Abstract of "Probabilistic Techniques in the Analysis of Dynamic Processes" by Aristidis Anagnostopoulos, Ph.D., Brown University, May 2006.

In this thesis we study some stochastic problems related to networks and search engines. We study systems in a dynamic setting where input is continually injected into the system and the algorithm (protocol) processes it. The goal is the design and analysis of protocols that are stable and efficient in the long run.

We first consider the problem of routing calls on telephone or ATM networks, for which we present a routing protocol and we compare it with the one that is commonly used (Dynamic Alternative Routing). We prove that under a standard input model (Poisson arrivals, exponential durations) our protocol has exponentially smaller bandwidth requirement than the traditional approach. Next we study the problem of load balancing on networks, where requests are continually created and serviced. We analyze a protocol under a variety of input models and we prove bounds on the expected load and the expected waiting time of a new request as time passes.

Subsequently, we address a problem related to search engines. We introduce the problem of sampling results from search-engine queries, which has applications in data mining the results and offering services to the user. We present algorithms for the problem and we analyze their running time and the quality of results that we obtain. We supplement the analysis with several experiments.

In modeling of dynamic phenomena as stochastic processes we place some stochastic assumption on the stream of inputs to the system. The stochastic process that generates the stream of inputs might be stationary, periodic, or even bursty. The goal is to obtain results that are valid under the weakest set of assumptions. To this end, we have to develop and apply various mathematical tools. In our analyses we apply tools from Markov processes, queueing theory, renewal theory and martingale or martingale-like processes, that enable us to handle the dependencies between the quantities that appear in our systems.

Probabilistic Techniques in the Analysis of Dynamic Processes

by
Aristidis Anagnostopoulos
Diploma, Computer Engineering & Informatics, University of Patras, 2000
Sc.M., Computer Science, Brown University, 2002
Sc.M., Applied Mathematics, Brown University, 2005

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2006

This dissertation by Aristidis Anagnostopoulos is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ______________________________
Eli Upfal, Director

Recommended to the Graduate Council

Date ______________ ______________________________
Andrei Z. Broder, Reader
Yahoo! Research

Date ______________ ______________________________
Ioannis Kontoyiannis, Reader
Athens University of Economics and Business

Date ______________ ______________________________
Pascal Van Hentenryck, Reader

Approved by the Graduate Council

Date ______________ ______________________________
Sheila Bonde
Dean of the Graduate School

Vita

Name	Aristidis Anagnostopoulos
Born	June 17, 1977, Houston, Texas
Education	University of Patras, Greece Diploma in Computer Engineering and Informatics, June 2000
	Brown University, Providence, Rhode Island, USA Sc.M. in Computer Science, May 2002
	Brown University, Providence, Rhode Island, USA Sc.M. in Applied Mathematics, May 2005
	Brown University, Providence, Rhode Island, USA Ph.D. in Computer Science, May 2006

Acknowledgements

This thesis has been the result of six very fun years at Brown and would have taken much longer to finish, or not have finished at all, had I not interacted with many people.

The first person I would like to thank is my advisor, Professor Eli Upfal, for his guidance and all the patience he showed all those years. In spite of his heavily loaded schedule as the department chair, his office door has always been open for discussions and advice. He has been a continuous source of research problems and he has been constantly coming up with ideas on attacking them whenever I was stuck. I remember numerous times going into his office completely demotivated and coming out excited. Choosing a Ph.D. advisor is one of the most important decisions in one's academic career, and I am very happy that mine was Eli.

Since my first year at Brown I was lucky to meet and start working with Professor Pascal Van Hentenryck. Pascal is the most enthusiastic professor that I have interacted with, full of ideas and with a remarkable intuition on the problems we would work on; at the same time he has a great sense of humor making him one of the most fun professors to work with. I want to thank him for transmitting to me some of his energy for attacking problems.

I was lucky that my arrival at Brown coincided with the arrival of Professor Ioannis Kontoyiannis at the Division of Applied Mathematics. Yiannis and I had several common research interests and as a result we ended up discussing several problems together. This interaction strengthened my mathematical intuition significantly and increased my appreciation for rigorous mathematical approaches to the extent that I decided to pursue a Masters in his department. I am grateful for that.

I spent the summers of 2004 and 2005 at internships at the IBM T. J. Watson research center under the supervision of Dr. Andrei Broder. Andrei was an extremely busy man, that was obvious. Nevertheless, I was meeting with him in a daily basis and in a very short amount of time he was able to convey to me his perspective in looking at problems, that there is an increased value in attacking a problem when implications extend beyond a purely academic level, and that an approach to tackle a problem is especially valuable when it leads to solutions that are not only theoretically sound but practically efficient as well.

At IBM I was also offered a lot of help by several researchers; I would especially like to thank David Carmel, Steve Gates, Kunal Punera, and Wilfried Teiken for their help in designing and implementing many algorithms.

Three of my professors at Brown have helped me since my first year by providing both technical and nontechnical advice. Discussions with Professors Uğur Çetintemel, John Savage, and Roberto Tamassia, affected many of my decisions during my studies. Especially Roberto, being the first professor at Brown that I worked closely with, exposed me to the process of discovering and solving research problems.

Throughout all the years at Brown I was very lucky to have the help and support of many friends and colleagues. There are many many people that I would like to thank, but I cannot omit the following: Yanif Ahmad, Russ Bent, Eleftheria Bourtzinakou, Sebastien Chan Tin, Kleopatra Christoforou, Niko Daniilidi, Fernando De la Torre, Paschali Karageorgi, Adam Kirsch, Liz Marai, Olga Papaemmanouil, Christo Peraki, Mano Renieri, Massimo Santini, Selim Tezel, Sri Tirthapura, Niko Triandopoule, Yianni Vergade, Michali Vlacho, Joel Young, and Song Zhang, without your help I would still be struggling with my work. Many thanks!

One of my best experiences at Brown has been the exposure to a broad range of classes. The following instructors introduced me to a wide range of topics and taught me much more than the class material: Miguel Bota, Costas Dafermos, Paul Dupuis, Stu Geman, Basilis Gidas, Mike Goodrich, Thomas Hoffman, Claire Kenyon, Yiannis Kontoyiannis, Shriram Krishnamurthi, Dimitris Livanios, Catalina Ocampo, Franco Preparata, Meinolf Sellman, Daniel Spirn, Roberto Tamassia, Eli Upfal, Andy Van Dam, and Pascal Van Hentenryck.

I also want to thank the administrative and technical staff at the department, who every single time that I needed their help provided it very fast.

Finally, my most thanks go to my dad, mom, and my brothers, Babi and Dimitri, for all their love and support all these years. I wouldn't have been able to finish without all the personal encouragement throughout my studies, and thanks a lot for that!

Contents

Chapter 1

Introduction

In this thesis we study some stochastic problems related to networks and to search engines.

Many of the processes that occur in computing systems are dynamic: input is continually injected into the system and the algorithm (protocol) processes it. Traditionally, theoretical computer science analyzes algorithms in a static setting where the algorithm accepts an input, it processes it, and it terminates with an answer. In contrast, in dynamic systems the protocol does not terminate, but instead it continually processes the input as it is being generated by making irrevocable decisions. The goal is the design of protocols that are stable and efficient in the long run. There are many examples of such dynamic network systems: the evolution of the World Wide Web, traffic in routing network, processing in distributed systems, contention resolution protocols, and so on.

Rigorous analysis of such processes seems to be necessary to characterize input conditions under which a given system behaves as desired. One approach is to model the systems according to a probabilistic model, as stochastic processes. This is one of the most popular ways for analyzing similar dynamic processes in other scientific fields. For example, Brownian motion and generalizations are used in economics to model the fluctuation in stock prices. Diffusion and branching processes are applied in biology, while the Poisson process models the arrival of customers in a queue, or the emission of radioactive particles.

In a large part of this thesis we analyze some networking systems where stochastic processes serve as natural models for their evolution. We believe that the experience gained from the aforementioned related fields is useful in understanding dynamic network processes so that we can design practical and efficient schemes for the associated problems.

Subsequently, we addresses a problem related to search engines. Web search continues its explosive growth and thus search algorithmic efficiency is as important as ever: although processor speeds are increasing and hardware is getting less expensive every day, the size of the corpus and the number of searches is growing at an even faster pace. Therefore, new tools are required to assist the user in his search process.

We introduce the problem of sampling results from search-engine queries, which has applications in data mining the results and offering services to the user. We present algorithms for the problem and we analyze their running time and the quality of results that we obtain.

1.1 System Stability

The primary performance measure for the networking systems that we analyze is their *stability.* The issue of stability arises in many different areas in modeling real-world systems. Hence, it does not have a unique formal definition—and this is especially true in the world of stochastic processes—but instead, depending on the context, it takes different technical interpretations, which try to capture the intuitive idea of a system being "stable."

It seems that the issue of stability first appeared in the analysis of problems encountered in mechanics, and the issue of whether a solution of a deterministic dynamical system that is modeled as an ordinary or partial differential equation is stable, is one of the major concerns. In that context, a solution is stable if small perturbations in the problem parameters or in the problem input do not create large changes to the solution, and there are various definitions that formalize this concept.

A very important contribution to the theory of stability was made in 1892 by A.M. Lyapunov in his Ph.D. thesis entitled "The General Problem of Motion Stability," who, firstly, provided a rigorous definition of motion stability, and, secondly, introduced a method for proving that a solution is stable, now referred to as the *Lyapunov-function method.* The main power of the method lies in the fact that it is fairly versatile and easy to apply, as one can prove that a solution to a differential equation is stable without having to explicitly solve the equation.

The ideas of Lyapunov for proving stability were carried to the world of stochastic processes, first, probably, by Foster [30], and subsequently by Bertram and Sarachik [16] and Kats and Krasovskii [44]. An early exposition of the method appears in Kushner's book [46]. Theorem 2.2.2 in this thesis is an example of the application of the method in Markov processes, while similar in spirit is Theorem 3.4.4, for more general stochastic processes.

As an important example, in the area of Markov chains there are various ways in which a system can be considered stable. A nice discussion appears in the book written by Meyn and Tweedie [55]. At a first level, a Markov chain can be considered stable if it is irreducible: no matter how the system starts executing it can reach any other state. A stronger notion of stability is that of recurrence: there is a guarantee that for every state, the return time is finite; and that of positive recurrence: the expected return time is finite.

Yet another notion of stability is that concerning the limiting or *ergodic* behavior. It turns out that for a positive recurrent Markov chain, there exists a stationary regime (distribution), π, such that if the chain starts in this regime it stays there, and if it starts in some other regime it converges to π in a probabilistic sense. Then we can study conditions that guarantee the convergence of empirical measures such as

$$\frac{1}{T}\sum_{t=0}^{T-1} f(X_t)$$

to $\mathbf{E}_\pi[f]$, and as a further step, we can study the speed of convergence.

All the aforementioned notions of stability, which are desired for the systems that we analyze, are not precisely what we are trying to achieve. Consider for example a queueing system. One idea of stability that we envisage is that the load of the queue remains bounded as time passes. In the case of a stochastic analysis, this criterion can be transformed to the expected load under the stationary regime being bounded, or to the maximum expected load over all time steps being bounded. This is, for example, the notion considered in previous work regarding backoff protocols for contention resolution [36, 41], queueing networks [8, 17, 22], and load-balancing protocols [15]. Even more, we would like those bounds to be (small) polynomial functions of the system size. Furthermore, we might like to have bounded higher moments of the load, or high-probability results.

For a finite system the requirement of stability translates to the avoidance of overloading the system, and since in a (nontrivial) infinite-time stochastic process this is inevitable, we desire that disasters take place infrequently and that whenever those disasters happen, the system manages to recover fast. (Recovering could mean taking a simple action such as dropping a packet and waiting for the sender to resend, or, in more complicated systems, performing a more sophisticated action.)

Returning to the queuing-system example, we may desire further properties. In particular we would like a bound on the time that a packet has to wait in the queue before it becomes served. A very useful tool is the famous Little's law [71], $L = \lambda W$, which can transform a bound on the expected load to a bound on the expected waiting time. If, however, we want to provide a bound on the expected waiting time of a given packet (as opposed to bounding the average expected waiting time over all packets, which is the quantity that Little's law involves) we need to perform more detailed analysis. Further analysis is also necessary when we want to obtain stronger guarantees than just bounding the expectation.

One point we should mention, is that the two different notions of stability are not unrelated. For instance, assume that we model our system as a Markov chain, for which we show that there exists a stationary distribution π and that the expected load under the stationary distribution is bounded. Then, Theorem 14.0.1 in [55] implies that as time passes the expected load remains bounded. This close relationship is one of the reasons that the authors in [37] use positive recurrence as the definition of stability, while all the protocols considered in [41] are either stable or unstable with respect to both stability notions.

In any case, the idea of stability that we consider is the one that we discussed previously, the boundedness of the system load and the waiting time for service as the system evolves, in expectation and with high probability. The statements of our theorems quantify in precise technical terms this notion of stability.

1.2 Adversaries in Modeling

One of the important decisions when modeling a system's evolution is the type of the input in the system. Traditionally, in computer science analyses of algorithms, the input is controlled by an adversary and the goal is to design algorithms that have good worst-case performance. In the case of the analysis of a dynamic system, *worst-case analysis* rarely gives any insight to the efficiency of a protocol. A worst-case adversary is too powerful, and can flood the system by creating extremely hard sequences, which hardly occur in practice, therefore the performance of a protocol on such sequences does not reflect its actual performance.

An alternative approach, common in other scientific fields, is the application of *probabilistic analysis*. Here the input is not controlled by an all-powerful adversary, but instead, it is a sample from a prespecified distribution or from a distribution selected from a family of distributions. In this case the measure of efficiency is the performance of the protocol under some probabilistic measure, for example in expectation or with high probability. In the analysis of dynamic systems like those that we analyze in this thesis, the probabilistic assumption is more realistic than in the traditional (in computer science) static analysis, for the input is defined by requests that are generated by a lot of independent (or weakly dependent) processors, and, therefore, infinitely divisible distributions, such as the Poisson distribution, provide good approximations to their ensemble. In Chapter 2 our input model is defined by a set of Poisson processes, a very realistic and commonly employed model in systems like the one that we analyze.

The main criticism against probabilistic analysis is that it is very specific. Often the assumption for the input is driven by the desire to perform detailed analysis of the process, therefore it may be only a very simplistic model of the "real world." As we mentioned, it turns out that these types of assumptions are fairly realistic in the problems that we analyze and experimental evidence justifies them, while efficient performance under those assumptions is an indication of a well-designed system. Nevertheless, it is desirable to obtain results that are valid in big generality, under the weakest set of assumptions. This is the route that we follow in Chapter 3. There, the input is controlled by an adversary who has significant power, for example, he has full information about the system and his decision may depend on the entire history of the process. However, several restrictions are necessary. A necessary and rather obvious restriction is that in the long run the expected rate with which new load is generated, must be lower than the rate with which existing load is consumed by the system when operating in full capacity. This assumption still allows for large bursts, but we need to keep them under control; this is formalized by placing some further bounds on higher moments of the input rate, and the more restrictions that we place, the stronger are the results that we obtain.

1.3 Overview

We study two problems related to network processes and a problem on search engines.

In Chapter 2 we consider the problem of routing calls on circuit-switching telecommunication systems. Examples of such networks are telephone and Asynchronous-Transfer–Mode (ATM) networks. We present a routing protocol and we compare it with the one that is commonly used. We show that under a standard input model (Poisson arrivals, exponential durations) our protocol offers significantly (asymptotically) higher performance than the traditional approach.

In Chapter 3 we study the problem of load balancing on networks. We consider a network model, suitable for networks where job processing is the dominant operation (as opposed to routing), we present a very simple distributed randomized protocol for distributing work to the processors through the network, and analyze the stability of the system, as jobs are created and as jobs are being consumed. In particular we show that the maximum expected load in the long run is bounded by a quantity that depends on the network structure (the expansion of the network), which is always a small polynomial in the network size. We also give similar results for the waiting time of a job in the system before being consumed. Furthermore, we show that the system has efficient behavior not only in expectation, but also with high probability.

In Chapter 4 we diverge and we consider a search-engine related problem. Consider a search engine and a given query which when issued to the search engine it returns a total of m results (documents). The problem that we study is how to sample efficiently a random subset of k documents from the result set, where $k \ll m$. We present the motivation for studying the problem and we introduce a general mechanism for sampling efficiently any query. We then present a specialization that performs additional optimizations and increases the performance for a class of queries. We justify our findings analytically and experimentally. Next we present and analyze a second method for sampling, faster from a theoretical viewpoint but probably less efficient in real system deployment.

1.4 Mathematical Tools

For the analysis of the problems that we study, we need to develop and apply certain tools from mathematics. In the network systems under consideration we focus on the asymptotic behavior and so every valid system state will occur infinitely often. Therefore the analysis must ensure that the system rarely enters undesirable states and quickly recovers from them when it does. Simple probabilistic techniques (e.g., estimating the probability of rare events for every time point and using a union bound over all time steps) are not sufficient to achieve these goals. We require more sophisticated arguments.

One of our goals is to demonstrate the application of different ideas in modeling and analyzing dynamic processes. An important one is the modeling of the system evolution as a Markov chain. In Chapter 2 we model our system as a countable-space, continuous-time Markov chain (or Markov process). We first adapt a Lyapunov-type drift criterion to this type of Markov chain (Theorem 2.2.2), which establishes the existence of a unique stationary regime and the convergence to it at an exponential rate. Subsequently, we combine queueing-theoretic results and combinatorial arguments to analyze the transient behavior of the chain, which exhibits the system's behavior under the stationary regime.

A *martingale* [60] is a stochastic process that generalizes Markov chains. It is frequently used in computer science to obtain tight concentration results through the Azuma inequality. Another fundamental theorem of martingales, less commonly employed in the computer-science literature, is the optional sampling theorem, which can be applied to calculate values for expectations. We use it in Chapter 4 to prove that an estimator has the correct expectation.

Often we can use a static algorithm and adapt it in order to create a dynamic protocol (e.g., [22]). This is the approach that we follow in Chapter 3. We adapt a protocol that has been proven to perform efficiently in a static setting, and then we apply it in a dynamic one. Here the input is controlled by an adversary whose decisions may depend on the entire history of the process, therefore in this case we cannot model the system as a Markov chain. Instead, we first consider a simple but powerful adversary and we analyze the system using results from renewal theory. This enables us to translate the efficient static performance to an efficient steady-state performance. Once again, we have to combine known results from renewal theory with combinatorial arguments.

Subsequently, we consider a broader class of adversaries. In this case the analysis follows a different path, namely we use results that analyze stochastic processes with behavior similar to supermartingales. Once again, we combine the efficient static performance, with additional combinatorial arguments and obtain results that show that the system has efficient performance in the long run, both in expectation and with high probability.

Chapter 2

Balanced-Allocation Routing

Fast, high bandwidth, circuit switching telecommunications systems such as Asynchronous Transfer Mode (ATM) and telephone networks often employ a limited path-selection algorithm in order to fully utilize the network resources while minimizing routing overhead. Typically, between each pair of nodes in the network there is a dedicated bandwidth for communication; namely, no more than a certain fixed number of calls can be simultaneously active between each pair of nodes. This dedicated bandwidth is chosen in order to satisfy the demand for communication between these stations. Only when this bandwidth is exhausted does the admission control protocol try to find an alternative route through intermediate nodes. To minimize overhead and routing delays, the protocol checks just a small number of alternative routes; if there are no free connections available on any of these alternatives, then the call or communication request is rejected. Implementations that use this technique include the Dynamic Alternate Routing (DAR) algorithm used by British Telecom [34], and AT&T's Dynamic Nonhierarchical Routing (DNHR) algorithm [10].

A common feature in these (and other) currently implemented protocols is the sequential examination of alternative routes. Only when the algorithm examines a route and finds it cannot be used is an alternative one examined. The criteria for when a route can or should be used, and the method in which the alternative route is selected have been the subject of extensive research, in particular, in the context of British Telecom's DAR algorithm [33, 34, 42]; see Kelly [45] for an extensive survey.

Dynamic routing can be viewed as a special case of the online load balancing problem, where the load (incoming calls or requests) may be assigned to one or more servers (network links), and jobs (communication requests) can be scheduled only on specific subsets (paths) of the set of servers, as defined by the network topology. In this chapter we study the impact of replacing the sequential searches of the routing algorithm by a version of the *balanced-allocation principle*. The basic idea is as follows: Instead of sequentially choosing alternative options (in our case, paths) until a desirable one is found, in the balanced-allocation regime the algorithm randomly chooses and examines a number of possible options, and assigns the job at hand to the option that appears to be the best at the time of the assignment.

A number of papers have demonstrated the advantage of the application of the balanced-allocation principle [11, 12, 21, 57, 58] for standard load balancing problems, where jobs require only one server and can be executed by any server in the system. This research has shown that balanced allocations usually produce a very substantial improvement in performance, at the cost of a small increase in overhead: Since several alternatives are examined even when the first alternative would have been satisfactory, the complexity of the routing algorithm is increased. But, as has been shown before and as we also demonstrate in the present context, examining even a very small number of alternative (thus increasing overhead by a very small amount) can offer great performance improvements.

The idea of employing the balanced-allocation principle to the problem of dynamic network routing as described here was first explored in [49]. In this context the goal is to reduce system congestion and minimize the blocking probability, that is, the probability that a call request is rejected. The main difficulty in applying and analyzing the balanced-allocation principle in a network setting is in handling the dependencies imposed by the topology of the network. The preliminary results in [49] show that the advantage of balanced allocations is so significant that it holds even in the presence of a set of dependencies.

The performance of a routing protocol can be analyzed in a static (finite, discrete time) or in a dynamic (infinite, continuous time) setting. The static case has been extensively studied in [48], extending and strengthening the results in [49]. In this chapter we consider the continuous-time case. The analysis of the continuous-time case suggested in [49] was based on applying Kurtz's density-dependent jump Markov chain technique, following the supermarket model analysis in [57, 58]. However, since the argument in [49] is incomplete, we present here a different analysis. Our results concern the long-term behavior of large networks employing a routing protocol based on the balanced-allocation principle. The main tools we employ are a Lyapunov drift criterion used to establish the existence of a stationary distribution for the BDAR routing protocol, and a continuous-time extension of the technique in [12], used to analyze the stationary behavior of a network.

Balanced allocations have also been studied in the context of *queueing* networks, where analogous results (under different asymptotic regimes than the ones in this chapter) are obtained in [51, 57, 73, 76], among others.

The results of this chapter have appeared in [7], and are joint work with Ioannis Kontoyiannis and Eli Upfal.

2.1 Model Description and Main Results

In the types of networks that we consider here, a logical link or "bandwidth" is reserved between each pair of stations, and an alternative route is only used when this logical link has already been exhausted. We model such a network as the complete undirected graph $G = (V, E)$ with $|V| = n$ vertices (stations) and $|E| = N = \binom{n}{2}$ edges (links).

The input to the system is a sequence of call requests, which are assumed to arrive at Poisson times: New calls onto each link (i.e., between each pair of nodes) arrive according to a Poisson process with rate λ, all arrival streams being independent. Similarly, the duration of a call is independent of all arrival times all other call durations, and it is exponentially distributed with mean $1/\mu$.

The routing algorithm has to process the calls on-line, that is, the tth request is either assigned a path or rejected before the algorithm receives the $(t+1)$th request. Once a call is assigned to a path, that path cannot be changed throughout the duration of the call. We assume that each edge has a capacity of $3B$ circuits (one circuit can transmit one call), where $1/3$ of this capacity is reserved for direct calls (namely, it will only be used for call requests between these two nodes), and the rest is reserved for being used as part of an alternative route between two stations.

