998), 233–362.

[80] VASSILIOU, P.-C. G. Stability in a non-homogeneous markov chain model in manpower systems. *Journal of Applied Probability 18*, 4 (1981), 924–930.

[81] VERBEKE, J., NADGIR, N., RUETSCH, G., AND SHARAPOV, I. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. In *Proceedings of the Third International Workshop on Grid Computing (GRID)* (2002), Springer-Verlag, pp. 1–12.

[82] VOULGARIS, S., GAVIDIA, D., AND STEEN, M. V. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management 13*, 2 (2005), 197–21.

[83] WATTS, D. J., AND STROGATZ, S. H. Collective dynamics of 'small-world' networks. *Nature 393* (1998).

[84] XU, Q., SHEN, H. T., DAI, Y., CUI, B., AND ZHOU, X. Achieving effective multiterm queries for fast DHT information retrieval. In *Proceedings of the 9th International Conference on Web Information Systems Engineering (WISE)* (2008), Springer-Verlag, pp. 20–35.

[85] ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications 22* (2004), 41–53.

[86] ZHONG, M., AND SHEN, K. Popularity-biased random walks for peer-to-peer search under the square-root principle. In *Proceedings of the 5th International Workshop on Peer-To-Peer Systems (IPTPS)* (2006).

[87] ZWEIG, K. A., AND ZIMMERMANN, K. Wanderer between the worlds - self-organized network stability in attack and random failure scenarios. In *Proceedings of the 2nd International Conference on Self-Adaptive and Self-Organizing Systems (SASO)* (2008), pp. 309–318.