As in most of our results we consider large networks with a number n of nodes growing to infinity, we will also assume that the capacity parameter B may vary with n. Specifically, we assume that $B = B_n$ is nondecreasing in n, and we also allow the possibility $B = \infty$.

The goal in designing an efficient routing protocol is to assign routes to the maximum possible number of call requests without violating the capacity constraints on the edges. We will compare the performance of the following two protocols:

The *d-Dynamic Alternative Routing (DAR) algorithm* works as follows. When a new call request arrives, it tries to route the call through the direct (one-link) path. If there are no available circuits on the direct path, then the algorithm sequentially chooses alternative routes of length two, without replacement, and assigns the call to the first available path. Up to d such choices are made, and they are made at random. If no possible path is found, then the request is rejected.

The *d-Balanced Dynamic Alternative Routing (BDAR) algorithm* also assigns a new call request to the direct path if there are available circuits. If not, then the algorithm chooses d length-two alternative paths at random, with replacement, and compares the maximum load among them (in the exact sense that we describe later). Then the call is assigned to the path with the minimum load. As before, if there is no path with free circuits among these d choices, then the call is rejected.

Consider some link e between two stations u and v, with a capacity of $3B$ circuits, from which B are reserved for routing calls between u and v. The rest of the $2B$ circuits, which are reserved for alternative paths, are further split into two. B circuits are reserved for routing calls with u as one of the endpoint station communicating, and B circuits for calls with v as the endpoint.

The model described so far, together with one of the two protocols above, induces a continuous-time stochastic process describing the behavior of the network. As we show below, this system (for fixed n) converges to a stationary regime exponentially fast. For our purposes, the main performance measure is the minimum required bandwidth that ensures that, under the stationary distribution of the network, the blocking probability (i.e., the probability that a new call is rejected) is appropriately small.

Our main goal is to compare the performance of the DAR algorithm with that of BDAR. It is clear that BDAR's performance is dominated by its performance on alternative (length-two) routes. Therefore, to simplify the analysis, we consider a variant of BDAR, which we call BDAR*, which ignores the direct links and services each call only via an alternative route, making use only of the $2B$

alternative connections of each edge. In other words, we assume that each edge has capacity $2B$ and all of it is dedicated to alternative routes. We show that even though the BDAR* policy ignores the direct links, it has superior performance compared to DAR.

The following result illustrates this superiority by exhibiting explicit asymptotic bounds on their bandwidth requirements. It follows from the results in Theorems 2.2.4 and 2.3.1.

Theorem 2.1.1. *Assume that all the edges have a capacity of $3B$ circuits.*

Under the DAR policy, capacity

$$B = \Omega\left(\sqrt{\frac{\ln n}{d \ln\ln n}}\right), \qquad \text{as } n \to \infty$$

is necessary to ensure that, under the stationary distribution, a new call is not lost with high probability.

On the other hand if we perform the BDAR policy (thus ignoring the B direct links), capacity*

$$B = \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right), \qquad \text{as } n \to \infty$$

suffices to ensure that, under the stationary distribution, a new call is not lost with high probability.

In the above result and throughout the thesis, we say that a limiting statement holds "with high probability" (abbreviated "whp.") if it holds with probability that is at least $1 - 1/n^c$ for some constant $c > 0$. For example, when we say that a random variable "$X_n = O(\ln n)$ whp." we mean that there are positive constants C and c such that $\mathbf{Pr}(X_n \leq C \ln n) \geq 1 - 1/n^c$ for all n large enough. Similarly, "$X_n = o(\ln n)$ whp." means that there is a $c > 0$ such that, for all $\epsilon > 0$, $\mathbf{Pr}(X_n \leq \epsilon \ln n) \geq 1 - 1/n^c$ for all n large enough.

Note that the result of Theorem 2.1.1 is exactly analogous to that obtained in [48] in the discrete-time case.

2.2 Analysis of Balanced-Allocation Routing

This section presents our main contribution in this chapter, a steady state analysis of the performance of the BDAR* routing algorithm. The network is a complete graph with n nodes and $N = \binom{n}{2}$ undirected edges. New calls arrive at Poisson times with rate λ and their durations are exponentially distributed with mean $1/\mu$, as described earlier. As it turns out, an important parameter in the analysis of the network load is the ratio $\rho = \lambda/\mu$.

2.2.1 Unbounded capacities

We first analyze the maximum load on edges when the algorithm is used on a network with unbounded edge capacity, corresponding to $B = B_n = \infty$. Consider some ordering of the edges, and let

$$\Gamma = \{(e, e') : e, e' \in E,\ e < e',\ e \text{ adjacent to } e'\},$$

be the set of edge pairs that are adjacent to each other. For every pair of adjacent edges $(e, e') \in \Gamma$, let $c_{e,e'}(t)$ denote the number of calls at time t that use edges e and e' (recall that every alternate path consists of two links). Then the above model induces a continuous-time Markov process $\mathbf{\Phi} = \{\Phi(t) \,:\, t \geq 0\}$, evolving on the state space

$$\Sigma = \mathbb{N}^{N(n-2)},$$

where

$$\Phi(t) = (c_{e,e'}(t))_{(e,e') \in \Gamma}.$$

For an edge $e = (u, v)$ we define also $\ell_{e,v}(t)$ to be the number of calls at time t that use edge e and have node v as an endpoint:

$$\ell_{e,v}(t) = \sum_{\substack{e' : \ (e',e) \in \Gamma,\, v \text{ not} \\ \text{adjacent to } e'}} c_{e',e}(t) + \sum_{\substack{e' : \ (e,e') \in \Gamma,\, v \text{ not} \\ \text{adjacent to } e'}} c_{e,e'}(t),$$

and we also define $\ell_e(t)$ to be its combined load at time t, that is,

$$\begin{aligned} \ell_e(t) &= \ell_{e,v}(t) + \ell_{e,u}(t) \\ &= \sum_{e' : (e',e) \in \Gamma} c_{e',e}(t) + \sum_{e' : (e,e') \in \Gamma} c_{e,e'}(t). \end{aligned}$$

Assume that a call arrives at time t on edge $e = (u, v)$. Algorithm BDAR* selects d nodes uniformly at random with replacement, from $V \backslash \{u, v\}$. Name these nodes $\{w_i\}$ for $i = 1, 2, \ldots, d$, and the corresponding edges $e_i^u = (u, w_i)$ and $e_i^v = (w_i, v)$. The call is then assigned to the path (e_i^u, e_i^v) corresponding to the minimum i satisfying

$$\max\{\ell_{e_i^u,u}(t-), \ell_{e_i^v,v}(t-)\} = \min_{j=1,2,\ldots,d} \max\{\ell_{e_j^u,u}(t-), \ell_{e_j^v,v}(t-)\}.$$

In the above expression, and throughout the entire chapter, $f(t-)$ denotes the left-side limit of function f at t, namely, $\lim_{\delta \downarrow 0} f(t - \delta)$. Note that instead of selecting the minimum i satisfying the above expression, we can choose any Markovian rule. Finally, we define

$$\begin{aligned} M_{\geq i}^v(t) &= \sum_{e : e \text{ incident to } v} (\ell_{e,v}(t) - i + 1)^+ \\ L_{\geq i}^v(t) &= \sum_{e : e \text{ incident to } v} \mathbf{1}_{\{\ell_{e,v}(t) \geq i\}}, \end{aligned}$$

where $\mathbf{1}_{\mathcal{E}}$ denotes the indicator function of event $\mathcal{E}$, and $x^+ = \max\{x, 0\}$. In words, $L_{\geq i}^v(t)$ counts the number of edges incident to node v with at least i calls with v as an endpoint at time t, and $M_{\geq i}^v(t)$ counts the excess above i at time t on edges incident to v, of calls that have node v as an endpoint. Trivially we have $L_{\geq i}^v(t) \leq M_{\geq i}^v(t)$.

As we show next, this Markov process has a stationary distribution π_n to which it converges exponentially fast, regardless of the initial state of the network. We then prove a high probability bound on the maximum load on any edge in the system under this stationary distribution.

The process $\mathbf{\Phi}$ evolves on Σ according to the model described above. This evolution is formalized by the transition semigroup $\{P^t : t \geq 0\}$ of $\mathbf{\Phi}$, where $P^t(\boldsymbol{c}, \boldsymbol{c}')$ is simply the probability that $\mathbf{\Phi}$ is in state $\boldsymbol{c}'$ at time t given that it was in state $\boldsymbol{c}$ at time zero, $P^t(\boldsymbol{c}, \boldsymbol{c}') = \mathbf{Pr}(\Phi(t) = \boldsymbol{c}' \mid \Phi(0) = \boldsymbol{c})$.

Our first result shows that $\mathbf{\Phi}$ has a stationary (or invariant) distribution to which it converges exponentially fast. It is stated in terms of the "Lyapunov function" $V(x)$, which is defined as 1+(total number of active calls in state $x \in \Sigma$):

$$V(x) = V(\{c_{e,e'} : (e, e') \in \Gamma\}) = 1 + \sum_{(e,e')\in\Gamma} c_{e,e'}, \tag{2.1}$$

where $c_{e,e'}$ counts the number of calls in state x that use edges e and e'.

Theorem 2.2.1. *Assume that the BDAR* algorithm is used on a network with n nodes, each of which has infinite capacity. Then the induced Markov process $\mathbf{\Phi}$ has a unique invariant distribution π_n, and, moreover, for any initial state $x \in \Sigma$, the distribution of $\Phi(t)$ converges to π_n exponentially fast, namely, there is a constant $\gamma < 1$, such that*

$$\sup_y |P^t(x, y) - \pi_n(y)| \leq V(x)\gamma^t, \ \textit{ for all } t \geq 0 \textit{ and all } x \in \Sigma.$$

Proof. Our proof uses the Lyapunov drift criterion for the exponential ergodicity of a continuous time Markov process [53, 26, 54]. To state our main tool, we recall a few definitions, adapted to our case of countable state space.

The *generator* $\mathcal{A}$ of the process $\mathbf{\Phi}$ is a linear operator on functions $F : \Sigma \to \mathbb{R}$ defined by

$$\mathcal{A}F(x) = \lim_{h\downarrow 0} \frac{\mathbf{E}(F(\Phi(h)) \mid \Phi(0) = x) - F(x)}{h}$$

whenever the above limit exists for all $x \in \Sigma$. The *explosion time* of $\mathbf{\Phi}$ is defined as

$$\zeta = \sup_n J_n,$$

where

$$J_0 = 0, \qquad J_{n+1} = \inf\{t \geq J_n : \Phi(t) \neq \Phi(J_n)\}$$

($J_0, J_1, \ldots$ are the jump times of the Markov process). We say $\mathbf{\Phi}$ is *nonexplosive* if $\mathbf{Pr}(\zeta = \infty \mid \Phi(0) = x) = 1$ for any starting state x.

The following theorem follows from the more general results in [54, 26], specialized to the case of a continuous-time Markov process with a countable state space.

Theorem 2.2.2. [54, 26] *Suppose a Markov process evolving on a countable state space that is nonexplosive, irreducible (with respect to the counting measure on Σ) and aperiodic. If there exists a finite set $C \subset \Sigma$, constants $b < \infty$, $\beta > 0$ and a function $V : \Sigma \to [1, \infty)$, such that,*

$$\mathcal{A}V(x) \leq -\beta V(x) + b\mathbf{1}_C(x) \qquad x \in \Sigma\,, \tag{2.2}$$

then the process is positive recurrent with some invariant probability measure π, and there exist constants $\gamma < 1$, $D < \infty$ such that

$$\sup_y |P^t(x, y) - \pi(y)| \leq D\,V(x)\gamma^t, \ \textit{ for all } t \geq 0 \textit{ and all } x \in \Sigma.$$

It is easy to verify that the process is ψ-irreducible and aperiodic, with the maximal aperiodicity measure ψ being the counting measure on Σ.[1] Also the process is nonexplosive since the number of new calls in a given interval has a Poisson distribution with a finite mean; therefore, the probability of infinite number of transitions in a finite interval is 0.

To show that the drift criterion (2.2) can be satisfied, we use the Lyapunov function $V(x)$=1+(total number of active calls in state x) defined in Equation (2.1) above.

In order to compute $\mathcal{A}V$ we notice that when a new call enters the system, it increases the loads of two edges by 1, hence the value of V by 1, and when a call terminates the value of V decreases by 1. Therefore, new calls are generated with rate λN and calls are terminated at a rate $\mu(V(x)-1)$. The probability that in a time interval h there are 2 or more new calls or terminations of calls is $o(h)$.[2] Using these observations we can compute $\mathcal{A}V$:

$$\begin{aligned}\mathcal{A}V(x) &= \lim_{h\downarrow 0}\frac{V(x)+\lambda N\cdot h-\mu\cdot(V(x)-1)\cdot h+o(h)-V(x)}{h}\\ &= \lambda N-\mu V(x)+\mu.\end{aligned}$$

We define

$$C=\left\{x\in\Sigma : V(x)<\frac{2\lambda}{\mu}N+2\right\},$$

which is clearly finite, and in order to analyze the drift condition we distinguish between the following two cases:

- $x \in C$:
$$\mathcal{A}V(x)=\lambda N-\mu V(x)+\mu\le -\frac{\mu V(x)}{2}+\lambda N+\mu$$
- $x\in\Sigma\backslash C$:
$$\mathcal{A}V(x)=\lambda N-\mu V(x)+\mu\le\frac{\mu V(x)}{2}-\mu V(x)=-\frac{\mu V(x)}{2}.$$

Thus, the drift condition holds for $\beta=\mu/2$ and $b=\lambda N+\mu$. □

Having shown the existence of an invariant limiting distribution π_n, we now analyze the maximum load on the edges under this distribution.

Theorem 2.2.3. *Consider a network with n nodes, and let π_n be the invariant distribution of the induced Markov process under the BDAR* policy with unbounded edge capacity. Under π_n, the maximum number of calls in any edge is bounded whp. by*

$$\frac{2\ln\ln n}{\ln d}+o\left(\frac{\ln\ln n}{\ln d}\right)\qquad\textit{as } n\to\infty.$$

[1] This follows along the lines of the arguments in Chapters 4 and 5 of [55]. In particular, note that all sets $\{y\}\in\Sigma$ are ν_1-small and $P^1(x,y)>0$ for all $x,y\in\Sigma$ so that in fact $\mathbf{\Phi}$ is irreducible and strongly aperiodic.

[2] Here and in the next expression with the notation $o(h)$ we mean that f is $o(h)$ if $\lim_{h\to 0}\frac{f(h)}{h}=0$. In the rest of the text $o(n)$ has the usual meaning.

Proof. In order to compute the maximum edge load under the stationary distribution, we start observing the system at some time point and study its transient behavior; we then use the results to deduce the properties of the invariant distribution. In particular, we show that there exists a $T = O\left(n\frac{\ln\ln n}{\ln d}\right)$, such that for any state of the system at time $\tau - T$ that has sufficiently large probability (we will be more precise later), whp. at time τ the maximum number of calls on any edge is

$$\frac{2\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right).$$

The high level idea is the following. We partition the time period T into $\frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right)$ periods of length $O(n)$. Roughly, we argue that at the end of the ith period, whp., for each node, the number of incident edges with load greater than i is at most $2\alpha_i$. The α_i's decrease doubly exponentially, so at the end of the last period we will be able to deduce that there are no edges with more than $\frac{\ln\ln n}{\ln d}$ load towards each direction, whp. The challenge is to handle the dependencies, as the number of calls during some period depends on the number of calls of the previous periods. We now proceed with the details.

We first define the sequence of values $\{\alpha_i\}$, which decrease doubly exponentially:

$$\alpha_\kappa = \frac{(n-2)\rho}{\kappa} \qquad \text{where } \kappa = e\rho \cdot \sqrt[d-1]{2\rho \cdot 4^d}$$

$$\alpha_i = \frac{2\rho \cdot 4^d \cdot \alpha_{i-1}^d}{(n-2)^{d-1}} \qquad \text{for } i > \kappa \text{ and } \alpha_{i-1} \geq \frac{1}{4} \cdot \sqrt[d]{\frac{25}{\rho}(n-2)^{d-1} \cdot \ln n}$$

$$\alpha_{i^*} = 50\ln n \qquad i^* \text{ is the smallest } i \text{ for which } \alpha_{i-1} < \frac{1}{4} \cdot \sqrt[d]{\frac{25}{\rho}(n-2)^{d-1} \cdot \ln n}$$

$$\alpha_{i^*+1} = 10$$

Solving the recurrence we get for $\kappa \leq i < i^*$,

$$\begin{aligned}\alpha_{i+\kappa} &= (2\rho \cdot 4^d)^{\frac{d^i-1}{d-1}} \cdot \left(\frac{\rho}{\kappa}\right)^{d^i}(n-2) = \frac{1}{\sqrt[d-1]{2\rho \cdot 4^d}} \cdot \left[\frac{\rho \cdot \sqrt[d-1]{2\rho \cdot 4^d}}{\kappa}\right]^{d^i}(n-2) \\ &= \frac{1}{\sqrt[d-1]{2\rho \cdot 4^d}} \cdot \frac{n-2}{e^{d^i}},\end{aligned} \tag{2.3}$$

and since i^* is the smallest integer satisfying

$$\alpha_{i^*-1} < \frac{1}{4} \cdot \sqrt[d]{\frac{25}{\rho}(n-2)^{d-1} \cdot \ln n},$$

we get that

$$i^* = \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right).$$

In order to see that, let first $i^* - 2 = j + \kappa$. Then

$$\alpha_{i^*-2} = \alpha_{j+\kappa} = \frac{1}{\sqrt[d-1]{2\rho \cdot 4^d}} \cdot \frac{n-2}{e^{d^j}}.$$

Since

$$\alpha_{i^*-2} \geq \frac{1}{4} \cdot \sqrt[d]{\frac{25}{\rho}(n-2)^{d-1} \cdot \ln n},$$

we get, successively,

$$\begin{aligned}
\frac{1}{\sqrt[d-1]{2\rho \cdot 4^d}} \cdot \frac{n-2}{e^{d^j}} &\ge \frac{1}{4} \cdot \sqrt[d]{\frac{25}{\rho}(n-2)^{d-1} \cdot \ln n} \\
\frac{1}{2^d \rho^d \cdot 4^{d^2}} \cdot \frac{(n-2)^{d(d-1)}}{e^{d^{j+1}(d-1)}} &\ge \frac{1}{4^{d(d-1)}} \cdot \frac{25^{d-1}}{\rho^{d-1}}(n-2)^{(d-1)^2} \cdot (\ln n)^{d-1} \\
\frac{1}{2^{2d^2+d}\rho} \cdot \frac{(n-2)^{d^2-d}}{e^{d^{j+1}(d-1)}} &\ge \frac{1}{2^{2d^2-2d}} \cdot 25^{d-1}(n-2)^{d^2-2d+1} \cdot (\ln n)^{d-1} \\
2^{-2d^2-d+2d^2-2d} \cdot \frac{(n-2)^{d^2-d-d^2+2d-1}}{e^{d^{j+1}(d-1)}} &\ge \rho \cdot 25^{d-1}(\ln n)^{d-1} \\
e^{d^{j+1}(d-1)} &\le \frac{1}{\rho \cdot 2^{3d} \cdot 25^{d-1}} \cdot (n-2)^{d-1} \cdot \frac{1}{(\ln n)^{d-1}} \\
d^{j+1}(d-1) &\le (d-1)\ln(n-2) - (d-1)\ln\ln n - \ln(\rho \cdot 2^{3d} \cdot 25^{d-1}) \\
d^{j+1} &\le \ln(n-2) - \ln\ln n - \frac{\ln(\rho \cdot 2^{3d} \cdot 25^{d-1})}{d-1} \\
j+1 &\le \frac{\ln\ln(n-2)}{\ln d} - \frac{\ln\ln\ln n}{\ln d} - \frac{\ln \frac{\ln(\rho \cdot 2^{3d} \cdot 25^{d-1})}{d-1}}{\ln d}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
i^* &= j + \kappa + 2 \\
&\le \frac{\ln\ln(n-2)}{\ln d} - \frac{\ln\ln\ln n}{\ln d} - \frac{\ln \frac{\ln(\rho \cdot 2^{3d} \cdot 25^{d-1})}{d-1}}{\ln d} + \kappa + 1 \\
&= \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right).
\end{aligned}$$

Next we define $T = n(i^* - \kappa + 3) = O\left(n\frac{\ln\ln n}{\ln d}\right)$ and an increasing sequence of points in time: Let $t_{\kappa-1} = \tau - T$ and for $i \ge \kappa$, $t_i = t_{i-1} + n$, so that the end of the last period, t_{i^*+2}, is the current time τ.

Let $\mathcal{E}$ denote the event "at time $t_{\kappa-1} = \tau - T$ there are at most $(1+\epsilon)N\rho$ calls in the system," for some constant $\epsilon > 0$, and let

$$\mathcal{C}_i = \{\forall v \in V, t \in [t_i, \tau] : L^v_{\ge i}(t) \le 2\alpha_i\}.$$

We show by induction that for $i = \kappa, \ldots, i^* + 1$

$$\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{E}) \le \frac{2i}{n^2}. \tag{2.4}$$

Initially we prove the following lemma, which we use throughout the proof.

Lemma 2.2.1. *Let $\mathcal{A}$ and $\mathcal{B}$ be events such that $\mathbf{Pr}(\mathcal{B}) \ge 1 - n^{-c}$ for some constant c, for n large enough. Then for any constant $\zeta > 0$ we have*

$$\mathbf{Pr}(\mathcal{A} \,|\, \mathcal{B}) \le (1+\zeta)\,\mathbf{Pr}(\mathcal{A}),$$

for sufficiently large n.

Proof. We have

$$\mathbf{Pr}(\mathcal{A} \,|\, \mathcal{B}) = \frac{\mathbf{Pr}(\mathcal{A}, \mathcal{B})}{\mathbf{Pr}(\mathcal{B})} \leq \frac{\mathbf{Pr}(\mathcal{A})}{\mathbf{Pr}(\mathcal{B})} \leq \frac{1}{1-n^{-c}}\,\mathbf{Pr}(\mathcal{A}) \leq (1+\zeta)\,\mathbf{Pr}(\mathcal{A}).$$

□

Now we examine the base case of Relation 2.4. Let $\mathcal{C}_i^v$ be the event

$$\mathcal{C}_i^v = \{\forall t \in [t_i, \tau] : L_{\geq i}^v(t) \leq 2\alpha_i\},$$

and $\mathcal{J}^v$ be the event "no more than $2\lambda(n-1)T$ calls are generated with node v as an endpoint during $[\tau - T, \tau]$." We need to bound the probability of $\overline{\mathcal{J}^v}$, so we prove the following lemma.

Lemma 2.2.2. *For sufficiently large n, we have*

$$\mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{E}) < n^{-4}.$$

Proof. Node v has $n-1$ incident links, on each of which new calls are generated according to a Poisson process with rate λ, independently of the other links. Therefore, the number of new calls with v as an endpoint during T steps is distributed according to a Poisson$(\lambda(n-1)T)$. So by applying a Chernoff bound for the Poisson distribution[3] we get that

$$\begin{aligned}\mathbf{Pr}(\overline{\mathcal{J}^v}) &\leq \frac{e^{-\lambda(n-1)T}(e\lambda(n-1)T)^{2\lambda(n-1)T}}{(2\lambda(n-1)T)^{2\lambda(n-1)T}} \\ &= e^{-\lambda(n-1)T + 2\lambda(n-1)T + 2\lambda(n-1)T\ln(\lambda(n-1)T) - 2\lambda(n-1)T\ln(2\lambda(n-1)T)} \\ &= e^{-\lambda(n-1)T(2\ln 2 - 1)} \\ &< n^{-4},\end{aligned}$$

for sufficiently large n. To complete the proof, we use the fact that the number of new calls during $[\tau - T, \tau]$ is independent of event $\mathcal{E}$. □

We now have

$$\begin{aligned}\mathbf{Pr}(\overline{\mathcal{C}_\kappa} \,|\, \mathcal{E}) &\leq n\,\mathbf{Pr}(\overline{\mathcal{C}_\kappa^v} \,|\, \mathcal{E}) \\ &\leq n\,\mathbf{Pr}(\overline{\mathcal{C}_\kappa^v} \,|\, \mathcal{J}^v, \mathcal{E}) + n\,\mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{E}).\end{aligned} \tag{2.5}$$

By Lemma 2.2.2, the second term is bounded by $n \cdot n^{-4}$, and we now bound the first term. Conditioning on $\mathcal{J}^v$, we have at most $2\lambda(n-1)T$ new jobs during $[t_{\kappa-1}, \tau]$, say at times $\{\hat{t}_j,\ j = 1, 2, \ldots\}$. Define also $\hat{t}_0 = t_\kappa$. Then

$$\mathbf{Pr}(\overline{\mathcal{C}_\kappa^v} \,|\, \mathcal{J}^v, \mathcal{E}) \leq \sum_{\substack{j=0 \\ \hat{t}_j \geq t_\kappa}}^{2\lambda(n-1)T} \mathbf{Pr}(L_{\geq\kappa}^v(\hat{t}_j) > 2\alpha_\kappa \,|\, \mathcal{J}^v, \mathcal{E}). \tag{2.6}$$

[3]Assume that X is distributed according to a Poisson distribution with rate λ. Then (see, for example, [69, page 416])

$$\mathbf{Pr}(X \geq i) \leq \frac{e^{-\lambda}(e\lambda)^i}{i^i}.$$

Let us compute the number of calls in the system with node v as an endpoint at time $\hat{t}_j$. These calls can be separated to calls that were in the system before time $t_{\kappa-1}$ (let x be their number), and calls that arrived after $t_{\kappa-1}$ (say y).

In order to compute x, we can notice that each of the x calls remains in the system until time $\hat{t}_j$ with probability $e^{-\mu(\hat{t}_j - t_{\kappa-1})}$. Since $\hat{t}_j \geq t_\kappa = t_{\kappa-1} + n$, the probability that a such call survives is bounded by $e^{-n\mu}$. So,

$$\mathbf{Pr}(x > 0 \,|\, \mathcal{E}) \leq (1+\epsilon)N\rho e^{-n\mu} < \frac{1}{n^7},$$

and we conclude that conditioning on event $\mathcal{E}$, $x = 0$ with probability at least $1 - n^{-7}$, for sufficiently large n.

In order to bound y, the number of calls arrived after time point $t_{\kappa-1}$, we prove the following lemma.

Lemma 2.2.3. *Consider a period Π and a given node v. The number of calls having node v as an endpoint that were generated during Π and are in the system at the end of Π is distributed according to a Poisson distribution with rate bounded by $\rho(n-1)$, independently of $\mathcal{E}$.*

Proof. Let Δ be the duration of the period Π, and let Y be a random variable counting the number of calls that were generated during Π, had v as an endpoint and are in the system at the end of Π. Node v has $n-1$ incident links on each of which new calls are generated with rate λ, independently of each other. The duration of each call is exponentially distributed with parameter μ. This process is an infinite server Poisson queue [67, page 18] in which the number of calls at the end of the period is distributed according to a Poisson distribution with rate

$$\lambda(n-1)\Delta p,$$

where

$$p = \int_0^\Delta \frac{e^{-\mu(\Delta - x)}}{\Delta} \mathrm{d}x = \frac{1}{\mu\Delta}\left(1 - e^{-\mu\Delta}\right) \leq \frac{1}{\mu\Delta}.$$

So Y is distributed according to a Poisson distribution with rate at most $\lambda(n-1)/\mu = \rho(n-1)$. Notice also that since Y does not depend on any event prior of Π, the distribution of Y conditioned on $\mathcal{E}$ is still Poisson with the same rate. □

By applying this lemma, we have that y is bounded by a Poisson($\rho(n-1)$). So, from the Chernoff bound, we conclude that $y \leq 2\rho(n-2)$ with probability at least $1 - n^{-7}$, for sufficiently large n.

The probability that at time $\hat{t}_j$ there are more than $2\rho(n-2)$ calls with node v as an endpoint is bounded by

$$\mathbf{Pr}(x > 0 \vee y > 2\rho(n-2) \,|\, \mathcal{E}),$$

which, using the previous facts, can be bounded by $2n^{-7}$.

Notice now that if node v has fewer than $2\rho(n-2)$ calls at time $\hat{t}_j$, then

$$L^v_{\geq\kappa}(\hat{t}_j) \leq \frac{2\rho(n-2)}{\kappa} = 2\alpha_\kappa.$$

Hence, for all $\hat{t}_j \geq t_\kappa$ we have

$$\mathbf{Pr}(L^v_{\geq\kappa}(\hat{t}_j) > 2\alpha_\kappa \,|\, \mathcal{E}) \leq 2n^{-7},$$

and by making use of Lemma 2.2.1, we get

$$\mathbf{Pr}(L^v_{\geq\kappa}(\hat{t}_j) > 2\alpha_\kappa \,|\, \mathcal{J}^v, \mathcal{E}) \leq 2 \cdot 2n^{-7} = 4n^{-7}. \tag{2.7}$$

Combining Relations (2.5), (2.6), (2.7), Lemma 2.2.2, and the fact that $T = O(n^2)$, we get that

$$\mathbf{Pr}(\overline{\mathcal{C}_\kappa} \,|\, \mathcal{E}) \leq n \cdot (2\lambda(n-1)+1) \cdot n^2 \cdot 4n^{-7} + n \cdot n^{-4} \leq n^{-2},$$

for large enough n, which completes the base case ($i = \kappa$) of Relation (2.4).

For the induction step we assume that

$$\mathbf{Pr}(\overline{\mathcal{C}_{i-1}} \,|\, \mathcal{E}) \leq \frac{2(i-1)}{n^2}. \tag{2.8}$$

Assume now that at time t a new call enters the system. Then the call is routed through an edge with (new) load greater or equal to i if in all the d alternative paths at least one of the two edges had load at least $i-1$. More concretely, let $\mathcal{G}$ denote the event "a new call is generated at time t with v as an endpoint," and let u be the other endpoint and $(w_j, j = 1, \ldots, d)$ be the intermediate nodes of the queried alternative paths.

We then have

$$\begin{aligned}
&\mathbf{Pr}(M^v_{\geq i}(t) > M^v_{\geq i}(t-) \,|\, \Phi(t-), \mathcal{G}) \\
&\qquad \leq \mathbf{Pr}(M^v_{\geq i}(t) > M^v_{\geq i}(t-) \vee M^u_{\geq i}(t) > M^u_{\geq i}(t-) \,|\, \Phi(t-), \mathcal{G}) \\
&\qquad \leq \mathbf{Pr}(\forall j \in \{1, \ldots, d\} : \ell_{(v,w_j)}(t-) \geq i-1 \vee \ell_{(u,w_j)}(t-) \geq i-1 \,|\, \Phi(t-), \mathcal{G}) \\
&\qquad \leq \left(\frac{L^v_{\geq i-1}(t-) + L^u_{\geq i-1}(t-)}{n-2}\right)^d,
\end{aligned}$$

therefore,

$$\begin{aligned}
&\mathbf{Pr}(M^v_{\geq i}(t) > M^v_{\geq i}(t-) \,|\, \mathcal{E}, \mathcal{G}, \forall z \in V : L^z_{\geq i-1}(t-) \leq 2\alpha_{i-1}) \\
&\qquad\qquad \leq \left(\frac{2 \cdot 2\alpha_{i-1}}{n-2}\right)^d \doteq q_i.
\end{aligned} \tag{2.9}$$

Notice that for $i = \kappa+1, \ldots, i^*$ we have

$$q_i \leq \frac{\alpha_i}{2\rho(n-2)}. \tag{2.10}$$

We now define

$$\mathcal{F}_i = \{\forall v \in V : M^v_{\geq i}(t_i) < \alpha_i\}$$

and prove Lemmata 2.2.4 and 2.2.6, that allow us to conclude that $\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{E}) \leq \frac{2i}{n^2}$, and establish Relation (2.4).

Lemma 2.2.4. *Under the inductive hypothesis*

$$\mathbf{Pr}(\overline{\mathcal{F}_i}\,|\,\mathcal{C}_{i-1},\mathcal{E}) \leq n^{-2}.$$

Proof. First we apply Lemma 2.2.3 for the interval $\Pi = [t_{\kappa-1}, t_{i-1}]$ and we deduce that the number of calls with v as an endpoint that were generated during Π and remained until time t_{i-1} follows a Poisson distribution with mean bounded by $\rho(n-1)$. Hence, with a Chernoff bound, we get that with probability at least $1-n^{-3}$ there are at most $2\rho(n-1)$ such calls. If we condition on event $\mathcal{E}$, then the total number of calls in the system at time t_{i-1} with node v as an endpoint is at most

$$(1+\epsilon)N\rho + 2\rho(n-1)$$

with probability at least $1-n^3$. The probability that each of these calls stays in the system until time t_i is bounded by $e^{-n\mu}$ (recall that $t_i - t_{i-1} = n$), so the probability, conditioned on the event $\mathcal{E}$, that some of the calls that were in the system up to time t_{i-1} and had v as an endpoint, stays in the system until time t_i is bounded by

$$n^{-3} + [(1+\epsilon)N\rho + 2\rho(n-1)]e^{-n\mu} < 2n^{-3}$$

for sufficiently large n. By applying Lemma 2.2.1 and making use of the induction hypothesis (Equation (2.8)) we deduce that the probability that some of those calls stay in the system conditioned on the events $\mathcal{C}_{i-1}$ and $\mathcal{E}$ is bounded by $4n^{-3}$. To analyze the number of the remaining calls that were created during the period $[t_{i-1}, t_i]$, we make use of Lemma 2.2.5 which completes the proof of this one. □

Lemma 2.2.5. *Consider a period Π and a given node v. Conditioning on $\mathcal{C}_{i-1}$ and $\mathcal{E}$, the number of new calls that increased $M^v_{\geq i}$ when they were generated, and remained until the end of Π is less than α_i, with probability at least $1-n^{-7}$.*

Proof. Let Y be the number of calls that were generated during Π, had v as an endpoint and are in the system at the end of Π. By applying Lemma 2.2.3 we get that conditioned on $\mathcal{E}$, Y follows a Poisson distribution with rate bounded by $\rho(n-1)$.

Let now Z be the number of calls in the system at the end of Π whose arrival resulted in the increase of $M^v_{\geq i}$. Denote with $\mathcal{H}_k$ the event $\{Y = k\}$ and let $\{\tilde{t}_j\}_{j=1}^k$ be the time of the arrival of the jth call that exists in the system at the end of Π. We can then write

$$\mathbf{Pr}(Z > r\,|\,\mathcal{E},\mathcal{C}_{i-1}) = \sum_k \mathbf{Pr}(Z > r\,|\,\mathcal{E},\mathcal{C}_{i-1},\mathcal{H}_k)\cdot\mathbf{Pr}(\mathcal{H}_k\,|\,\mathcal{E},\mathcal{C}_{i-1}).$$

We now fix k and we consider the random variables $\{Z_j\}_{j=1}^k$, where

$$\begin{aligned} Z_j = 1 \qquad & \text{if } M^v_{\geq i}(\tilde{t}_j) > M^v_{\geq i}(\tilde{t}_j-) \\ & \text{and } \forall z \in V : L^z_{\geq i-1}(\tilde{t}_j-) \leq 2\alpha_{i-1}. \end{aligned}$$

From Relation (2.9) we get that

$$\mathbf{Pr}(Z_j = 1\,|\,\mathcal{E}) \leq q_i,$$

so, since (induction hypothesis (2.4)) $\mathbf{Pr}(\mathcal{C}_{i-1} \,|\, \mathcal{E}) \geq 1 - 2(i-1)/n^2$, we can apply Lemma 2.2.1 and get

$$\mathbf{Pr}(Z_j = 1 \,|\, \mathcal{E}, \mathcal{C}_{i-1}) \leq (1+\zeta)q_i, \tag{2.11}$$

for some constant ζ (say 0.05), independently of all the previous Z_j. Notice now that conditioning on events $\mathcal{C}_{i-1}$, and $\mathcal{H}_k$, we have

$$Z = \sum_{j=1}^{k} Z_j.$$

Hence

$$\mathbf{Pr}(Z > r \,|\, \mathcal{E}, \mathcal{C}_{i-1}) = \sum_k \mathbf{Pr}\left(\sum_{j=1}^{k} Z_j > r \,\middle|\, \mathcal{E}, \mathcal{C}_{i-1}, \mathcal{H}_k\right) \cdot \mathbf{Pr}(\mathcal{H}_k \,|\, \mathcal{E}, \mathcal{C}_{i-1}).$$

Again by Lemma 2.2.1, we get

$$\mathbf{Pr}(\mathcal{H}_k \,|\, \mathcal{E}, \mathcal{C}_{i-1}) \leq 2\mathbf{Pr}(\mathcal{H}_k \,|\, \mathcal{E}).$$

So by the fact that the distribution of Y conditioned on $\mathcal{E}$ is Poisson with rate at most $\rho(n-1)$, and by Relation (2.11), we can finally conclude that

$$\begin{aligned}\mathbf{Pr}(Z > r \,|\, \mathcal{E}, \mathcal{C}_{i-1}) &\leq 2\sum_k \mathbf{Pr}(\text{Binomial}(k, (1+\zeta)q_i) > r) \cdot \mathbf{Pr}(\text{Poisson}(\rho(n-1)) = k) \\ &\leq 2\mathbf{Pr}(\text{Poisson}((1+\zeta)\rho q_i(n-1)) > r).\end{aligned}$$

We now distinguish the following two cases:

Case 1: For $i \leq i^*$, by using Equation (2.10) we get that $(1+\zeta)\rho q_i(n-1) \leq 1.1\alpha_i/2$ for $\zeta = 0.05$, and by applying the Chernoff bound, we get that the probability that the number of calls is higher than α_i is bounded by

$$2\frac{e^{-\frac{1.1\alpha_i}{2}}(e\frac{1.1\alpha_i}{2})^{\alpha_i}}{\alpha_i^{\alpha_i}} \leq 2e^{-0.147\alpha_i}.$$

For $i < i^*$ we have from the definition of α_i

$$\begin{aligned}2e^{-0.147\alpha_i} &= 2e^{-0.147\frac{2\rho\cdot 4^d\alpha_{i-1}^d}{(n-2)^{d-1}}} \\ &\leq 2e^{-0.147\frac{2\rho\cdot 4^d \frac{1}{4^d}\frac{25}{\rho}(n-2)^{d-1}\ln n}{(n-2)^{d-1}}} \\ &= 2e^{-0.147\cdot 50\ln n} \\ &= o\left(\frac{1}{n^7}\right),\end{aligned}$$

while for $i = i^*$ we get

$$\begin{aligned}e^{-0.147\alpha_i} &= 2e^{-0.147\cdot 50\ln n} \\ &= o\left(\frac{1}{n^7}\right).\end{aligned}$$

Case 2: For $i = i^* + 1$, using Equation (2.9) we get that the parameter of the Poisson distribution is

$$(1+\zeta)\rho q_i(n-1) \leq (1+\zeta)\frac{4^d \cdot \alpha_{i-1}^d}{(n-2)^d}\rho(n-1) = (1+\zeta)\frac{(4 \cdot 50 \ln n)^d}{(n-2)^d}\rho(n-1),$$

and with the Chernoff bound we get that the probability that the number of calls is higher than $\alpha_{i^*+1} = 10$ is $o(1/n^7)$.

□

Lemma 2.2.6. *Under the inductive hypothesis*

$$\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \leq n^{-2}.$$

Proof. First we compute

$$\begin{aligned}\mathbf{Pr}(\mathcal{F}_i, \mathcal{C}_{i-1} \,|\, \mathcal{E}) &= \mathbf{Pr}(\mathcal{C}_{i-1} \,|\, \mathcal{E}) \cdot \mathbf{Pr}(\mathcal{F}_i \,|\, \mathcal{C}_{i-1}, \mathcal{E}) \\ &\geq \left(1 - \frac{i-1}{n^2}\right) \cdot \left(1 - \frac{1}{n^2}\right),\end{aligned}$$

by Relation (2.8) and Lemma 2.2.4, so

$$\mathbf{Pr}(\mathcal{F}_i, \mathcal{C}_{i-1} \,|\, \mathcal{E}) \geq 1 - \frac{1}{n}.$$

So, by Lemma 2.2.1 we get

$$\mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \leq 2\,\mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{E})$$

and finally, by using Lemma 2.2.2, we conclude

$$\mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \leq 2\,n^{-4}. \tag{2.12}$$

Hence, we can get

$$\begin{aligned}\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) &\leq n \cdot \mathbf{Pr}(\overline{\mathcal{C}_i^v} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \\ &\leq n \cdot \mathbf{Pr}(\overline{\mathcal{C}_i^v} \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) + n \cdot \mathbf{Pr}(\overline{\mathcal{J}^v} \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E})\end{aligned} \tag{2.13}$$

We have a bound for the second term, so we want to bound the first one. For that, we write (recall that $\{\hat{t}_j\}$ are the times of the arrivals of the new calls with node v as an endpoint)

$$\begin{aligned}\mathbf{Pr}(\overline{\mathcal{C}_i^v} \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) &\leq \mathbf{Pr}(\exists \tilde{t} \in [t_i, \tau] : L_{\geq i}^v(\tilde{t}) > 2\alpha_i \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \\ &\leq \mathbf{Pr}(\exists \tilde{t} \in [t_i, \tau] : M_{\geq i}^v(\tilde{t}) > 2\alpha_i \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) \\ &\leq \sum_{\substack{j=1 \\ \hat{t}_j \geq t_i}}^{2\lambda(n-1)T} \mathbf{Pr}(M_{\geq i}^v(\hat{t}_j) > 2\alpha_i \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E})\end{aligned} \tag{2.14}$$

Conditioning on event $\mathcal{F}_i$, we have $M_{\geq i}^v(\hat{t}_j) > 2\alpha_i$ only if $M_{\geq i}^v$ increased by at least α_i during the interval $[t_i, \hat{t}_j]$. Therefore, by applying Lemmata 2.2.1, 2.2.4, and 2.2.5, we get

$$\mathbf{Pr}(M_{\geq i}^v(\hat{t}_j) > 2\alpha_i \,|\, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) < \frac{2}{n^7}.$$

We combine this result with Relation (2.12) and Lemma 2.2.1 and we have

$$\mathbf{Pr}(M^v_{\geq i}(\hat{t}_j) > 2\alpha_i \,|\, \mathcal{J}^v, \mathcal{F}_i, \mathcal{C}_{i-1}, \mathcal{E}) < \frac{4}{n^7}. \tag{2.15}$$

If we combine Relations (2.13), (2.14), and (2.15), we get the result. □

Having proven Lemmata 2.2.4 and 2.2.6 we can now show that $\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{E}) \leq 2i/n^2$:

$$\begin{aligned}
\mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{E}) &= \mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{C}_{i-1}, \mathcal{E}) \cdot \mathbf{Pr}(\mathcal{C}_{i-1}, \mathcal{E}) \\
&\quad + \mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \overline{\mathcal{C}_{i-1}}, \mathcal{E}) \cdot \mathbf{Pr}(\overline{\mathcal{C}_{i-1}}, \mathcal{E}) \\
&\leq \mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{C}_{i-1}, \mathcal{E}) + \frac{2(i-1)}{n^2} \\
&= \mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{C}_{i-1}, \mathcal{F}_i, \mathcal{E}) \cdot \mathbf{Pr}(\mathcal{F}_i \,|\, \mathcal{C}_{i-1}, \mathcal{E}) \\
&\quad + \mathbf{Pr}(\overline{\mathcal{C}_i} \,|\, \mathcal{C}_{i-1}, \overline{\mathcal{F}_i}, \mathcal{E}) \cdot \mathbf{Pr}(\overline{\mathcal{F}_i} \,|\, \mathcal{C}_{i-1}, \mathcal{E}) + \frac{2(i-1)}{n^2} \\
&\leq \frac{1}{n^2} + \frac{1}{n^2} + \frac{2(i-1)}{n^2} \\
&= \frac{2i}{n^2}
\end{aligned}$$

We have therefore shown that the event $\mathcal{C}_{i^*+1}$ holds whp., which implies that for every node v, after the (i^*+1)th period, there will be no more than $2\alpha_{i^*+1} = 20$ incident edges with load more than i^*+1. We will now bound the probability that in the next interval ($[t_{i^*+1}, t_{i^*+2}]$, the last interval of T) there will be an incident edge of v with load more than i^*+3, conditioning on the event $\mathcal{C}_{i^*+1}$. For this to happen, we must have at least 2 new calls to be routed using one of the 20 highly loaded edges. The probability that two specific new calls use these edges is at most

$$\left(\frac{20+20}{n-2}\right)^{2d} = O\left(\frac{1}{n^4}\right), \tag{2.16}$$

since $d \geq 2$. The expected number of calls with v as an endpoint is $\lambda(n-1)n$, since $(n-1)$ links are connected to v in each of which new calls are generated with rate λ, while the total length of the interval is n. This implies that whp. there will be $O(n^2)$ new calls in the whole period. By combining this fact with Equation (2.16), applying Lemma 2.2.1, and summing for all the nodes we conclude that at the end of period T there will be no edges with load more than i^*+3 whp.

We now consider the stationary distribution π_n, and show that under it

$$\mathbf{Pr}\left(\ell_{\max} \leq \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right)\right) = 1 - o\left(\frac{1}{n}\right).$$

where

$$\ell_{\max} = \max_{e=(u,v)\in E} \max\{\ell_{e,u}, \ell_{e,v}\}$$

denotes the maximum number of calls on any edge, in the stationary regime ($\ell_{e,u}$ is the number of calls with u as an endpoint routed through edge e in the stationary regime). Recall that $\Phi(t)$ is the state of the system at time t, and consider the following partitioning of the state space, Σ, of the underlying Markov process:

- $$S_1 = \left\{ x : V(x) \le (1+\epsilon)N\rho,\ \ell_{\max} \le \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right) \right\},$$
  that is, states in which the total number of calls in the system is at most $(1+\epsilon)N\rho$, and the maximum load is at most $\frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right)$.
- $$S_2 = \left\{ x : V(x) \le (1+\epsilon)N\rho,\ \ell_{\max} > \frac{\ln\ln n}{\ln d} + \Omega\left(\frac{\ln\ln n}{\ln d}\right) \right\},$$
  that is, states in which the total number of calls in the system is at most $(1+\epsilon)N\rho$, and the maximum load is higher than $\frac{\ln\ln n}{\ln d} + \Omega\left(\frac{\ln\ln n}{\ln d}\right)$.
- $$S_3 = \{x : V(x) > (1+\epsilon)N\rho\},$$
  that is, states in which the total number of calls in the system is higher than $(1+\epsilon)N\rho$.

We have shown that

$$\mathbf{Pr}(\Phi(\tau) \in S_2 \,|\, \Phi(\tau - T) \in S_1 \cup S_2) = o\left(\frac{1}{n}\right)$$

and we can easily show that

$$\mathbf{Pr}(\Phi(\tau) \in S_3 \,|\, \Phi(\tau - T) \in S_1 \cup S_2) = o\left(\frac{1}{n}\right)$$

Moreover, in the stationary distribution the number of calls in the system has a Poisson distribution with parameter $N\rho$. Hence by using the Chernoff bound

$$\sum_{i \in S_3} (\pi_n)_i = o\left(\frac{1}{n}\right)$$

Then we have

$$\sum_{i \in S_2 \cup S_3} (\pi_n)_i = \sum_{i \in S_2} (\pi_n)_i + \sum_{i \in S_3} (\pi_n)_i$$

The second term is $o(1/n)$, while for the first one

$$\begin{aligned}
\sum_{i \in S_2} (\pi_n)_i &= \sum_j \mathbf{Pr}(\Phi(\tau) \in S_2 \,|\, \Phi(\tau - T) = j) \cdot (\pi_n)_j \\
&= \sum_{j \in S_1 \cup S_2} \mathbf{Pr}(\Phi(\tau) \in S_2 \,|\, \Phi(\tau - T) = j) \cdot (\pi_n)_j \\
&\quad + \sum_{j \in S_3} \mathbf{Pr}(\Phi(\tau) \in S_2 \,|\, \Phi(\tau - T) = j) \cdot (\pi_n)_j \\
&= \sum_{j \in S_1 \cup S_2} (\pi_n)_j \cdot o\left(\frac{1}{n}\right) + o\left(\frac{1}{n}\right) = o\left(\frac{1}{n}\right)
\end{aligned}$$

Therefore,

$$\sum_{i \in S_2 \cup S_3} (\pi_n)_i = o\left(\frac{1}{n}\right),$$

which implies that

$$\sum_{i \in S_1} (\pi_n)_i = 1 - o\left(\frac{1}{n}\right)$$

and completes the proof of Theorem 2.2.3. □

2.2.2 Bounded Capacities

In this section we use the analysis of the BDAR* algorithm for unbounded capacities to compute the bandwidth requirement B ($< \infty$) that ensures that a new call is not lost whp.

Theorem 2.2.4. *Assume that all the edges have capacity $3B$ circuits, which can be a function of n. Then, if we perform the BDAR* policy, capacity*

$$B = \frac{\ln\ln n}{\ln d} + o\left(\frac{\ln\ln n}{\ln d}\right) \qquad \text{as } n \to \infty$$

ensures that under the stationary distribution a new call is not lost whp.

Proof. The result for finite B follows from the proof of Theorem 2.2.3, which concerns unbounded capacity. Since the Markov process is finite and aperiodic there exists a stationary distribution. Moreover, the analysis for the unbounded case still holds for finite B as long as $B \leq i^* + 1$.

A new call between nodes u and v will be rejected if in all the d choices, either the edge incident to node u is used in routing $i^* + 1 = \ln\ln n/\ln d + o(\ln\ln n/\ln d)$ calls with node u as an endpoint, or the edge incident to node v is used in routing $i^* + 1$ calls with node v as an endpoint. With probability at least $1 - o(n^{-1})$, for each node, the number of incident edges with load at least $i^* + 1$ is at most $2\alpha_{i^*+1}$. Therefore, the probability for a call to be rejected is no more than

$$o\left(\frac{1}{n}\right) + \left(\frac{2\alpha_{i^*+1} + 2\alpha_{i^*+1}}{n-2}\right)^d = o\left(\frac{1}{n}\right)$$

since $\alpha_{i^*+1} = 10$. □

2.3 Lower Bound on the Performance of the DAR Algorithm

To demonstrate the advantage of the balanced-allocation method we prove here a lower bound on the maximum channel load when requests are routed using the DAR algorithm. This bound shows an exponential gap between the capacity required by the balanced-allocation algorithm and the capacity required by the standard DAR algorithm for the same stream of inputs.

Recall from Section 2.1 that we consider a complete network of n nodes and $N = \binom{n}{2}$ edges. Requests for connections between a given pair arrive according to a Poisson process with rate λ, the duration of a connection has an exponential distribution with expectation $1/\mu$. Edges have capacities of $3B$ circuits, B are used for direct connections, and the remaining $2B$ are used for alternative routes with the capacity reserved for alternative routes furthermore split into two, so that B circuits are used for alternate paths with one node of the edge as an endpoint and B for calls with the other node as an endpoint.

Theorem 2.3.1. *Assume that all the edges have capacity $3B$ circuits, which can be a function of n. Then, if we perform the DAR policy, capacity*

$$B = \Omega\left(\sqrt{\frac{\ln n}{d\ln\ln n}}\right), \qquad \text{as } n \to \infty$$

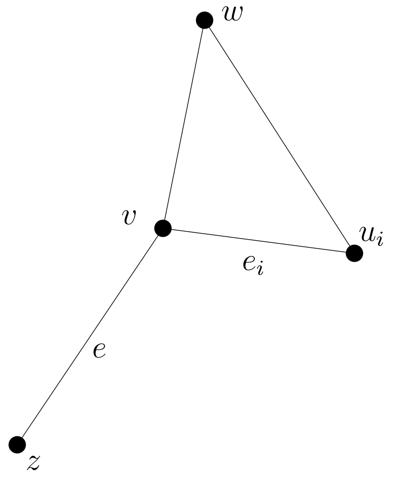


Figure 2.1: A call is generated on edge e at time t.

is necessary to ensure that under the stationary distribution a new call is not lost whp.

Proof. We will compute a lower bound on the probability $P = P(B)$, that a request arriving at an arbitrary time t is rejected.

We consider first the probability P_1 that the new call is not routed through the direct link. The process of routing calls through the direct link is an $M/M/B/B$ loss system (Poisson arrival, exponential service time, B servers—corresponding to the B direct links, up to B customers in the system—corresponding to up to B calls that can be routed through the direct links). Applying Erlang's loss formula (e.g., [45]),

$$P_1 = \frac{(\lambda/\mu)^B}{B!}\left(\sum_{i=0}^{B}\frac{(\lambda/\mu)^B}{i!}\right)^{-1} \geq e^{-\lambda/\mu}\frac{(\lambda/\mu)^B}{B!}. \tag{2.17}$$

Since the arrival is Poisson, it is independent of the state of the queue at the time of arrival, hence the probability that a given pair (v, w) had a request during interval $\Pi = [t-1, t]$ that could not be routed by the direct link is

$$P_{\text{alternate}} = (1 - e^{-\lambda})P_1.$$

Next we lower bound the probability P_2 that a request generated at time t that failed to use the direct link $e = (v, z)$, fails also to be routed by an alternative path (i.e., all the d attempts to find a nonsaturated alternative path do not succeed). In fact, we will restrict our discussion to the probability that in each of these d routes the first edge (v, u_i) on the alternate route was saturated for alternate paths with endpoint v (Figure 2.1).

In order to estimate the probability P_2, we compute a lower bound for the probability $P(e_i, t)$, that an arbitrary edge $e_i = (v, u_i)$ was carrying, at time t, B alternate paths with endpoint v (and thus blocked for any other alternate path starting at v). For this we study the evolution of the system during period $\Pi = [t-1, t]$. We will lower bound the probability $P(e_i, t)$ by the probability that at some point during the interval Π the edge carried B alternate paths with endpoint v, and that none of these paths terminated during this interval.

The second requirement is easy to evaluate. Since the calls have exponential duration with parameter μ, every call that is on edge e_i at time $t-1$, or that is created during Π, will stay in the system until time t with probability at least $e^{-\mu}$, and all the calls do not terminate in that interval with probability at least $e^{-\mu B}$.

Let $\mathcal{C}_i$ be the event "during the interval Π, B different pairs $(v, w_1), \ldots, (v, w_B)$ try to use edge $e_i = (v, u_i)$ as a first choice for alternate path, and for each of these pairs the edge (u_i, w_j) (the second edge in the alternate path) was not blocked." Then,

$$P(e_i, t) \geq \mathbf{Pr}(\mathcal{C}_i) e^{-\mu B}.$$

The difficulty in computing $\mathbf{Pr}(\mathcal{C}_i)$ is bounding the probability that the second edge on the alternate path is not blocked. The following lemma simplifies this computation.

Lemma 2.3.1. *Let $\mathcal{D}$ be the event "there is a vertex $u \neq v$ that during the interval Π was the center node for more than $c_1 d\left(\frac{\lambda}{\mu} + \lambda\right)(n-1)$ alternate paths with no endpoint in v." Then,*

$$\mathbf{Pr}(\mathcal{D}) \leq e^{-c_2 n},$$

for some constants $c_1, c_2 > 0$.

Proof. There are $\binom{n-1}{2}$ possible pairs of vertices not containing v. For each pair the number of active calls at time $t-1$ is bounded by a Poisson random variable with parameter λ/μ. The number of new calls between a given pair during the interval is bounded by Poisson random variable with parameter λ.

Fix a vertex u. The probability that a given call uses u as a center vertex in an alternate path is bounded by $d/(n-2)$, independently of other calls. Thus, the number of alternating paths through u is stochastically dominated by a Poisson distribution with parameter $\lambda\left(1 + \frac{1}{\mu}\right) d \frac{n-1}{2}$. Applying the Chernoff bound for u and summing over all $n-1$ vertices gives the lemma. $\square$

There can be no more than B alternate paths with endpoint v that use a vertex w as a center node. Thus, conditioning on the event $\overline{\mathcal{D}}$, no more than $c_1 d\left(\frac{\lambda}{\mu} + \lambda\right)(n-1) + B$ alternate paths use any vertex $w \neq v$ during the interval Π, and thus, during any time in that interval no more than $\frac{1}{B}\left(c_1 d\left(\frac{\lambda}{\mu} + \lambda\right)(n-1) + B\right)$ edges adjacent to w are blocked for alternating paths using w as a center node.

Focusing back on the edge $e_i = (v, u_i)$, there is a set W_i of vertices such that the edge from u_i to $w \in W_i$ is not blocked for an alternate path with endpoints v and $w \in W_i$ throughout the interval Π. Conditioned on $\overline{\mathcal{D}}$, we have $|W_i| \geq \alpha n$ for some constant $\alpha > 0$.

We can compute

$$\begin{aligned} \mathbf{Pr}(\mathcal{C}_i \mid \overline{\mathcal{D}}) &\geq \binom{\alpha n}{B}\left(P_{\text{alternate}} \cdot \frac{1}{n-2}\right)^{B}\left(1 - P_{\text{alternate}} \cdot \frac{1}{n-2}\right)^{\alpha n - B} \\ &= e^{-O(B^2 \ln B - B^2 \ln(\lambda/\mu))}. \end{aligned} \tag{2.18}$$

The above follows from the fact that there are at least αn edges $(v, w), w \in W_i$, that can create a call during Π with probability $P_{\text{alternate}}$, and select as a first choice for alternate path the path $v - u_i - w$. Note that in the computation we consider no more than one communication request for each pair of vertices $(v, w), w \in W_i$, in order to avoid further dependencies.

Consider now a request that arrives at time t with endpoint v. The probability that the direct link for that request is blocked is P_1.

For simplicity, label the d alternative paths that the call generated at time t (between nodes v and z) as $v - u_i - z$, $i = 1, 2, \ldots, d$, and let $\mathcal{E}_i$ be the event "the ith alternative path $(v - u_i - z)$ is blocked." We want to lower bound the probability $P_2 = \mathbf{Pr}(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_d)$ that the request generated at time t that failed to use the direct link, fails to use all the d alternate paths. Then

$$\begin{aligned} P_2 &\geq \mathbf{Pr}(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_d) \cdot e^{-d\mu B} \\ &\geq \mathbf{Pr}(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_d \,|\, \overline{\mathcal{D}}) \cdot \mathbf{Pr}(\overline{\mathcal{D}}) \cdot e^{-d\mu B} \\ &\geq (1 - e^{-c_2 n}) \cdot e^{-d\mu B} \cdot \prod_{j=1}^{d} \mathbf{Pr}(\mathcal{C}_j \,|\, \overline{\mathcal{D}}, \mathcal{C}_1, \ldots, \mathcal{C}_{j-1}). \end{aligned}$$

Let us try to compute $\mathbf{Pr}(\mathcal{C}_j \,|\, \overline{\mathcal{D}}, \mathcal{C}_1, \ldots, \mathcal{C}_{j-1})$. Let

$$U_i = \{w \in W_i : v - u_i - w \text{ became an active alternate path during } \Pi\}$$

and

$$W_i = W_{i-1} \backslash U_{i-1} = W_1 \backslash \bigcup_{j=1}^{i-1} U_j.$$

Notice that if the calls $(v - u_i - w)$ do not terminate during Π, we have $|U_i| = B$, so as long as $dB = o(n)$, conditioned on $\overline{\mathcal{D}}$, there exists a constant α such that $|W_i| \geq \alpha n$, for all $i = 1, \ldots, d$. We can repeat the calculation of (2.18) and get that

$$\mathbf{Pr}(\mathcal{C}_j \,|\, \overline{\mathcal{D}}, \mathcal{C}_1, \ldots, \mathcal{C}_{j-1}) = e^{-O(B^2 \ln B - B^2 \ln(\lambda/\mu))},$$

since a call in W_i is generated, fails to use a direct route, and uses the alternate path $v - u_i - z$, independently of events $\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}$. So, finally, we get that

$$P_2 = e^{-O(dB^2 \ln B - dB^2 \ln(\lambda/\mu))}.$$

Putting everything together we conclude that the probability that the call generated at time t is rejected is at least

$$P_1 \cdot P_2 \geq e^{-O(dB^2 \ln B - dB^2 \ln(\lambda/\mu))}.$$

Therefore, in order to guarantee that a new call is not lost whp., the bandwidth must be at least

$$B = \Omega\left(\sqrt{\frac{\ln n}{d \ln \ln n}}\right).$$

$\square$

Chapter 3

Load Balancing Under Stochastic Adversarial Input

Efficient utilization of parallel and distributed systems can often depend on dynamic load balancing of individual tasks between processors. In the dynamic load balancing problem, we consider a system that is designed to run indefinitely. New jobs arrive during the run of the system, and existing jobs are executed by the processors and leave the system. The arrival of new jobs may not be evenly distributed between the processors. The task of the load balancing protocol is to maintain approximately uniform job load between the processors, and in particular to keep all processors working as long as there are jobs in the system waiting for execution.

We assume a simple combinatorial model of load balancing following a number of earlier studies. The computing system is represented by a connected, undirected, n-node graph. Jobs (represented by tokens) have equal execution time. The load of a node is the number of tokens in its queue. A processor can execute (or consume) one token per step, and we assume that it executes the oldest job in its queue. In each step a processor can also move a number of jobs from its queue to the queue of an adjacent node in the network. This abstraction models the case where the execution time of a job is significantly longer than the time required to move a job to an adjacent node. The assumption that all jobs have equal execution time simplifies the analysis while still capturing the combinatorial complexity of the load balancing problem in networks.

Dynamic load balancing algorithms have been studied extensively in experimental settings, demonstrating significant run-time improvements obtained by relatively simple load balancing techniques [78, 80]. Rigorous, theoretical study of load balancing in the past has focused mainly on static analysis [2, 31, 32, 52, 63, 65], where a set of jobs is initially placed in the processors and the algorithm needs to distribute the jobs almost evenly between the processors in a minimum number of parallel rounds. A number of important techniques have been developed in this line of work, and in particular our work here builds on the static analysis in [32].

Load balancing, however, is best analyzed in a dynamic setting that captures the actual application of such protocols. An important step in that direction was taken in [15], where the work stealing model was shown to be stable on the complete network. The main tool used in that work was the stability conditions for ergodic Markov chains, and consequently their stability result holds only for Markovian adversaries. In addition, a number of other works have studied dynamic load balancing under the assumptions that jobs are generated by a randomized process that is oblivious to the current state of the system [50, 56, 59, 70]. Finally, [9, 62] proved stability results for load balancing on a general network assuming the deterministic adversarial model. The computational model there is different, assuming that only one job can traverse an edge per step.

Most of the results of this chapter have appeared in [5], and in [6], and are joint work with Adam Kirsch and Eli Upfal.

3.1 Adversarial Models

There is a substantial class of results, based on modeling a system's evolution as a Markov chain, which is frequently employed in these sorts of analyses—this is the case, for example, in [15], where the adversary is a weaker version of the one that we present in Section 3.1.1. Furthermore, for this Markov-chain setting, there are additional tools for proving rapid convergence to a stationary distribution (see Meyn and Tweedie [55]), as well as for deriving properties of the stationary distribution (see [15] and the references therein). However, many of these results are qualitative and not quantitative. More importantly, they restrict the class of problems that they can model. In the particular context of adversarial load balancing, which is the focus of this work, any straightforward application of such tools will only yield results for Markovian adversaries (i.e., adversaries whose usage allows for the system to be modeled as a time-homogeneous Markov chain)—the analysis in [15] is an example. While it is not obvious whether this restricted class of adversaries is really any weaker than the general class presented earlier, it is clear that results in the general adversarial model serve as a compelling argument for the efficacy of the load balancing protocol under a wide range of input conditions.

Since we seek greater generality than Markov chain results are known to provide, we need more elaborate tools. In Section 3.1.1 we present a class of non-Markovian adversaries that generalize those that have appeared earlier in the literature [15], and in Section 3.4.1 we analyze the performance of our system under those adversaries. The main tools that we apply in order to show that the system has efficient long-term behavior under that class of adversaries are some results from renewal theory.

Subsequently, in Section 3.1.2 we initially consider the more general class of adversaries that was introduced by Borodin et al. in [17], and in Section 3.4.2 we analyze the system under those adversaries. To this end, we show that the load in the system behaves like a supermartingale above some threshold and then employ a result of Pemantle and Rosenthal [64] for the analysis of such processes. We show that the expected load and the expected waiting time of a newly arrived job is bounded by a small polynomial. Subsequently, we place some additional restriction to the power of those stochastic adversaries, and in Section 3.4.2, we derive stronger, high-probability results for those adversaries.

It turns out that the general adversarial model presented in Section 3.1.2 is strictly more general than the class of adversaries presented in Section 3.1.1, and, therefore, the results of Section 3.4.1 can be deduced as corollaries (if we ignore constant factors) of those of Section 3.4.2. Nevertheless, we choose to separately consider the former to illustrate the idea of applying tools from the theory of renewal processes to analyze the long-term system behavior.

3.1.1 Adversarial-Generator Models

The adversarial models that we consider here, are generalizations of the one presented in [15]. We present two simple but powerful adversaries:

1. a deterministic adversary that can add up to λn new jobs to the network at any step, and
2. a randomized adversary that places n generators in the processors in an arbitrary fashion, and each generator adds one new job to the processor in which it is placed, independently with probability at most $\lambda < 1$.

The placement of the generators to the processors can depend on the entire history of the system and not only on the current system state (i.e., the load of each processor at the current moment). This is the main difference of this input model from the one in [15], where the placement of the generators depends only on the current system state.

We give a high-level description of the protocol here, deferring the details to Section 3.2. After the generation of new load, the nodes execute a particular distributed randomized algorithm for choosing a random matching. The matching is not necessarily perfect, nor is it necessarily chosen uniformly from all possible matchings, although it does have some important properties that we will exploit later. Once the matching is chosen, every two matched nodes equalize their load (up to one job). For simplicity, we refer to this protocol as $\mathcal{P}$.

We first show that under these adversary input models the system (using protocol $\mathcal{P}$) is stable, that is, the expected total load in the system is bounded with respect to time. The following theorem, proven in Section 3.4.1, relates the load in the system to the network topology, and establishes the stability of the system. We assume a network $G = (V, E)$ that has n nodes, maximum degree at most Δ, and whose Laplacian[1] has smallest nontrivial eigenvalue Λ. For convenience, we define the quantity $\gamma = \Lambda/16\Delta$.

Theorem 3.1.1. *Suppose that we run the system under one of the two adversarial-generator models, using protocol $\mathcal{P}$. Then the system is stable and as time tends to infinity the expected total load in the system is $O(\gamma^{-1} n \ln n)$. If the system starts with no load, the result holds for any time step.*

[1] Let A denote the adjacency matrix of a graph G, and let $D = (d_{ij})$, where d_{ij} is the degree of node i if $i = j$, and is 0 otherwise. The Laplacian of G is the matrix $L = D - A$. The eigenvalues of L are $0 = \Lambda_1 \leq \Lambda_2 \leq \cdots \leq \Lambda_n$, and $\Lambda_2 = \Omega(n^{-2})$, if G is connected.

Next we address the efficiency of the load balancing protocol, an important performance parameter that was not addressed in most previous work. That is, we relate the waiting time of jobs to the topology of the network. Since protocol $\mathcal{P}$ treats all jobs equally regardless of their ages, it cannot guarantee efficient delivery time for individual packets. To bound the waiting time of jobs in the system we augment the protocol with a distributed version of the *first-come-first-served* (FCFS) queueing discipline, requiring that a node always respect the ages of its jobs in the load balancing step, and always consume its oldest job in the load consumption step (see Section 3.2 for details). We denote this version of $\mathcal{P}$ by $\mathcal{P}^*$.

Theorem 3.1.2. *Suppose that we run the system under one of the two adversarial-generator models, using protocol $\mathcal{P}^*$. Let W_t be the wait of a job that arrived at time t. For any constant $c > 1$ there is a constant $\kappa = \kappa(c)$, such that*

1. $\lim_{t\to\infty} \mathbf{Pr}(W_t \le \kappa\gamma^{-1}\ln n) \ge 1 - n^{-c}$.
2. $\lim_{t\to\infty} \mathbf{E}[W_t] = O(\gamma^{-1}\ln n)$.

The above bounds hold without the limits if the system starts with no load.

In particular, for bounded degree regular expanders $\gamma = \Theta(1)$, and the diameter of the graph is $O(\log n)$, so for these graphs the above result is optimal.

3.1.2 General Stochastic Adversarial Model

Here we generalize the input model that we presented in the previous section. In particular, we consider the type of adversaries proposed by Borodin et al. in [17]. Following that work, we define a $(w, \lambda n)$ input adversary as a process that inserts jobs in the system subject to the condition that for every sequence of w consecutive time steps, the total inserted load is at most λnw. This allows the adversary to insert more jobs at some time steps, as long as the total load in windows of size w is bounded. An extension is a $(w, \lambda n)$ stochastic adversary, whose input load is a random variable, with the property that the expected injected load during any sequence of w consecutive time steps is bounded by λnw, and additionally, for some $p > 2$, the pth moment of the new load is bounded (see [17] for a detailed discussion).

For these adversaries, we derive asymptotic (with respect to the network size) bounds on both the expected total load and the waiting time of a job in the system. As before, we wish to augment the expected performance results with high-probability bounds. In this case, the stochastic adversarial model is too powerful, since it allows for large bursts of load to occur in certain time steps with decent probability. Therefore we need to place additional restrictions on the adversary if we wish to derive high-probability results. To this end, we introduce a constrained version of the stochastic adversary by enforcing a large-deviation–type bound for the incoming load, similar to a Chernoff bound. We call these adversaries *strongly bounded*.

Formally, let $\mathcal{A}$ be an adversary that is injecting load into the system. Let I_t be the load injected by the adversary during time step t.

Definition 3.1.1. *We say that $\mathcal{A}$ is a (λ, w, p, M)* stochastic adversary *(where we assume that w is a positive integer) if the following conditions hold for any time t and any event $\mathcal{H}$ determined entirely by information about the system at or before time t.*

1. $\mathbf{E}\left[\sum_{i=1}^{w} I_{t+i} \mid \mathcal{H}\right] \leq \lambda n w$.
2. $\mathbf{E}\left[\left(\sum_{i=1}^{w} I_{t+i}\right)^p \mid \mathcal{H}\right] \leq M n^p w^p$.

In addition, we say that $\mathcal{A}$ is strongly bounded *if it also satisfies the following condition.*

3. *There is a constant $\alpha > 0$ and a constant $\beta \geq 1$ such that for any $\epsilon > 0$*

$$\mathbf{Pr}\left(\sum_{i=1}^{w} I_{t+i} > (1+\epsilon)\lambda n w \mid \mathcal{H}\right) \leq e^{-\alpha\lambda n w \epsilon^{\beta}}.$$

Throughout the chapter, we always assume that λ and p do not depend on n, while w and M may be functions of n.

We first show that the system (using $\mathcal{P}$) is stable under the stochastic adversary model (for $\lambda < 1$ and $p > 2$). That is, the expected total load in the system is bounded with respect to time. The following theorem, proven in Section 3.4.2 is the analog of Theorem 3.1.1.

Theorem 3.1.3. *Suppose that we run the system with a (λ, w, p, M) adversary, where $\lambda < 1$ and $p > 2$, using protocol $\mathcal{P}$ (described in Section 3.2). Let L_t be the load of the system at time t. Then the system is stable and*

$$\limsup_{t\to\infty} \mathbf{E}[L_t \mid L_0] = O(\gamma^{-1} n(w + \ln n)(1+M)^{3p}) \qquad \text{as } n \to \infty.$$

In addition, if the adversary is strongly bounded, then for any constant $c > 0$ there is a constant $\kappa = \kappa(c)$, such that for sufficiently large n,

$$\liminf_{t\to\infty} \mathbf{Pr}(L_t \leq \kappa\gamma^{-1} n(w + \ln n)) \geq 1 - n^{-c}.$$

The above bounds hold without the limits if the system starts with no load.

Once again, we desire to show that a new job does not wait into the system for too long before being consumed. The following theorem provides such guarantees, for protocol $\mathcal{P}^*$.

Theorem 3.1.4. *Suppose that we run the system with a (λ, w, p, M) adversary, where $\lambda < 1$ and $p > 2$, using protocol $\mathcal{P}^*$ (described in Section 3.2). Let W_t be the wait of a job that arrived at time t. Then*

$$\limsup_{t\to\infty} \mathbf{E}[W_t \mid L_0] = O(\gamma^{-1}(w + \ln n)(1+M)^{3p}) \qquad \text{as } n \to \infty.$$

In addition, if the adversary is strongly bounded, then for any constant $c > 0$ there is a constant $\kappa = \kappa(c)$, such that for sufficiently large n,

$$\liminf_{t\to\infty} \mathbf{Pr}(W_t \leq \kappa\gamma^{-1}(w + \ln n)) \geq 1 - n^{-c}.$$

The above bounds hold without the limits if the system starts with no load.

Notice that the generator adversary models of Section 3.1.1 are special cases of a strongly bounded (λ, w, p, M) adversary, with $w = 1$, $p = 3$ and the corresponding M being 0 for the deterministic and a constant for the stochastic. The deterministic model is clearly strongly bounded, while a standard Chernoff bound establishes the claim for the stochastic model. Therefore we can deduce the results of the previous section from the above theorems. In order, however, to illustrate the technique of applying renewal theory for long-term analysis, we consider the previous model separately. It seems however that renewal theory is not sufficient to handle the more general adversaries presented here, therefore, in Section 3.4.2 we use a more sophisticated martingale technique.

3.2 Protocol

If we were simply interested in a stability result we could use the protocol studied in [32]: in each step, nodes are matched randomly with adjacent neighbors in the network, and if node v is matched with node u, they equalize their load subject to integer rounding. The details of the protocol (which we call protocol $\mathcal{P}$) are given below.

1. **Matching Phase:**
    - **For** each node i
        - Node i inserts each incident edge (i, j) into a set S with probability $\frac{1}{8\max(d_i, d_j)}$, where d_i is the degree of node i.
        - Node i removes edge (i, j) from S if some edge (i, k) or (j, k) is in S, with $k \neq i, j$.
    - Let the matching M consist of the remaining edges in S.
2. **Transfer Phase:**
    - **If** $(i, j) \in M$
        - i and j equalize their loads so that, say, i gets load $\lceil(\ell_t(i) + \ell_t(j))/2\rceil$ and j gets load $\lfloor(\ell_t(i) + \ell_t(j))/2\rfloor$, where $\ell_t(i)$ is the load of processor i in the beginning of the step.

To bound the waiting time of jobs in the system we need to augment the above protocol with a distributed version of the *first-in-first-out* queueing discipline. It is not enough to require that a node consume the oldest job in its queue; we also need to consider the jobs' ages in the load balancing procedure. In particular, when u and v are matched they should not only equalize their total load but also equalize (up to rounding) the load that they have above any given age. A simple method to maintain this property is given by protocol $\mathcal{P}^*$ below. Note that protocol $\mathcal{P}^*$ is a special case of protocol $\mathcal{P}$.

1. **Matching Phase:**
   - **For** each node i
     - Node i inserts each incident edge (i, j) into a set S with probability $\frac{1}{8\max(d_i, d_j)}$, where d_i is the degree of node i.
     - Node i removes edge (i, j) from S if some edge (i, k) or (j, k) is in S, with $k \neq i, j$.
   - Let the matching M consist of the remaining edges in S.
2. **Transfer Phase:**
   - **If** $(i, j) \in M$
     - Let $J^i_1, J^i_2, \ldots, J^i_{\ell_t(i)}$ (where $\ell_t(i)$ is the load of processor i in the beginning of the step), and let $J^j_1, J^j_2, \ldots, J^j_{\ell_t(j)}$ be the jobs in the queues of nodes i and j, respectively, sorted from oldest to newest.
     - Node i sends jobs $J^i_2, J^i_4, \ldots, J^i_{2\lfloor \ell_t(i)/2 \rfloor}$ to node j. Similarly, node j sends jobs $J^j_2, J^j_4, \ldots, J^j_{2\lfloor \ell_t(i)/2 \rfloor}$ to node i.
     - Node i merges jobs $J^i_1, J^i_3, J^i_5, \ldots$ with $J^j_2, J^j_4, J^j_6, \ldots$ in its queue, so that, finally, if a job J is older than a job J', then job J is in the queue before job J'.
     - Similarly, node j merges jobs $J^j_1, J^j_3, J^j_5, \ldots$ with $J^i_2, J^i_4, J^i_6, \ldots$ in its queue, so that, finally, if a job J is older than a job J', then job J is in the queue before job J'.

3.3 Analysis of the Static Case

We first analyze our load balancing protocol in a static setting in which some initial load is placed on the n processors, and load is moved between processors until the loads in all the processors are approximately equal. No new load is added to or removed from the system throughout the execution of the protocol. Later we will use the results for the static setting to analyze the dynamic one.

Our analysis of the static case is based on coupling the execution of our protocol with the nonintegral protocol studied in [32]. The only difference between the two protocols is that in the nonintegral protocol the load is equally distributed between the two matched processors with no rounding. We consider two copies of the network starting with the same initial distribution, one using our protocol and the other using the nonintegral protocol. The two processes are coupled so that they use the same matching at every time step.

Fix any initial distribution of K tokens to the nodes in V, and let $\bar{\ell} = K/n$. For each time step $t \geq 0$ and $u \in V$, let $\ell_t(u)$ be the number of tokens at u at the end of step t of the original, integral protocol, and let $\ell'_t(u)$ be the number of tokens at u at the end of step t in the nonintegral copy of the protocol. Also, for each time step $t \geq 0$, let $\Phi'_t = \sum_{v \in V}(\ell'_t(v) - \bar{\ell})^2$. Note that if the total load in the system is K, then for any $t \geq 0$, $\Phi'_t \leq K^2$. Notice that Φ'_t corresponds to the variance of the load on the nodes, and it is easy to see that it is nonincreasing with time, which, intuitively, means that successive applications of the load balancing protocol even out the distribution of the tokens to the nodes.

Recall that $\gamma = \Lambda/16\Delta$. The performance of the nonintegral protocol was analyzed in terms of γ in [32]. The relevant result is the following lemma, which follows immediately from Theorem 1 in that work.

Lemma 3.3.1. *For any $t \geq 0$,*

$$\mathbf{E}[\Phi'_{t+1} \mid \Phi'_t] \leq (1-\gamma)\Phi'_t.$$

Adapting the technique in [32] we can prove the following static load balancing result for the nonintegral copy.

Lemma 3.3.2. *If $t \geq \gamma^{-1}(2\ln K + c\ln n)$, then the probability that there is some $v \in V$ with $|\ell'_t(v) - \bar{\ell}| > 1$ is at most $1/n^c$.*

Proof. By Lemma 3.3.1 we get

$$\mathbf{E}[\Phi'_t] = \mathbf{E}[\,\mathbf{E}[\Phi'_t \mid \Phi'_{t-1}]\,] \leq (1-\gamma)\mathbf{E}[\Phi'_{t-1}].$$

By applying the same argument t times we get

$$\begin{aligned}\mathbf{E}[\Phi'_t] &= (1-\gamma)^t\Phi'_0 \\ &\leq \Phi'_0 e^{-\gamma t} \\ &\leq e^{-c\ln n},\end{aligned}$$

where in the last inequality we used the facts that $\Phi'_0 \leq K^2$ and $t \geq \gamma^{-1}(2\ln K + c\ln n)$. Applying Markov's inequality yields $\mathbf{Pr}(\Phi'_t > 1) < n^{-c}$. ☐

We will now tie the performance of our protocol to the performance of the nonintegral copy.

Lemma 3.3.3. *For any $t \geq 0$ and any $v \in V$, $|\ell_t(v) - \ell'_t(v)| \leq t/2$, regardless of the chosen matchings and the rounding decisions in the original protocol.*

Proof. The proof is by induction on $t \geq 0$. If $t = 0$, then $\ell_t(v) = \ell'_t(v)$ for every $v \in V$, because no randomness has been introduced into the process yet. For the induction step, suppose that the lemma holds for time t. Choose any $v \in V$. If v is not matched at time $t+1$, then $\ell_{t+1}(v) = \ell_t(v)$ and $\ell'_{t+1}(v) = \ell'_t(v)$, so the lemma holds by the induction hypothesis. Otherwise, v is matched to some vertex u at time $t+1$. In this case, for any rounding choice, $(\ell_t(u) + \ell_t(v) - 1)/2 \leq \ell_{t+1}(v) \leq (\ell_t(u) + \ell_t(v) + 1)/2$. Also, $\ell'_{t+1}(v) = (\ell'_t(u) + \ell'_t(v))/2$. These observations give

$$\begin{aligned}|\ell_{t+1}(v) - \ell'_{t+1}(v)| &\overset{(a)}{\leq} \frac{1}{2} + \left|\frac{\ell_t(u)+\ell_t(v)}{2} - \frac{\ell'_t(u)+\ell'_t(v)}{2}\right| \\ &\overset{(b)}{\leq} \frac{1}{2} + \frac{1}{2}|\ell_t(u) - \ell'_t(u)| + \frac{1}{2}|\ell_t(v) - \ell'_t(v)| \\ &\overset{(c)}{\leq} \frac{1}{2}\cdot\frac{t}{2} + \frac{1}{2}\cdot\frac{t}{2} + \frac{1}{2} \\ &= \frac{t+1}{2},\end{aligned}$$

where (a) follows from previous observations, (b) follows from the triangle inequality, and (c) follows from the induction hypothesis. ☐

Putting Lemmata 3.3.2 and 3.3.3 together yields the following theorem.

Theorem 3.3.1. *Let $\hat{t} = \gamma^{-1}(2\ln K + c\ln n)$. Then, for any $t \geq \hat{t}$, the probability that there is some $v \in V$ with $|\ell_t(v) - \bar{\ell}| > 1 + \frac{\hat{t}}{2}$ is at most n^{-c}.*

Proof. We first prove the theorem for the case $t = \hat{t}$. Regardless of the matchings generated in the first $\hat{t}$ steps, $|\ell_{\hat{t}}(v) - \ell'_{\hat{t}}(v)| \leq \hat{t}/2$ for every $v \in V$ by Lemma 3.3.3. With probability at least $1 - 1/n^c$, $|\ell'_{\hat{t}}(v) - \bar{\ell}| \leq 1$ for all $v \in V$ by Lemma 3.3.2. Adding these inequalities proves the theorem for the case $t = \hat{t}$. To extend the result for the case $t > \hat{t}$, notice that the sequence $\{\max_{v\in V}|\ell_t(v) - \bar{\ell}|\}_{t\geq\hat{t}}$ is nonincreasing. □

3.4 Analysis of the Dynamic Case

In order to analyze the long-term performance of the system, we split the time into *epochs* of a fixed length T_E (to be defined later). We analyze each epoch in Theorem 3.4.1, which is the key ingredient in showing the stability and waiting-time properties of the system. Notice that the first part of the theorem, which we will apply for the stability result, holds for any protocol obeying the rules of $\mathcal{P}$, while the second part, which will be applied for the waiting-time guarantees, uses protocol $\mathcal{P}^*$.

Theorem 3.4.1. *For any constant $c > 0$ and load $\Theta = \Theta_n > 0$, consider an epoch of length $T_E = T_D + T_C$, such that*

$$T_D \geq \gamma^{-1}(2\ln\Theta + c\ln n) \qquad \textit{and} \qquad T_C \geq \frac{\Theta}{n} + \frac{T_D}{2} + 1.$$

1. *Running protocol $\mathcal{P}$, if at time τ the load is L_τ, then the system consumes at least $\min\{L_\tau, \Theta\}$ tokens in the next T_E steps with probability at least $1 - n^{-c}$.*

2. *Running protocol $\mathcal{P}^*$, if at time τ the load is bounded by Θ, then with probability at least $1 - n^{-c}$, all the jobs that exist in the system at time τ will be consumed by time $\tau + T_E$.*

Proof. For the purpose of the analysis we split the epoch into two parts. In the first T_D steps we focus on the distribution of load between the processors, and in the remaining T_C steps we focus on the consumption of load by the processors (although load is consumed throughout the whole execution by processors that have load).

We start by proving the first part of the theorem; a modification of that argument gives the second part. To analyze the distribution of load between the processors, we couple the actual execution of the protocol in the first T_D steps with an execution of the protocol in a static setting that starts with a total load of exactly Θ and does not generate or consume any jobs. We refer to the actual execution of the protocol as the *dynamic copy* and the static execution as the *static copy*.

To formulate the coupling we color all the tokens (jobs) in the dynamic copy at time τ by red and blue. A subset of $M = \min\{L_\tau, \Theta\}$ tokens in the system at time τ is colored red, and the rest are colored blue. We now place Θ tokens in the static copy so that each node in the static copy starts the process with at least as many tokens as the number of red tokens in the corresponding node of the dynamic copy. New tokens that arrive through the execution of the dynamic copy are colored blue.

The executions of the two copies are coupled so that they use the same matching in each step and the same rounding decisions. When we equalize (up to one) the load between two vertices in the dynamic execution, we also equalize (up to one, using the same rounding rule) the number of red tokens in the two nodes, keeping the total number of red tokens in the two nodes as before (this can be achieved by recoloring some tokens). Finally, after each matching we recolor the tokens in the nodes again, preserving the total number of red tokens in each queue, but putting all the red tokens in a queue ahead of all the blue tokens.

Lemma 3.4.1. *At any time $\tau \leq t \leq \tau + T_D$ the number of red tokens in each node of the dynamic copy is bounded by the number of tokens in the corresponding node in the static copy.*

Proof. We begin with some intuition. Initially the coloring of the tokens in the dynamic copy is such that the claim holds. New tokens that enter the system in the dynamic copy are colored blue, while no tokens in the static copy are removed. Furthermore, the matching operations in both copies are coupled, so we expect that if one node in the static copy gives half of its tokens to a neighbor, the same node in the dynamic copy will do the same with its red tokens. Therefore, we expect that the claim will hold at the next step. By induction, the lemma follows.

We now proceed formally by induction on t. The claim is true for $t = \tau$ by the construction of the static copy. Assume that the claim holds after the execution of step $t-1$, and consider the load of node u after the execution of step t. If u was not part of a matching in step t, then the load of the static copy did not change. The number of red tokens of the dynamic copy either did not change, or was reduced by 1 if a red token was consumed. In both cases, using the induction hypothesis, the number of red tokens in the dynamic copy after the execution of step t is bounded by the number of tokens in the static copy of u.

Assume now that node u was matched with node v during step t. Note that every time step in the process can be decomposed into three substeps: the insertion substep (where new load is created in the dynamic copy), the balancing substep, (where the matching is chosen and the nodes in both copies equalize their loads), and the consumption substep (where the nonempty nodes in the dynamic copy consume a job). With this in mind, we define some new variables:

- $\ell_t^i(u)$, $\ell_t^b(u)$, $\ell_t^c(u)$ are the total number of tokens at node u in the dynamic copy immediately following the insertion, balancing, and consumption substeps of the tth time step, respectively.
- $r_t^i(u)$, $r_t^b(u)$, $r_t^c(u)$ are the number of red tokens at node u in the dynamic copy at that time.
- $s_t^i(u)$, $s_t^b(u)$, $s_t^c(u)$ are the number of tokens at node u in the static copy at that time.

We analyze each substep separately.

Initially, by the induction hypothesis, we have

$$r_{t-1}^c(u) \leq s_{t-1}^c(u), \qquad r_{t-1}^c(v) \leq s_{t-1}^c(v). \tag{3.1}$$

After the insertion substep, we color the new tokens blue so the number of red tokens remains the same:

$$r_t^i(u) = r_{t-1}^c(u), \qquad r_t^i(v) = r_{t-1}^c(v). \tag{3.2}$$

The static copy does not accept new tokens so

$$s_t^i(u) = s_{t-1}^c(u), \qquad s_t^i(v) = s_{t-1}^c(v). \tag{3.3}$$

So from Relations (3.1), (3.2), and (3.3) we get that

$$r_t^i(u) \leq s_t^i(u), \qquad r_t^i(v) \leq s_t^i(v). \tag{3.4}$$

Notice also that the number of red tokens is bounded by the total number of tokens, so

$$r_t^i(u) \leq \ell_t^i(u), \qquad r_t^i(v) \leq \ell_t^i(v). \tag{3.5}$$

After the balancing substep, we assume, without loss of generality, that the rounding is such that

$$\ell_t^b(u) = \left\lceil \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rceil, \qquad \ell_t^b(v) = \left\lfloor \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rfloor. \tag{3.6}$$

The results of the analysis of the static case in Section 3.3 hold for an arbitrary rounding procedure. Thus, in the static copy we perform the rounding so that

$$s_t^b(u) = \left\lceil \frac{1}{2}(s_t^i(u) + s_t^i(v)) \right\rceil, \qquad s_t^b(v) = \left\lfloor \frac{1}{2}(s_t^i(u) + s_t^i(v)) \right\rfloor. \tag{3.7}$$

Finally, we recolor the tokens in the dynamic copy (we swap colors between some tokens), so that

$$r_t^b(u) = \left\lceil \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rceil, \qquad r_t^b(v) = \left\lfloor \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rfloor. \tag{3.8}$$

Then, using Relation (3.4), we get

$$r_t^b(u) = \left\lceil \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rceil \leq \left\lceil \frac{1}{2}(s_t^i(u) + s_t^i(v)) \right\rceil = s_t^b(u), \tag{3.9}$$

and

$$r_t^b(v) = \left\lfloor \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rfloor \leq \left\lfloor \frac{1}{2}(s_t^i(u) + s_t^i(v)) \right\rfloor = s_t^b(v). \tag{3.10}$$

Notice that

$$r_t^b(u) + r_t^b(v) = \left\lceil \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rceil + \left\lfloor \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rfloor = r_t^i(u) + r_t^i(v),$$

which means that we haven't changed the number of red tokens, and notice also that by using Relation (3.5) we get that

$$r_t^b(u) = \left\lceil \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rceil \leq \left\lceil \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rceil = \ell_t^b(u)$$

and

$$r_t^b(v) = \left\lfloor \frac{1}{2}(r_t^i(u) + r_t^i(v)) \right\rfloor \leq \left\lfloor \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rfloor = \ell_t^b(v),$$

which ensure that the recoloring is valid (i.e., for both u and v, the number of the red tokens is not more than the total number of tokens).

Finally, we recolor the tokens again (preserving the number of red tokens at each node), so that every red token in a queue is ahead of every blue token in that queue.

After the consumption substep, since some red tokens may be consumed, we get that

$$r_t^c(u) \leq r_t^b(u), \qquad r_t^c(v) \leq r_t^b(v). \tag{3.11}$$

In the static copy there are no tokens being consumed, so

$$s_t^c(u) = s_t^b(u), \qquad s_t^c(v) = s_t^b(v). \tag{3.12}$$

Hence, from Relations (3.9), (3.10), (3.11), and (3.12) we can finally prove the induction hypothesis for the tth step:

$$r_t^c(u) \leq s_t^c(u), \qquad r_t^c(v) \leq s_t^c(v).$$

□

We now turn to studying the consumption phase. Applying Theorem 3.3.1 to the execution of the static copy, we see that at time $\tau + T_D$ (the reader can verify that the condition of Theorem 3.3.1 holds for $t = T_D$ and $K = \Theta$), with probability at least $1 - n^{-c}$ no node of the static copy has more than $T_D/2 + \Theta/n + 1 \leq T_C$ tokens. Thus, by Lemma 3.4.1, with the same probability, no node in the dynamic copy has more than T_C red tokens.

We continue to run the coupling from time $\tau + T_D$ with this new coloring. When balancing between two nodes, we recolor the tokens in exactly the same way as in the distribution phase (see Lemma 3.4.1), so that, in particular, if a node has some red tokens in its queue, then these tokens are at the front of the node's queue, before the blue tokens. In this case, notice that after a balancing substep, the maximum (over all the nodes in the graph) number of tokens does not increase. In the consumption substep the maximum (again, over all the nodes in the graph) number of red tokens decreases by 1, since the red tokens are at the front of their queues. Since initially at most T_C red tokens are at a node with probability at least $1 - n^{-c}$, we conclude that with probability $1 - n^{-c}$, all the red tokens (which means at least $M = \min\{L_\tau, \Theta\}$ tokens) are consumed in this epoch. This completes the proof of the first part of the theorem.

The proof of the second part of the theorem is almost identical to the first one. We color initially all the $L_\tau \leq \Theta$ tokens of the dynamic copy at time τ red, while all the subsequent ones are colored blue, and again we consider the coupled static copy. Since now we are interested in the identities of the jobs that are being consumed, we do not allow recolorings of the tokens.

Notice though that protocol $\mathcal{P}^*$ ensures that during the whole epoch, if a node has both red and blue tokens in its queue, the red tokens are before the blue ones. Moreover, the Transfer Phase of $\mathcal{P}^*$ ensures that when node u is matched with node v, they balance their red tokens (up to a difference of 1). Without loss of generality, we assume that the rounding is such that if the ensemble of the red tokens in the two nodes is odd, then node u ends up with one more red token. Hence equations (3.8) remain true. We then perform the rounding in the static copy to ensure that equations (3.7) remain true as well.

As an aside, notice that it may be the case that after the balancing substep, node v's *total* load is exactly one larger than u's total load, in which case Equations (3.6) may become

$$\ell_t^b(u) = \left\lfloor \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rfloor, \qquad \ell_t^b(v) = \left\lceil \frac{1}{2}(\ell_t^i(u) + \ell_t^i(v)) \right\rceil.$$

Here, however, we analyze only the distribution of the red tokens, so the analysis is not affected by this fact.

Everything else is identical to the first part, and by the same reasoning we conclude that by the end of the distribution phase, for every node u, the number of red tokens in the dynamic copy is bounded by the number of tokens in the static copy, with probability at least $1 - n^{-c}$, which, by applying Theorem 3.3.1, implies that every node has at most T_C red tokens.

Then, as in the first part, we can conclude that by the end of the consumption phase all the red tokens—which are exactly the tokens that were in the system in the beginning of the epoch—are consumed. $\square$

An immediate consequence of the analysis of the second part of Theorem 3.4.1 is the following lemma, which gives a bound on the expected time needed until the initial load is distributed. We make use of the lemma in order to bound the expected waiting time.

Lemma 3.4.2. *Assume that we are given c, Θ, T_D, T_C, satisfying the conditions of Theorem 3.4.1, and assume that we balance according to protocol $\mathcal{P}^*$. If at time τ the load is bounded by Θ, then the expected time needed until every node in the system has at most T_C of the initial jobs is bounded by $2T_D$ for sufficiently large n. At every time, when a node has some of the initial load, this is at the head of its queue.*

Proof. The analysis is an extension of the proof of the second part of Theorem 3.4.1. We color the initial load at time τ red, all the new incoming load blue, and we let T be the time until every node in the network has at most T_C red tokens. We perform the same coupling as in Theorem 3.4.1 from time τ until time $\tau + T$. During this whole period the number of red tokens in the dynamic copy is bounded by the number of tokens in the static copy. Since the red tokens are the oldest tokens in the system, protocol $\mathcal{P}^*$ ensures that if a node has some red tokens and some blue tokens, the red tokens are always in front of the blue ones.

Thus, by the proof of Theorem 3.4.1, part 2, we get that $T \leq T_C$ with probability at least $1 - n^{-c}$. It follows that $\lceil T/T_D \rceil$ is stochastically dominated by a geometric random variable with parameter $1 - n^{-c}$, so

$$\mathbf{E}[T] \leq \frac{1}{1 - n^{-c}} T_D \leq 2T_D$$

for sufficiently large n. $\square$

3.4.1 Adversarial-Generator Model

We first prove that for any constant $\lambda < 1$ and any connected network topology, the load-distribution protocol is stable. We then use the structure developed here to analyze the waiting time of individual jobs.

Having worked out the constants that appear in the proof, we give here the values of the main ones, for easy reference. $\epsilon = 1 - \lambda$, $c > 3$, $c_3 = c + 9$, $c_1 = \frac{2\lambda(2c_3+1)}{(1+\epsilon)^{-1}-\lambda}$, $c_4 = c_1 + \frac{c_3}{2} + 1$, $c_2 = c_3 + c_4$, $c_6 = (1+\epsilon)\lambda c_2$, $c_5 = c_1 - c_6$, $\rho = c_5/(c_2 + c_5)$.

As we mentioned, in order to analyze the behavior of the system over time, we partition time into *epochs*, where each epoch has $T_E = c_2\gamma^{-1}\ln n$ steps. We say that the system is in a *good* state at the end of an epoch if the total load in the system is less than $\Theta = c_1\gamma^{-1}n\ln n$, otherwise the system is in a *bad* state.

We will define a two state renewal process that alternates between states G and B. The crux of the proof is to show that the distribution of the length of a B segment in the renewal process stochastically dominates the distribution of the number of successive epochs in which the original system is in a bad state, conditioned on its past. On the other hand, the distribution of the number of successive epochs in which the original system is in a good state, conditioned on its past, stochastically dominates the length of time that the renewal process is in the G state. Once this relation is established, the analysis of the renewal process implies the stability and waiting-time results for the load balancing protocol.

Analysis of One Epoch

We say that an epoch is *successful* if one of the following conditions holds:

- The total system load is less than $\Theta = c_1\gamma^{-1}n\ln n$ immediately after the epoch finishes.
- The total load of the system decreases by at least $c_5\gamma^{-1}n\ln n$ during the epoch.

Lemma 3.4.3. *The probability that an epoch is successful, conditioned on all events in the past, is at least* $1 - 2n^{-c}$.

Proof. Recall that the epoch length is $c_2\gamma^{-1}\ln n$. The expected number of new tokens arriving during the epoch is at most $\lambda c_2\gamma^{-1}n\ln n$, and by applying a Chernoff bound (for the version of the randomized adversary) we deduce that with probability at least $1 - n^{-c}$ this number is bounded by $c_6\gamma^{-1}n\ln n = (1+\epsilon)\lambda c_2\gamma^{-1}n\ln n$.

Let τ be the time that the epoch begins, so the load of the system at this time is L_τ. We apply Theorem 3.4.1, part 1, for $\Theta = c_1\gamma^{-1}n\ln n$, $T_E = c_2\gamma^{-1}\ln n$, $T_D = c_3\gamma^{-1}\ln n$, and $T_C = c_4\gamma^{-1}\ln n$ (the reader can verify that the conditions of Theorem 3.4.1 hold for sufficiently large n), and we get that during the epoch the system consumes at least $\min\{L_\tau, \Theta\}$ tokens, with probability at least $1 - n^{-c}$.

We distinguish now between two cases.

1. If $L_\tau \leq \Theta = c_1\gamma^{-1}n\ln n$, then with probability at least $1 - n^{-c}$, the initial L_τ tokens are consumed during this epoch, and with probability at least $1-n^{-c}$ no more than $c_6\gamma^{-1}n\ln n < c_1\gamma^{-1}n\ln n$ tokens join the system during this epoch. Thus, with probability at least $1-2n^{-c}$ the epoch is successful since it ends with less than $c_1\gamma^{-1}n\ln n$ total load.

2. If $L_\tau > \Theta = c_1\gamma^{-1}n\ln n$, then with probability at least $1-2n^{-c}$ at least $c_1\gamma^{-1}n\ln n$ tokens are consumed during this epoch and no more than $c_6\gamma^{-1}n\ln n$ new tokens arrive during this epoch. In this case, the epoch is successful, since the total load decreases by at least $c_1\gamma^{-1}n\ln n - c_6\gamma^{-1}n\ln n = c_5\gamma^{-1}n\ln n$.

Note that the probability space considered here is defined by the distribution of the matchings performed during this epoch, and by the arrival of new load (and not by the adversary's placement of the generators). So, in this probability space, the event that the epoch is successful is independent of any event in the past. □

Basic Renewal Theory

In this section we provide some basic background for renewal processes, that is needed for the analysis of the process. For more information refer for example to [68, Chapter 3].

Let $\{X_n, n = 1, 2, \dots\}$ be a sequence of nonnegative independent random variables with a common distribution function F. Let $S_0 = 0$, and for $n \geq 1$, $S_n = \sum_{i=1}^{n} X_i$; finally define $N(t) = \sup\{n : S_n \leq t\}$. The process $\{N(t), t \geq 0\}$ is a *Renewal Process*. We say that a renewal occurs at time t, if $S_n = t$ for some n. Since the interarrival times X_i are independent and identically distributed, it follows that after each renewal the process starts over again.

We say that a nonnegative random variable X (or the corresponding distribution function) is *lattice* (or periodic) if there exists a $d \geq 0$ such that $\sum_{n=0}^{\infty} \mathbf{Pr}(X = nd) = 1$. We are interested in renewal processes that are nonlattice.

Let $Z(t) = t - S_{N(t)}$ be the *age* at time t, that is, the time that has elapsed from the last renewal before t until t. We can compute the distribution of $Z(t)$ as $t \to \infty$ by Lemma 3.4.4 (see [68, Corollary 3.13]).

Lemma 3.4.4. *If F is not lattice, then*

$$\lim_{t\to\infty} \mathbf{Pr}(Z(t) \leq z) = \frac{1}{\mathbf{E}[X_1]} \int_0^x [1 - F(y)]\mathrm{d}y.$$

An *Alternating Renewal Process* can be in one of two states, *on* or *off*. Initially, it is on and it remains on for a time X_1; it then goes off and remains off for a time Y_1; it then goes on for a time X_2, then off for a time Y_2, and so on. We assume, moreover, that the X_i's have the same distribution, that the Y_i's have the same distribution, and that all the X_i's and the Y_i's are independent of each other. Let $P(t)$ be the probability that the system is on at time t. Then, Lemma 3.4.5 (repeating [68, Proposition 3.11]) gives the probability that at some time, in the limit, the system is found in state on.

Lemma 3.4.5. *If* $\mathbf{E}[X_1 + Y_1] < \infty$ *and if* $X_1 + Y_1$ *is nonlattice, then*

$$\lim_{t\to\infty} P(t) = \frac{\mathbf{E}[X_1]}{\mathbf{E}[X_1] + \mathbf{E}[Y_1]}.$$

Stability

We now define the two state renewal process that dominates the execution of our protocol. The process alternates between states G and B. Let $T(G)$ be the length of a G segment and $T(B)$ be the length of a B segment. To apply Lemmata 3.4.4 and 3.4.5 we define these distributions on continuous time. For any $x \geq 1$, let

- $\mathbf{Pr}(T(G) \geq x) = (1 - 2n^{-c})^{(x-1)}$ and
- $\mathbf{Pr}(T(B) \geq x) = n^{-\rho(c-1)(x-1)}$.

Note that these functions are monotonically decreasing in x,

$$\begin{aligned}\mathbf{E}[T(G)] &= 1 + \int_1^\infty (1 - 2n^{-c})^{(x-1)} \mathrm{d}x \\ &\geq \sum_{i=1}^{\infty} (1 - 2n^{-c})^{(i-1)} = \frac{1}{2} n^c,\end{aligned}$$

and

$$\begin{aligned}\mathbf{E}[T(B)] &= 1 + \int_1^\infty n^{-\rho(c-1)(x-1)} \mathrm{d}x \\ &\leq 1 + \sum_{i=1}^{\infty} n^{-\rho(c-1)(i-1)} = 1 + \frac{1}{1 - n^{-\rho(c-1)}}.\end{aligned}$$

Now consider the sequence of epochs in the execution of our protocol. Recall that at the end of an epoch the system is in a *good* state if the total load in the system is bounded by $c_1\gamma^{-1} n \ln n$, otherwise the system is in a *bad* state.

Lemma 3.4.6. *Let* $T(good)$ *be the distribution of the number of epochs in a given segment in which the system is in a good state, and* $T(bad)$ *be the number of epochs in a given segment in which the system is in a bad state. Conditioned on all events before the start of the segment, for any* $i \geq 1$,

1. $\mathbf{Pr}(T(good) \geq i) \geq \mathbf{Pr}(T(G) \geq i)$,
2. $\mathbf{Pr}(T(bad) \geq i) \leq \mathbf{Pr}(T(B) \geq i)$.

Proof. If the total load in the system at the start of an epoch is less than $c_1\gamma^{-1} n \ln n$, and the epoch was successful, then the total load in the system at the end of that epoch is also less than $c_1\gamma^{-1} n \ln n$. Therefore, once the system is in a good state it stays in a good state until at least the first unsuccessful epoch. The probability that an epoch is successful, conditioned on the past, is at least $1 - 2n^{-c}$, by Lemma 3.4.3. Thus,

$$\mathbf{Pr}(T(good) \geq i) \geq (1 - 2n^{-c})^{(i-1)} = \mathbf{Pr}(T(G) \geq i).$$

To bound the distribution of $T(\mathit{bad})$ we observe that if a segment of i epochs begins when the system is in a good state, and each of those i epochs finishes when the system is in a bad state, then at least ρi of those i epochs must have been unsuccessful. Otherwise, the total change in the system load over the course of those i epochs is at most $\rho i c_2 n \ln n - i(1-\rho)c_5 n \ln n \leq 0$, which implies that the last epoch finishes with the system being in a good state, yielding a contradiction.

The probability of at least ρi unsuccessful epochs in a sequence of i epochs, for n large enough, is bounded by

$$\begin{aligned}\binom{i}{\rho(i-1)}\left(\frac{2}{n^c}\right)^{\rho(i-1)} &\leq \left(\frac{ei}{\rho(i-1)}\right)^{\rho(i-1)}\left(\frac{2}{n^c}\right)^{\rho(i-1)} \\ &= \left(n^{-c}\cdot\frac{2e}{\rho}\cdot\frac{i}{i-1}\right)^{\rho(i-1)} \\ &\leq n^{-\rho(c-1)(i-1)}.\end{aligned}$$

Thus, for sufficiently large n,

$$\mathbf{Pr}(T(\mathit{bad}) \geq i) \leq n^{-\rho(c-1)(i-1)} = \mathbf{Pr}(T(B) \geq i).$$

□

We now use the stochastic dominance proven in Lemma 3.4.6 to analyze the performance of our protocol using renewal theory. Let $\mathcal{E}_t$ be the event "the system is in a bad state at time t." Then Lemmata 3.4.5 and 3.4.6 along with the fact that $T(B)$ and $T(G)$ have continuous distribution functions and therefore are not lattice, yield

Lemma 3.4.7.

$$\lim_{t\to\infty}\mathbf{Pr}(\mathcal{E}_t) \leq \frac{\mathbf{E}[T(B)]}{\mathbf{E}[T(B)]+\mathbf{E}[T(G)]} \leq \frac{\mathbf{E}[T(B)]}{\mathbf{E}[T(G)]} \leq 3n^{-c}$$

for large enough n.

We now turn to bounding the expected load in the system at time t, thus establishing the fact that the system is stable.

Let L_t denote the total load in the system at time t.

$$\begin{aligned}\mathbf{E}[L_t] &= \mathbf{E}[L_t \mid \overline{\mathcal{E}_t}]\mathbf{Pr}(\overline{\mathcal{E}_t}) + \mathbf{E}[L_t \mid \mathcal{E}_t]\mathbf{Pr}(\mathcal{E}_t) \\ &\leq c_1\gamma^{-1}n\ln n + \mathbf{E}[L_t \mid \mathcal{E}_t]\mathbf{Pr}(\mathcal{E}_t),\end{aligned} \tag{3.13}$$

where in the inequality we use the fact that in a good state the load of the system is bounded by $c_1\gamma^{-1}n\ln n$.

To compute

$$\mathbf{E}[L_t \mid \mathcal{E}_t] = \sum_{x=1}^{\infty}\mathbf{Pr}(L_t \geq x \mid \mathcal{E}_t)$$

we need a bound on the probability $\mathbf{Pr}(L_t \geq x \mid \mathcal{E}_t)$. Observe that for $i \geq 1$, if $L_t \geq (c_1+ic_2)\gamma^{-1}n\ln n$ then the system must have been in a bad state for at least the last $i-1$ epochs. Otherwise, by the beginning of the current epoch, the load would have been at most $(c_1 + (i-2)c_2)\gamma^{-1} n \ln n$, so at time t the load would be at most $(c_1 + (i-1)c_2)\gamma^{-1} n \ln n$. Thus, we need a bound on Z_t, the time from the start of the bad segment until t. Let Z'_t be the corresponding random variable in the renewal process, that is, the time from the start of a B segment until t, conditioned on t being in a B segment. With the same argument as in Lemma 3.4.6, we have

$$\mathbf{Pr}(Z_t \geq i-1 \mid \mathcal{E}_t) \leq \mathbf{Pr}(Z'_t \geq i-1 \mid \mathcal{E}_t).$$

Using that fact, applying Lemma 3.4.4, and using again the fact that $T(B)$ and $T(G)$ have continuous distribution functions and therefore are nonlattice, we compute for $i \geq 1$

$$\begin{aligned}\lim_{t\to\infty} \mathbf{Pr}(L_t \geq (c_1 + ic_2)\gamma^{-1} n \ln n \mid \mathcal{E}_t) &\leq \lim_{t\to\infty} \mathbf{Pr}(Z_t \geq i-1 \mid \mathcal{E}_t) \\ &\leq \lim_{t\to\infty} \mathbf{Pr}(Z'_t \geq i-1 \mid \mathcal{E}_t) \\ &= \frac{1}{\mathbf{E}[T(B)]} \int_{i-1}^{\infty} \mathbf{Pr}(T(B) \geq x) \mathrm{d}x \qquad (3.14) \\ &= \frac{1}{\mathbf{E}[T(B)]} \cdot \frac{n^{-\rho(c-1)(i-2)}}{\rho(c-1)\ln n} \\ &\leq n^{-\rho(c-1)(i-2)}\end{aligned}$$

where in the last inequality we used the fact that (trivially) $\mathbf{E}[T(B)] \geq 1$. We can also write

$$\begin{aligned}\mathbf{E}[L_t \mid \mathcal{E}_t] &= \sum_{x=1}^{\infty} \mathbf{Pr}(L_t \geq x \mid \mathcal{E}_t) \\ &\leq (c_1 + 3c_2)\gamma^{-1} n \ln n + \sum_{x=(c_1+3c_2)\gamma^{-1} n \ln n}^{\infty} \mathbf{Pr}(L_t \geq x \mid \mathcal{E}_t) \\ &= (c_1 + 3c_2)\gamma^{-1} n \ln n + \sum_{x=0}^{\infty} \mathbf{Pr}(L_t \geq (c_1 + 3c_2)\gamma^{-1} n \ln n + x \mid \mathcal{E}_t) \\ &= (c_1 + 3c_2)\gamma^{-1} n \ln n + \sum_{i=3}^{\infty} \sum_{j=0}^{c_2\gamma^{-1} n \ln n - 1} \mathbf{Pr}(L_t \geq (c_1 + ic_2)\gamma^{-1} n \ln n + j \mid \mathcal{E}_t) \\ &\leq (c_1 + 3c_2)\gamma^{-1} n \ln n + (c_2\gamma^{-1} n \ln n) \sum_{i=3}^{\infty} \mathbf{Pr}(L_t \geq (c_1 + ic_2)\gamma^{-1} n \ln n \mid \mathcal{E}_t).\end{aligned}$$

So, by making use of Inequality (3.14), we have

$$\begin{aligned}\lim_{t\to\infty} \mathbf{E}[L_t \mid \mathcal{E}_t] &\leq (c_1 + 3c_2)\gamma^{-1} n \ln n + (c_2\gamma^{-1} n \ln n) \sum_{i=3}^{\infty} n^{-\rho(c-1)(i-2)} \\ &\leq (c_1 + 4c_2)\gamma^{-1} n \ln n.\end{aligned}$$

Therefore, Relation (3.13) and Lemma 3.4.7 give, for sufficiently large n,

$$\begin{aligned}\lim_{t\to\infty} \mathbf{E}[L_t] &\leq \lim_{t\to\infty} \mathbf{E}[L_t \mid \overline{\mathcal{E}_t}]\mathbf{Pr}(\overline{\mathcal{E}_t}) + \mathbf{E}[L_t \mid \mathcal{E}_t]\mathbf{Pr}(\mathcal{E}_t) \\ &\leq c_1\gamma^{-1} n \ln n + 3n^{-c}(c_1 + 4c_2)\gamma^{-1} n \ln n \\ &\leq (c_1 + 1)\gamma^{-1} n \ln n.\end{aligned}$$

Thus, we prove the following.

Theorem 3.4.2. *The system is stable and as time tends to infinity the expected total load in the system is $O(\gamma^{-1} n \ln n)$.*

Note that, given that the system started with no load, or even just in a good state, the above bound holds for any t, not only in the limit. To see this, assume that at some point before time t the system had no more than $c_1\gamma^{-1}n \ln n$ load. Then the system can have load $c_1\gamma^{-1}n\ln n + ic_2\gamma^{-1}n\ln n$ at time t only if the system was in a bad state in the last i epochs up to time t. The probability of that event is bounded by $n^{-\rho i(c-1)}$. Thus,

$$\begin{aligned}\mathbf{E}[L_t] &\le c_1\gamma^{-1}n\ln n + c_2\gamma^{-1}n\ln n\sum_{i\ge1} n^{-\rho i(c-1)}\\ &= O(\gamma^{-1}n\ln n).\end{aligned}$$

Waiting Time

Having established that the system is stable, the next important performance parameter is the waiting time of a job from when it enters the system until it is executed. For a given task that entered the system at time t, let W_t be the number of steps until the task is executed.

By Theorem 3.4.1, part 2, we get that if an epoch starts when system is in a good state (i.e., with less than $c_1\gamma^{-1}n\ln n$ load), then with probability at least $1 - n^{-c}$ the load that was in the system before the epoch started is consumed during the epoch. Thus, if the system is in a good state at time t, and both the current and the next epoch are successful, then $W_t \le 2c_2\gamma^{-1}\ln n$. By summing the failure probabilities we prove that

$$\lim_{t\to\infty} \mathbf{Pr}(W_t \le 2c_2\gamma^{-1}\ln n) \ge 1 - 7n^{-c} \tag{3.15}$$

for large enough n, and the limit can be removed if the system starts in a good state.

Next we turn to computing $\mathbf{E}[W_t]$. The first problem we have to address is that there may be unsuccessful epochs during a good segment. The probability that t is in a good segment but the next epoch is unsuccessful is bounded by $2n^{-c}$. To simplify the computation, we assume that if an unsuccessful epoch occurs during a good segment, then the system switches to a bad state, and add $2n^{-c}$ to the probability that the system is in a bad state at time t. To bound the waiting time when a job arrives during a bad state, we note that the waiting time of a job is always bounded by the total load of the system at the time it arrives. If the system is in a bad state at time t, and the system switched to a bad state i epochs back, then the load of the system at time t is bounded by $c_1\gamma^{-1}n\ln n + ic_2\gamma^{-1}n\ln n$.

If we let $\mathcal{F}_t$ denote the event "the system is in a good state at time t, and both the current and the next epochs are successful," then by Lemma 3.4.6 and Relation (3.15), we have that

$$\begin{aligned}
\lim_{t\to\infty} \mathbf{E}[W_t] &\le \lim_{t\to\infty} \left(2c_2\gamma^{-1}\ln n \cdot \mathbf{Pr}(\mathcal{F}_t) + \mathbf{E}[L_t \mid \overline{\mathcal{F}_t}] \cdot \mathbf{Pr}(\overline{\mathcal{F}_t})\right) \\
&\le 2c_2\gamma^{-1}\ln n + \left(c_1\gamma^{-1}n\ln n + c_2\gamma^{-1}n\ln n \sum_{i=1}^{\infty} n^{-\rho i(c-1)}\right) \cdot 7n^{-c} \\
&= O(\gamma^{-1}\ln n).
\end{aligned}$$

Again the limit can be removed if the system starts in a good state. We have proven the following theorem:

Theorem 3.4.3. *Let W_t be the wait of a job that arrived at time t.*

1. $\lim_{t\to\infty} \mathbf{Pr}(W_t \le 2c_2\gamma^{-1}\ln n) \ge 1 - 7n^{-c}$.
2. $\lim_{t\to\infty} \mathbf{E}[W_t] = O(\gamma^{-1}\ln n)$.

The above bounds hold without the limits if the system starts in a good state.

3.4.2 General Stochastic Adversarial Model

In order to analyze the behavior of the system under the more power adversarial model that we presented in Section 3.1.2 it seems that we cannot resort to renewal theory; the adversary's behavior is much less predictable. We still, however, have the property that when the load is above some threshold Θ, then the expected consumed load after some time steps is smaller than the expected new load. Therefore, the load of the system behaves like a supermartingale as long as it is above Θ and it decreases in expectation. Just decreasing the expectation is not sufficient as we show later in this section. The machinery that we apply and formalizes the above thinking is Theorem 3.4.4, that follows from [64, Corollary 2]. Refer to [64] for the proof and an insightful discussion.

Stability

In this section we prove the stability of the system under a $(\lambda < 1, w, p > 2, M)$ stochastic adversary. The main technical tool that we use is the following theorem, which follows immediately from [64, Corollary 2].

Theorem 3.4.4. *Let $X_1, X_2, \ldots$ be a sequence of nonnegative random variables satisfying the following conditions:*

1. *There exist positive numbers $\alpha = \alpha_n$ and $\Theta = \Theta_n$ such that for all $x_1, \ldots, x_i$ with $x_i > \Theta$,*

$$\mathbf{E}[X_{i+1} - X_i \mid X_1 = x_1, \ldots, X_i = x_i] \le -\alpha.$$

2. *There exists a positive number $\xi = \xi_n$ and a $p = p_n > 2$ such that for all $x_1, \ldots, x_i$*

$$\mathbf{E}[|X_{i+1} - X_i|^p \mid X_1 = x_1, \ldots, X_i = x_i] \le \xi.$$

Then there exists $\Xi = \Xi(X_0, \alpha, \Theta, \xi)$ such that for all t,

$$\mathbf{E}[X_t \mid X_0] \leq \Xi + \max(0, X_0 - \Theta).$$

Furthermore, assuming that p is a constant with respect to n,

$$\Xi = O\left(\Theta + \alpha\left(1 + \frac{\xi}{\alpha^p}\right)^{3p}\right) \qquad \text{as } n \to \infty.$$

Our stability result is summarized in the following theorem.

Theorem 3.4.5. *Suppose that we run protocol $\mathcal{P}$ with a (λ, w, p, M) adversary, where $\lambda < 1$ and $p > 2$. Then*

$$\sup_{t \geq 0} \mathbf{E}[L_t \mid L_0] = O(\max(\gamma^{-1} n(w + \ln n)(1 + M)^{3p}, L_0)) \qquad \text{as } n \to \infty.$$

Proof. We first present the high-level idea of the proof. Recall that L_t is the load of the system at time t. As with the generator model, we partition the time into *epochs* of some length d and apply Theorem 3.4.4 to the subsequence $\{L_{di} \mid i \geq 0\}$. Using Theorem 3.4.1, part 1, we show that if the load at the beginning of an epoch is above some threshold Θ, then the expected load at the end of that epoch is strictly smaller by a significant amount, proving condition 1 of Theorem 3.4.4. We then derive the required bound on the pth moment $L_{(i+1)d} - L_{id}$, establishing condition 2 of Theorem 3.4.4. Applying Theorem 3.4.4 then gives the desired bound on the maximum expected load at the end of epochs. Finally, we generalize to steps that are not multiples of d.

Let us now formalize the above argument. The two conditions that we want to satisfy are

$$\mathbf{E}[L_{(i+1)d} - L_{id} \mid L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}, L_{id} = \ell_{id} > \Theta] \leq -\alpha,$$
$$\mathbf{E}[|L_{(i+1)d} - L_{id}|^p \mid L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}, L_{id} = \ell_{id}] \leq \xi$$

for all $(l_0, l_1, \ldots, l_{id})$. In order to simplify the notation we denote

$$\mathcal{L}_{i-1} = \{L_0 = l_0, L_d = l_d, \ldots, L_{(i-1)d} = l_{(i-1)d}\},$$

so the conditions become

$$\mathbf{E}[L_{(i+1)d} - L_{id} \mid \mathcal{L}_{i-1}, L_{id} = \ell_{id} > \Theta] \leq -\alpha, \tag{3.16}$$
$$\mathbf{E}[|L_{(i+1)d} - L_{id}|^p \mid \mathcal{L}_{i-1}, L_{id}] \leq \xi. \tag{3.17}$$

Let

$$\Theta = \sigma\gamma^{-1} n(w + \ln n) \tag{3.18}$$

for some constant σ that we will fix later. Also let

$$T_D = \gamma^{-1}(2\ln\Theta + \ln n) \qquad \text{and} \qquad T_C = \frac{\Theta}{n} + \frac{T_D}{2} + 1,$$

so that

$$\begin{aligned} T_E &= T_D + T_C \\ &= \frac{\Theta}{n} + 3\gamma^{-1}\ln\Theta + \frac{3}{2}\gamma^{-1}\ln n + 1 \\ &= \sigma\gamma^{-1}(w+\ln n) + 3\gamma^{-1}\ln\sigma + 3\gamma^{-1}\ln\gamma^{-1} + 3\gamma^{-1}\ln(w+\ln n) + \frac{9}{2}\gamma^{-1}\ln n + 1. \end{aligned} \tag{3.19}$$

We let σ be such that T_E/w is an integer, say k (we show later that such a σ exists), so that T_E is a multiple of the window size w, and define the epoch length

$$d = kw = T_E. \tag{3.20}$$

Notice that $d = O(\gamma^{-1}(w+\ln n))$, a fact that we use later. Fix some time $t = id$ and consider what happens in the case that $L_t > \Theta$. By Theorem 3.4.1, part 1 (for $c = 1$), we get that within $d = T_E$ steps the system consumes at least Θ units of load, with probability at least $1 - 1/n$. Thus, the expected number of tokens consumed between steps id and $(i+1)d$ is at least

$$\left(1 - \frac{1}{n}\right)\cdot\Theta = \sigma\gamma^{-1}n(w+\ln n) - \sigma\gamma^{-1}(w+\ln n), \tag{3.21}$$

independently of the past, and of any of the adversary's decisions.

Since an epoch consists of k windows, by the definition of the adversary, the expected injected new load in the system from time id until $(i+1)d$, conditioned on $\mathcal{L}_{i-1}$, is bounded by

$$k\lambda nw = \lambda nd.$$

Using equations (3.19) and (3.20) we can conclude that the expected new load injected by the adversary, conditioned on the history, is bounded by

$$\begin{aligned} \lambda nd &= \lambda\sigma\gamma^{-1}n(w+\ln n) + 3\lambda\gamma^{-1}n\ln\sigma + 3\lambda\gamma^{-1}n\ln\gamma^{-1} + 3\lambda\gamma^{-1}n\ln(w+\ln n) + \frac{9}{2}\lambda\gamma^{-1}n\ln n + \lambda n \\ &\le \lambda\sigma\gamma^{-1}n(w+\ln n) + 3\lambda\gamma^{-1}n\ln\sigma + 9\lambda\gamma^{-1}n\ln(c'n) + 3\lambda\gamma^{-1}n\ln(w+\ln n) + \frac{9}{2}\lambda\gamma^{-1}n\ln n + \lambda n \\ &= \gamma^{-1}n(w+\ln n)\left[\lambda\sigma + \frac{3\lambda\ln\sigma}{w+\ln n} + \frac{9\lambda\ln(c'n)}{w+\ln n} + \frac{3\lambda\ln(w+\ln n)}{w+\ln n} + \frac{9\lambda\ln n}{2(w+\ln n)} + \frac{\lambda}{\gamma^{-1}(w+\ln n)}\right], \end{aligned} \tag{3.22}$$

where the inequality follows from the fact that $\gamma^{-1} \le c'n^3$ for some constant c', since $\Lambda = \Omega(n^{-2})$ for any connected graph. Therefore, from Relations (3.21) and (3.22) we get

$$\begin{aligned} \mathbf{E}[L_{(i+1)d} - L_{id} \mid \mathcal{L}_{i-1}] &\le -\sigma\gamma^{-1}n(w+\ln n) + \gamma^{-1}n(w+\ln n)\left[\lambda\sigma + \frac{3\lambda\ln\sigma}{w+\ln n} + \frac{9\lambda\ln(c'n)}{w+\ln n}\right. \\ &\qquad \left. + \frac{3\lambda\ln(w+\ln n)}{w+\ln n} + \frac{9\lambda\ln n}{2(w+\ln n)} + \frac{\lambda}{\gamma^{-1}(w+\ln n)} + \frac{\sigma}{n}\right] \\ &\le -\sigma\gamma^{-1}n(w+\ln n) + \gamma^{-1}n(w+\ln n)(\lambda\sigma + Q), \end{aligned}$$

for sufficiently large n, where Q is a constant independent of σ. Hence,

$$\mathbf{E}[L_{(i+1)d} - L_{id} \mid \mathcal{L}_{i-1}] \le -\gamma^{-1}n(w+\ln n)(\sigma - \lambda\sigma - Q).$$

If

$$\sigma > \frac{Q}{1-\lambda},$$

then Relation (3.16) is satisfied for sufficiently large n. We therefore let σ be the smallest number that is greater than $Q/(1-\lambda)$ that ensures that $d = T_E$ is a multiple of the window size w. Notice that the fact that d is a continuous function of σ ensures that such a value of σ exists and satisfies

$$\sigma \leq \frac{2Q}{1-\lambda}.$$

Thus condition 1 is satisfied for $\alpha = O(\gamma^{-1}n(w + \ln n))$.

We now turn our attention to Relation (3.17). Let J_i and Z_i be the number of tokens injected by the adversary and consumed by the processors, respectively, during the ith epoch. We note that

$$\begin{aligned} |L_{(i+1)d} - L_{id}|^p &= |J_i - Z_i|^p \\ &\leq J_i^p + Z_i^p \\ &\leq J_i^p + d^p n^p, \end{aligned} \tag{3.23}$$

where the last inequality follows from the fact that the system (deterministically) consumes at most n tokens in every step.

We now bound $\mathbf{E}[J_i^p \mid \mathcal{L}_{i-1}, L_{id}]$. Write $J_i = \sum_{j=0}^{k-1} Y_j$, where

$$Y_j = \sum_{t=0}^{w-1} I_{id+jw+t}$$

is the number of tokens injected by the adversary during the window $[id + jw, id + (j+1)w - 1]$.

The second condition on the stochastic adversary gives, for all j,

$$\mathbf{E}[Y_j^p] \leq M n^p w^p.$$

Applying Hölder's inequality gives

$$J_i^p = \left(\sum_{j=0}^{k-1} Y_j\right)^p \leq \left(\sum_{j=0}^{k-1} 1\right)^{p\left(1-\frac{1}{p}\right)} \left(\sum_{j=0}^{k-1} Y_j^p\right) = k^{p-1} \sum_{j=0}^{k-1} Y_j^p$$

and therefore we get

$$\mathbf{E}[J_i^p \mid \mathcal{L}_{i-1}, L_{id}] \leq k^p M n^p w^p. \tag{3.24}$$

Therefore, relations (3.23) and (3.24) imply that Relation (3.17) is satisfied with

$$\xi = k^p M n^p w^p + d^p n^p = (M+1) d^p n^p,$$

and by Theorem 3.4.4 we deduce that

$$\sup_{i \geq 0} \mathbf{E}[L_{id} \mid L_0] = O(\max(\gamma^{-1} n(w + \ln n)(1+M)^{3p}, L_0)).$$

We have now proven the theorem for t corresponding to the beginning of an epoch. To finish the proof, we have for any $t \geq 0$

$$\begin{aligned}
\mathbf{E}[L_t \mid L_0] &= \mathbf{E}[L_{\lfloor t/d \rfloor d + (t - \lfloor t/d \rfloor d)} \mid L_0] \\
&\leq \mathbf{E}[L_{\lfloor t/d \rfloor d} \mid L_0] + \mathbf{E}\left[\sum_{j=\lfloor t/d \rfloor d}^{t} I_j \,\middle|\, L_0 \right] \\
&\leq \sup_{i \geq 0} \mathbf{E}[L_{id} \mid L_0] + \lambda n d \\
&= O(\max(\gamma^{-1} n (w + \ln n)(1+M)^{3p}, L_0)).
\end{aligned}$$

□

Theorem 3.4.6. *Given any initial load L_0, with probability 1 there is an $i \geq 0$, such that $L_{id} \leq \Theta$ for Θ defined as in* (3.18)*.*

Proof. For intuition, assume that the initial load L_0 is above Θ. Then by Relation (3.16), we expect it to decrease below Θ after a sufficiently long time period. In order to prove this fact, we use martingale techniques. Since the expected load decreases independently of the past, we can couple the load with a supermartingale until it drops below Θ. Then we can apply a martingale convergence theorem to show that the supermartingale (and therefore the coupled system load) will eventually reach Θ.

Proceeding formally, we define a supermartingale $\{Y_{id} \mid i \geq 0\}$ with respect to the sequence $\{L_{id} \mid i \geq 0\}$, where

$$Y_{id} = \begin{cases} \max(L_0, \Theta) & \text{for } i = 0, \\ L_{id} & \text{if } L_{(i-1)d} > \Theta, \\ \Theta & \text{if } L_{(i-1)d} \leq \Theta. \end{cases}$$

As long as $Y_{(i-1)d} > \Theta$ the two sequences of random variables are identical. The sequence L_{id} assumes a value $\leq \Theta$ if and only if there is an index j such that $Y_{jd} = \Theta$.

The (nonnegative) supermartingale Y_{id} converges with probability 1 to a random variable Y (see [43, Theorem 5.1]), and since

$$\mathbf{E}[Y_{(i+1)d} \mid Y_{id} > \Theta] \leq Y_{id} - \alpha$$

for some $\alpha > 0$ (defined in the proof of Theorem 3.4.5), we have $\lim_{i\to\infty} \mathbf{E}[Y_{id}] = \Theta$. By applying Fatou's lemma [28, page 110] we get $\mathbf{E}[Y] \leq \liminf_{i\to\infty} \mathbf{E}[Y_{id}] = \Theta$. Thus, with probability 1 the sequence $\{L_{id}\}$ assumes at some time jd a value less than or equal to Θ. □

Corollary 3.4.1. *1. If the system starts with no load, then for sufficiently large n,*

$$\sup_{t \geq 0} \mathbf{E}[L_t] = O(\gamma^{-1} n (w + \ln n)(1+M)^{3p}).$$

2. For any starting conditions

$$\limsup_{t\to\infty} \mathbf{E}[L_t \mid L_0] = O(\gamma^{-1} n (w + \ln n)(1+M)^{3p}) \qquad \textit{as } n \to \infty.$$

Proof. The first part follows from Theorem 3.4.5, while the second part also uses Theorem 3.4.6. □

An Instability Result

In [64] the authors give several counterexamples to justify the necessity of the second condition of Theorem 3.4.4. One of the goals in [64] is to establish a theorem that holds under general conditions, thus the counterexamples presented there consider fairly strong adversaries. For example, in all of those, for any state there is a positive probability for the next state to equal 0 (thus allowing the process to have large increments and yet keep the expectation bounded). Our setting is more restricted. For example, the decrement is bounded since in one time step the load can decrease by at most n. Nevertheless, here we show that the first condition of Theorem 3.4.4 is not sufficient to ensure stability.

Assume a network with just $n = 1$ node. Consider an adversary that at time step t inserts t new jobs with probability λ/t, where $\lambda < 1$. Then the expected number of new jobs at time t is λ. The second moment of the new load at time t is λt, so notice that is is bounded for every t but it is not uniformly bounded.

Let $\mathcal{E}_t$ be the event that "there were t new jobs inserted at time t." Then $\mathbf{Pr}(\mathcal{E}_t) = \lambda/t$. Fix $k > 0$. Since in one step the system can consume only one job, we have for any $t \geq 2k$:

$$\begin{aligned}
\mathbf{Pr}(L_t \geq t-2k) &\geq \mathbf{Pr}(\cup_{i=t-k+1}^{t} \mathcal{E}_i) \\
&\geq \sum_{i=t-k+1}^{t} \mathbf{Pr}(\mathcal{E}_i) - \sum_{t-k+1 \leq i < j \leq t} \mathbf{Pr}(\mathcal{E}_i \cap \mathcal{E}_j) \\
&\geq \frac{k\lambda}{t} - \binom{k}{2}\left(\frac{\lambda}{t-k+1}\right)^2 \\
&= \lambda k \left(\frac{1}{t} - \frac{\lambda(k-1)}{2(t-k+1)^2}\right).
\end{aligned}$$

Therefore, for every $t \geq 2k$ we get

$$\mathbf{E}[L_t] \geq (t-2k) \cdot \mathbf{Pr}(L_t \geq t-2k) \geq (t-2k) \cdot \lambda k \left(\frac{1}{t} - \frac{\lambda(k-1)}{2(t-k+1)^2}\right),$$

which implies that $\sup_t \mathbf{E}[L_t] \geq \lambda k$.

Since this holds for arbitrary k, we have $\sup_t \mathbf{E}[L_t] = \infty$.

Waiting Time

Having established that the system is stable, the next important performance parameter is the waiting time of a job from the time it enters the system until it is executed. For a given task that enters the system at time t, let W_t be the number of steps until the task is executed. Following the discussion of Section 3.2, throughout this section we assume that we perform protocol $\mathcal{P}^*$.

Theorem 3.4.7. *Suppose that we run protocol $\mathcal{P}^*$ with a (λ, w, p, M) adversary, where $\lambda < 1$ and $p > 2$. Then*

$$\sup_{t \geq 0} \mathbf{E}[W_t \mid L_0] = O(\max(\gamma^{-1}(w + \ln n)(1+M)^{3p}, L_0/n)) \qquad \textit{as } n \to \infty.$$

Proof. We begin with some intuition. By the results of Section 3.4.2, we expect the load L_t at time t to be low, namely, bounded by $O(\max(\gamma^{-1}n(w+\ln n)(1+M)^{3p}, L_0))$. We also expect that the distribution protocol will rapidly distribute this load among the nodes (even if it is not already distributed, and regardless of any load that comes in after time t)—this is formalized by Lemma 3.4.2. Once this load is evenly distributed, it can be quickly consumed, since $\mathcal{P}^*$ ensures that every node consumes the oldest token in its queue at the end of every time step. Therefore, the expected time to consume all the load is $O(\mathbf{E}[L_t]/n)$.

We now proceed formally. Assume that at some time t the load in the system is L_t. We apply Lemma 3.4.2 with $c = 1$, $\Theta = L_t$, $T_D = \gamma^{-1}(2\ln L_t + \ln n)$ and

$$T_C = \frac{\Theta}{n} + \frac{T_D}{2} + 1,$$

and we get that the expected time, conditioned on any past event $\mathcal{H}$, until all the tokens that were present at time t are consumed (measured from time t) is at most

$$2T_D + T_C \leq 4T_C + T_C = 5T_C$$

for $n \geq 2$. Thus for $n \geq 2$,

$$\mathbf{E}[W_t \mid L_t, \mathcal{H}] \leq 5T_C = \frac{5L_t}{n} + \frac{5\gamma^{-1}}{2}(2\ln L_t + \ln n) + 5.$$

Consequently, for sufficiently large n, we have that for any $t \geq 0$,

$$\begin{aligned}
\mathbf{E}[W_t \mid L_0] &= \sum_{\ell_t} \mathbf{E}[W_t \mid L_0, L_t = \ell_t] \cdot \mathbf{Pr}(L_t = \ell_t \mid L_0) \\
&\leq \sum_{\ell_t} \left(\frac{5\ell_t}{n} + \frac{5\gamma^{-1}}{2}(2\ln \ell_t + \ln n) + 5 \right) \cdot \mathbf{Pr}(L_t = \ell_t \mid L_0) \\
&= \mathbf{E}\left[\frac{5L_t}{n} + \frac{5\gamma^{-1}}{2}(2\ln L_t + \ln n) + 5 \,\middle|\, L_0 \right] \\
&= \frac{5\,\mathbf{E}[L_t \mid L_0]}{n} + \frac{5\gamma^{-1}}{2}(2\,\mathbf{E}[\ln L_t \mid L_0] + \ln n) + 5 \\
&\leq \frac{5\,\mathbf{E}[L_t \mid L_0]}{n} + \frac{5\gamma^{-1}}{2}(2\,\ln \mathbf{E}[L_t \mid L_0] + \ln n) + 5 \\
&= O(\max(\gamma^{-1}(w+\ln n)(1+M)^{3p}, L_0/n)),
\end{aligned}$$

where the second-to-last step follows from Jensen's inequality applied to the concave function $f(x) = \ln x$, and the last step follows from Theorem 3.4.5. □

Applying Theorem 3.4.6 we have the following

Corollary 3.4.2. *1. If the system starts with no load, then for sufficiently large n,*

$$\sup_{t \geq 0} \mathbf{E}[W_t] = O(\gamma^{-1}(w+\ln n)(1+M)^{3p}).$$

2. For any starting conditions,

$$\limsup_{t \geq 0} \mathbf{E}[W_t \mid L_0] = O(\gamma^{-1}(w+\ln n)(1+M)^{3p}) \qquad \text{as } n \to \infty.$$

Proof. The first part follows from Theorem 3.4.7, while the second part also uses Theorem 3.4.6. □

Strongly Bounded Adversaries

Recall that a strongly bounded adversary satisfies the additional requirement that for some constants $\alpha > 0$, $\beta \geq 1$, for any $\epsilon > 0$, the probability that the total number of new jobs that arrive in a given interval of length w is greater than $(1+\epsilon)\lambda n w$ is bounded by $e^{-\alpha\lambda n w \epsilon^{\beta}}$.

In this subsection we strengthen the preceding results for adversaries that are strongly bounded. The Chernoff-type restrictions on the input stream allow us to get high-probability results for the load and the waiting time.

High-Probability Bound on Load

Theorem 3.4.8. *Consider a system where load is injected by a strongly bounded adversary. Let L_t be the load of the system at time t. Then for any sufficiently large constant c_1 there exists a constant $c' > 0$ such that*

$$\limsup_{t\to\infty} \mathbf{Pr}(L_t > c_1\gamma^{-1}n(w + \ln n)) \leq 3n^{-c'}.$$

The above holds without the limit if the system starts with no load.

Proof. Let $\Theta = c_1\gamma^{-1}n(w+\ln n)$. We observe the system at some time t, and we need to bound the probability that the load at time t is above Θ. Therefore, we assume that the load at time t is above Θ and calculate the probability of the events that may have led to such a load. If the load at some time $t' < t$ were smaller than Θ (which holds with probability 1 as $t \to \infty$ by Theorem 3.4.6), then from time t' up to t some rare events have taken place and increased the load much more than expected. We bound the probability of those events, thus bounding the probability that the load at time t is above Θ.

Similarly to Theorem 3.4.5, we split time into epochs of length

$$T_E = c_2\gamma^{-1}(w + \ln n)$$

starting from time t and going backwards. The constant c_2 (which depends on c_1) is chosen so that the epoch length is a multiple k of the window size ($T_E = kw$). Let $\mathcal{B}$ be the event $\{L_t \geq \Theta\}$, and for $i \leq t/T_E$ let $\mathcal{B}_i$ be the event that the load of the system is above Θ for exactly the last i epochs. More precisely,

$$\mathcal{B}_i = \{\forall j = 0,\ldots,i-1 :\ L_{t-jT_E} \geq \Theta,\quad L_{t-iT_E} < \Theta\}.$$

Let $\mathcal{C}_t$ be the event that the load in the system was not always above Θ before time t. Formally,

$$\mathcal{C}_t = \{\exists t' \leq t :\ L_{t'} \leq \Theta\}.$$

Then we have

$$\mathbf{Pr}(\mathcal{B} \mid \mathcal{C}_t) = \sum_{i=1}^{\lfloor t/T_E \rfloor} \mathbf{Pr}(\mathcal{B}_i \mid \mathcal{C}_t). \tag{3.25}$$

To estimate $\mathbf{Pr}(\mathcal{B}_i \mid \mathcal{C}_t)$ we distinguish between two cases, depending on the total load injected by the adversary during the i epochs immediately before t. Either the adversary inserted a lot of new jobs during this time, or he inserted a reasonable number of new jobs and the protocol failed to reduce the total load. Both cases are intuitively unlikely: the first by the strong bound on the adversary, and the second by the efficacy of the protocol. We bound the two cases separately and then use a union bound. We fix a constant $\epsilon > 0$ (whose actual value will be determined later) and define $\mathcal{M}$ as the event that "the injected load during the i epochs immediately preceding t is less than $K = (1+\epsilon)i\lambda n T_E = (1+\epsilon)i\lambda nkw$." Then we have

$$\mathbf{Pr}(\mathcal{B}_i \mid \mathcal{C}_t) \leq \mathbf{Pr}(\overline{\mathcal{M}} \mid \mathcal{C}_t) + \mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \mid \mathcal{C}_t). \tag{3.26}$$

We bound each term separately, starting with $\mathbf{Pr}(\overline{\mathcal{M}} \mid \mathcal{C}_t)$. The first K jobs can be distributed in the ik windows of the period in

$$\binom{K+ik-1}{ik} \tag{3.27}$$

ways.

We now bound the probability of each such distribution of the inserted jobs $(K_1, K_2, \dots, K_{ik})$ (so that $\sum_{j=1}^{ik} K_j = K$). Recall that the expected number of jobs during the jth time window is at most λnw. Define ϵ_j as the deviation of K_j above λnw, namely,

$$\epsilon_j = \max\left(0, \frac{K}{\lambda nw} - 1\right).$$

In other words, ϵ_j is such that

$$K_j = (1+\epsilon_j)\lambda nw$$

if $K > \lambda nw$ and $\epsilon_j = 0$ otherwise. Since $\sum K_j = K$, we get that

$$\sum_{j=1}^{ik} \epsilon_j \geq ik\epsilon.$$

By using the definition of the strongly bounded adversary, we can bound the probability (conditioned on any past event) that in the jth window at least K_j were generated by

$$e^{-\alpha\lambda nw\epsilon_j^\beta}.$$

Therefore, the probability of a particular distribution $(K_1, \dots, K_{id})$ of the first K jobs is bounded by

$$\prod e^{-\alpha\lambda nw\epsilon_j^\beta} = e^{-\alpha\lambda nw\sum\epsilon_j^\beta}.$$

Since $\beta \geq 1$, and $\sum \epsilon_j \geq ik\epsilon$, we obtain $\sum \epsilon_j^\beta \geq ik\epsilon^\beta$ (by raising to the βth power and using Hölder's inequality as in Relation (3.24)), and the aforementioned probability becomes

$$e^{-\alpha\lambda nwik\epsilon^\beta}.$$

Together with Equation (3.27) we get that

$$\begin{aligned}
\mathbf{Pr}(\overline{\mathcal{M}} \mid \mathcal{C}_t) &\leq \binom{K+ik-1}{ik} e^{-\alpha\lambda nwik\epsilon^\beta} \\
&< \binom{K+ik}{ik} e^{-\alpha\lambda nwik\epsilon^\beta} \\
&\leq \left(\frac{(K+ik)e}{ik}\right)^{ik} e^{-\alpha\lambda nwik\epsilon^\beta} \\
&= e^{ik\ln(K+ik)+ik-\alpha\lambda nwik\epsilon^\beta - ik\ln(ik)} \\
&\leq e^{ik\ln[ik((1+\epsilon)\lambda nw+1)]+ik-\alpha\lambda nwik\epsilon^\beta - ik\ln(ik)} \\
&= e^{ik\ln[(1+\epsilon)\lambda nw+1]+ik-\alpha\lambda nwik\epsilon^\beta} \\
&< n^{-\kappa i}
\end{aligned} \tag{3.28}$$

for any constant κ and sufficiently large n.

Next we bound $\mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \mid \mathcal{C}_t)$. By Theorem 3.4.1, part 1, if at the beginning of an epoch the load of the system is at least Θ, then the load decreases by at least Θ with probability at least $1 - n^{-c}$. This is the case for the last $i-1$ epochs. However, at time $t - iT_E$ the load of the system is below Θ, while at time t the load is above Θ. Moreover we have assumed that the total new load is at most $K = (1+\epsilon)i\lambda nT_E$. These facts imply that the consumed load is less than K, which in turn implies that in fewer than

$$\frac{K}{\Theta} = \frac{(1+\epsilon)\lambda c_2}{c_1} i \doteq \mu i$$

epochs the consumed load was more than Θ. By making c_1 sufficiently large and ϵ sufficiently small, we can guarantee that $\mu < 1$. In this case, the probability that, among the $i-1$ epochs, fewer than μi consumed at least Θ load can be bounded by

$$\begin{aligned}
\mathbf{Pr}(\mathcal{B}_i \cap \mathcal{M} \mid \mathcal{C}_t) &\leq \binom{i-1}{(1-\mu)i} \left(\frac{1}{n^c}\right)^{(1-\mu)i} \\
&\leq \left(\frac{e(i-1)}{(1-\mu)i}\right)^{(1-\mu)i} \left(\frac{1}{n^c}\right)^{(1-\mu)i} \\
&= \left(n^{-c} \cdot \frac{e}{1-\mu} \cdot \frac{i-1}{i}\right)^{(1-\mu)i} \\
&\leq n^{-(1-\mu)(c-1)i}.
\end{aligned} \tag{3.29}$$

By combining equations (3.26), (3.28), and (3.29) we get that

$$\mathbf{Pr}(\mathcal{B}_i \mid \mathcal{C}_t) \leq 2n^{-(1-\mu)(c-1)i}.$$

Finally, we estimate the probability that the load is above Θ at time t using Equation (3.25). If we make c and c_1 sufficiently large, we get that $(1-\mu)(c-1) > 0$, so the sum converges and we get

$$\mathbf{Pr}(\mathcal{B} \mid \mathcal{C}_t) = \sum_{i=1}^{\lfloor t/T_E \rfloor} 2n^{-(1-\mu)(c-1)i} \leq \sum_{i=1}^{\infty} 2n^{-(1-\mu)(c-1)i} \leq 3n^{-(1-\mu)(c-1)}.$$

From Theorem 3.4.6 we have $\lim_{t\to\infty} \mathbf{Pr}(\mathcal{C}_t) = 1$, which gives the result. $\square$

Waiting Time By Theorem 3.4.1, part 2, we get that if the load of the system is bounded by Θ at some particular time, then with probability at least $1 - n^{-c} \geq 1 - n^{-c'}$ all the load that was in the system at that time is consumed during the next T_E steps. The limiting probability that the load of the system is above Θ is bounded by $3n^{-c'}$. Summing the failure probabilities proves the following.

Theorem 3.4.9. *Consider a system whose load is injected by a strongly bounded adversary. Let W_t be the wait of a job that arrived at time t.*

$$\liminf_{t\to\infty} \mathbf{Pr}(W_t \leq c_2\gamma^{-1}(w + \ln n)) \geq 1 - 4n^{-c'}.$$

The above holds without the limit if the system starts with no load.

3.5 Conclusion

We analyze a simple load balancing system and show that it has many desirable steady state properties under a big variety of input conditions. In particular, we derive low-degree polynomial bounds on the asymptotic expected load and waiting times of jobs in the system, and we match these expected performance results with high-probability results in a very natural restriction of the general stochastic adversary model. While there are many stability results for similar systems in the literature, our analysis of waiting time, the strength of our bounds, and our high-probability results are novel. In addition, our application of the Pemantle–Rosenthal result reveals many of the challenges in using general adversaries and will likely provide good insight for other applications. Finally, unlike much of the related work, our results hold for arbitrary connected network topologies and are optimal for the extremely important expander topology.

Chapter 4

Sampling Search-Engine Results

In this chapter we study a problem concerning search engines, namely the problem of efficiently sampling a random subset out of the entire set of results satisfying a given search query. We present the motivation for the problem and two algorithms for addressing it. We analyze them theoretically and we present some experimental results.

Most of the results of this chapter have appeared in [4], and are joint work with Andrei Broder and David Carmel.

4.1 Problem Description and Motivation

Web search continues its explosive growth: according to the Pew Internet & American Life Project [27], there are over 107 million web search users in United States alone, and they did over 3.9 billion queries in the month of June 2004. At the same time, the Web corpus grows: a study during the beginning of 2005 argues that the size of the indexable web is at least 11.5 billion pages [39].

Thus search algorithmic efficiency is as important as ever: although processor speeds are increasing and hardware is getting less expensive every day, the size of the corpus and the number of searches is growing at an even faster pace.

On the other hand, Web search users tend to make very short queries (less than 3 words long [72]), which result in very large result sets. Although by now search engines have become very accurate with respect to navigational queries (see [19] for definitions), for informational queries the situation is murkier: quite often the responses do not meet the user's needs, especially for ambiguous queries.

As an example, consider a user that is interested in finding out about famous opera sopranos and enters the query "`sopranos`" in the Google search box. It turns out that the most popular responses refer to the HBO's TV-series with the same name: in the top 100 Google results, only 7 documents *do not refer* to the HBO program. (All Google numbers, here and below, refer to experiments conducted in early 2005.)

This situation has stimulated search engines to offer various "post-search" tools to help users deal with large sets of somewhat imprecise results. Such tools include query suggestions or refinements (e.g., `yahoo.com` and `ask.com`), result clustering and the naming of clusters (e.g., `wisenut.com` and `vivisimo.com`), and mapping of results against a predetermined taxonomy, such as ODP (the Open Directory Project used by Google and many others), Yahoo, and LookSmart. All these tools are based in full or in part on the analysis of the result set.

For instance in the previous example, the search engine may present the categories "TV series," "Opera," etc., or the query extensions "HBO sopranos," "mezzo sopranos," etc. Ideally, in order to extract the most frequent categories within the results set, all the documents matching the query should be examined; for Web size corpora this is of course prohibitive, as thousands or millions of documents may match. Therefore, a common technique is to restrict attention only to the top few hundreds ranked documents and extract the categories from those. This is much faster since search engines use a combination of static (query-independent) rank factors (such as PageRank [18]) and query dependent factors. By sorting the index in decreasing order of static rank and using a branch-and-bound approach, the top 200 (say) results can be produced much faster than the entire set of results.

The problem with this approach is that the highly-ranked documents are not necessarily representative for the entire set of documents, as they may be biased towards popular categories. In the "sopranos" example, although 93 of the top 100 documents in Google refer to the HBO series, the query for "`sopranos AND HBO`" matches about 265,000 pages in Google (per Google report), while the query "`sopranos AND opera -HBO`" matches about 320,000, a completely different picture.

Many corporate search engines, and especially e-commerce sites, implement a technique called *multi-faceted* or *multidimensional search.* This approach allows the refinement of full-text queries according to meta-data specifications associated to the matching items (e.g., price range, weight) in any order, but only nonempty refinements are possible. The refinement is presented as a "browsing" of those results that satisfy certain metadata conditions, very similar to narrowing results in a particular category.

As an example, consider a user who visits an online music store such as `towerrecords.com`, and performs a query, say, the string "`james`." The engine (from `mercado.com`) provides a number of hits, but also numerous possible refinements, according to various "facets," for instance by "Genre" (Blues, Children's, Country, . . .), by price (Under \$7, Under \$10, Under \$15, . . .), by "Format" (Cassette, CD, Maxi-Single, Compact Disc, . . .), and so on. The refinements offered depend on the initial query, so that only nonempty categories are offered, and sparse subcategories are merged into an "Other" subcategory. Similar approaches are used by many other e-tailers.

Multi-faceted search is used in other contexts as well, for instance, Yee et al. [79] show the benefits of this approach as applied within the "Flamenco" project at U. C. Berkeley for searching images using metadata refinement.

Since the categories displayed for multi-faceted search depend on the result set of the query, they have to be extracted quickly, a procedure that becomes a problem when the corpus is large. It seems that some current multi-faceted search engines are limited to corpora that can be represented in memory.

4.1.1 Sampling the Search Results

The applications described above require significant processing time; in order to apply them to large corpora we propose to only *sample* the set of documents that match the user's query. Asymptotically, the average running time of one of our sampling approaches is only proportional to the sample size. On the other hand, sampling allows us to extract information that is unbiased with respect to the search-engine's ranking, and therefore produce better coverage of all topics or all meta-data values present in the full result set.

The main technical difficulty in sampling follows from the fact that we do not have the results of the query explicitly available, but instead the results are generated one after the other, by a rather expensive process, potentially involving numerous disk accesses for each query term. The straightforward implementation is to pay the price, find and store pointers to all the documents matching the original query, and build a uniform sample from these results. However, as we already mentioned, our algorithm will obtain the sample after generating and examining only a small fraction of the result set and yet the sample produced is uniform, that is, every set of matching pages of size k (the desired sample size) has an equal probability to be selected as the output sample.

Although, to the best of our knowledge, the idea of sampling query results from search engines is new, sampling has been applied in different contexts as a means to give fast approximate answers to a particular problem. The areas of randomized and approximation algorithms provide numerous examples. In the area of data streams, where the input size is very large, sampling the input and operating on it is a common technique (see e.g., [13, 35, 61]). Even databases allow the user to specify a sampling rate in a *select* operation that instead of performing the query on the full set of data operates on a sample [40]; as a result the DB2 relational database has been augmented to support this option.

4.1.2 Further applications

Besides the two applications already mentioned, result categorization and multi-faceted search, a random sample of the query results has more potential uses. In Theorem 4.2.3 we show that after the execution of our algorithm we can obtain an unbiased estimator of the total number of documents matching the user's original query, while in Theorem 4.2.4 we show that the estimator can achieve any prespecified degree of accuracy and confidence. Many users seem to like such estimates, maybe to help them decide whether they should try to refine the query further. In any case, Web search engines generally provide estimates of the number of results matching a query. For instance, both Google and Yahoo provide such estimates at the top of the search results page. However these estimates are notoriously unreliable, especially for disjunctions (see the discussion in [14]). As an example, as of early 2005, Google reports about 105M results containing the term "`George`," about 185M pages containing the term "`Washington`," while its estimate for the documents satisfying the query "`George OR Washington`" is about 33M. We get similar inconsistencies with other search engines, such as MSN Search. In contrast, in our experiments (see Section 4.5) even a 50-result uniform sample yielded estimates within 15% of target in all cases.

Yet another application of random sampling is to identify terms or other properties associated to the query terms. For instance one might ask “Who is the person most often mentioned on the Web together with Osama bin Laden?” The approach we envisage is to sample the results of the query “`Osama bin Laden`,” fetch the sample pages, run an entity detection text analyzer that can recognize people names, extract these names, and so on. Again the advantage of this approach compared to using the top results for the query “`Osama bin Laden`” is that the top results might be biased towards a particular context.

A similar application is suggested by Amitay et al. [3] where the authors demonstrate how finding (by “hand”) new terms relevant or irrelevant to a given query can be useful for building “corpus independent” performance measures for information retrieval systems. The main idea is that by providing a set of relevant and a set of irrelevant terms for a given query, we can evaluate the performance of the information retrieval system by checking whether the documents retrieved contained the specified relevant and irrelevant terms. However, discovering these sets of terms is a daunting task, which requires the time and skill of an IR specialist; a sample of the search results for the query can help the specialist identify both relevant and irrelevant terms. Again the lack of bias is probably useful.

Yet another application is suggested by Radev et al. [66], who propose the use of the Web as a knowledge source for domain-independent question answering by paraphrasing natural language questions in a way that is most likely to produce a list of hits containing the answer(s) to the question. It might well be the case that the results would be better when using a random sample of matches rather than a ranked set of matches, since the ranking is based on a very different idea of “best” results.

Finally, it might be possible to use sampling on the results of search-engine queries in order to extract summary information from the ensemble of the results and then we can use this information as a means of providing feedback to the user in order to refine his query.

The list of potential applications of search results sampling that we proposed above is probably far from complete. We hope that our work will stimulate search engines to implement a random sampling feature, and this in turn will lead to many more uses than we can conceive now.

4.1.3 Alternative Implementations

A very simple way of producing (pseudo) random samples is to keep the index in a random order. Then the first k matches of a query can be viewed as a random sample, or, if more than one sample is needed, we can take matches x to $x + k$ as our sample. In fact this is the approach used in IBM’s WebFountain [38], a system for large scale Web data mining.

However, in a standard Web search engine, there are many disadvantages for such an architecture:

1. If the index is in random order, rather than in decreasing static rank order, ranking regular searches (“top-k”) is very expensive since no branch-and-bound optimization can be used. Thus the random-order index has to be stored separately from the search index, and this

doubles the storage cost. (This is not an issue in WebFountain where "top-k" searches are a small fraction of the load.)

2. Maintaining a true random order as documents are added and deleted is nontrivial. A good solution is to have a "random static score" associated to each document and keep the index sorted by this "random score." This allows having an old index and a delta index to deal with additions.
3. Creating multiple truly independent random samples for the same query is nontrivial.

Thus, for regular Web search engines, sampling is a much better alternative.

4.1.4 Retrieval Model and Notations

Our model is a traditional *Document-at-a-time* (DAAT) model for IR systems [74]. Every document in the database is assigned a unique document identifier (DID). The DIDs are assigned in such a way that increasing DIDs corresponds to decreasing static scores; however this is not relevant to the rest of our discussion. Every possible term is associated with a *posting list*. This list contains an entry for each document in the collection that contains the index term. The entry consists of the document's DID, as well as any other information required by the system's scoring model such as number of occurrences of the term in the document, offsets of occurrences, etc. Posting lists are ordered in increasing order of the document identifiers.

Posting lists are stored on secondary storage media, and we assume that we can access them through stream-reader operations. In particular, each pointer to the posting list of some term A supports the following standard operations.

1. $A.loc()$: returns the current location of the pointer.
2. $A.next()$: advances the pointer to the next entry in the term's posting list and returns this entry's DID.
3. $A.next(r)$: moves the pointer to the first document with DID greater than or equal to r, and returns this DID.

For our purposes, we need a special operator

4. $A.jump(r, s)$: moves the pointer to the sth entry in the posting list after the document with DID greater than or equal to r, and returns this DID. (Equivalent to $A.next(r)$ followed by s $A.next()$ operations. However, simulating $A.jump(r, s)$ this way would cost s moves rather than one—see below.)

Operations $loc()$ and $next()$ are easily implemented with a linked-list data structure, while for $next(r)$ search engines augment the linked lists with tree-like data structures in order to perform the operation efficiently. For example, one can use a binary tree where the leaves are posting locations corresponding to the first posting in consecutive disk records and every inner node x contains the first location in the subtree rooted at x.

The *jump*(r, s) operation is not traditionally supported but can be easily implemented using the same tree data-structures needed for *next*(r)—we simply augment the inner nodes with a count of all the postings contained within the rooted subtree.

In the modern object-oriented approach to search engines based on posting lists and DAAT evaluation, posting lists are viewed as streams equipped with the *next* method above, and the *next* method for Boolean and other complex queries is built from the *next* method for primitive terms. For instance, $(A\,\mathbf{OR}\,B).\mathit{next}() = \min(A.\mathit{next}(), B.\mathit{next}())$. We will show later how to construct a basic *sample-next*(p) method that samples term posting lists with probability p, and show how to construct *sample-next*(p) methods for Boolean operators (**AND**, **OR**, **WAND**) from primitive methods.

Since the posting lists are stored on secondary storage, each *next* or *jump* operation may result in one or more disk accesses. The additional search-engine data structures ensure that we have at most one disk access per operation. Our goal is to minimize the number of disk accesses, and hence we want to minimize the number of the stream-reader pointer move operations. In the rest of the paper, we assume that these moves (i.e., *next*, *jump*, and *sample-jump*) have unit cost, while any other calculation has a negligible cost. (This assumption is of course only a first approximation, but it is well correlated with observed wall clock times [20]. A more accurate model would have to distinguish at least between "within-a-block" moves and "block-to-block" moves.)

For easy reference, we list here the notations used in the remainder of the paper. The total number of documents is N, while the number of documents containing term T_i is N_i. For the query under consideration, we let t be the number of terms contained in the query, and $m \leq N$ be the number of documents that satisfy the query. The sample size that we require is of size k; we expect in general to have $k \ll m$, and we let $p_s = k/m$ to be the ideal sampling probability.

The most general sampling technique that we propose is applicable to many search engine architectures. We describe it in Section 4.2. Next, in Section 4.3, we specialize to a particular architecture based on the **WAND** operator, which was first introduced in [20], and this specialization allows us to achieve better performance. Subsequently, in Section 4.4, we present an alternative scheme to sample results; this method is more efficient theoretically, but probably less efficient in practice. We implemented some of our algorithms and performed various experiments, and we present the results in Section 4.5.

4.2 A General Scheme for Sampling

4.2.1 Two Motivating Examples

In order to build some intuition for the sampling problem, we present two examples: one where the query is a conjunction (**AND**) of two terms and another where the query is a disjunction (**OR**) of two terms. Later in the paper we will provide more details about the sampling mechanism, and generalize it to a broader class of queries.

For the **AND** example consider some term A that appears in $10M$ documents, a term B that appears in $100M$ documents, and assume that the number of documents containing both terms is $5M$. Assume, moreover, that we want a sample of 1000 results. Then sampling each document that satisfies the **AND** query with probability equal to $p_s = 1000/5M = 1/5000$ creates a random sample with the desired expected size.

We use the notation A (resp. B, C, etc.) to mean both the term A and the set of postings associated to A. The meaning should be clear from context.

An initial problem arises from the fact that although we may know how many documents contain the term A and how many contain the term B, we do not know a priori the number of documents that contain both terms, and thus we do not know the proper sampling probability. There are ways to circumvent this issue and we discuss them later in Section 4.2.3. For now, assume that we know the correct sampling probability, and the question is how to sample efficiently.

The naive approach would be to identify every document that contains both terms and, for each document independently, add it to the sample with probability p_s. This means checking at least all the postings for the rarest of the terms, so we need to examine at least the $10M$ postings on A's posting list.

Instead consider the following approach: Sample the posting list of A (the rarest term) with probability p_s and create a virtual term A_{p_s} whose posting list contains the sampled postings of A. Then the posting list for A_{p_s} contains roughly $10M/5000 = 2000$ documents. We return the documents satisfying the query $A_{p_s}\,\mathbf{AND}\,B$. It is easy to verify that the result is a uniform sample over all the documents containing $A\,\mathbf{AND}\,B$. Later we will show how, given p_s, we can create the posting list of A_{p_s} online in time proportional to $|A_{p_s}|$; hence, this method allows us to examine only 2000 postings, a clear gain over the $10M$ postings examined by the naive approach.

Now let us look at the **OR** example that turns out to be somewhat more complicated. Consider another term C that appears also in $10M$ documents and assume that there are $15M$ documents containing $A\,\mathbf{OR}\,C$. Again we want a sample of 1000 documents, so in this case $p_s = 1000/15M = 1/15000$. The naive approach is to check every document in $A\,\mathbf{OR}\,C$ and insert it into the sample with probability p_s, which means traversing the posting lists of both A and C, or $20M$ operations. However we can apply the same technique as before and create a term A_{p_s} in time proportional to $|A_{p_s}|$. However, a document may satisfy the query even if it does not contain A, so we create also a virtual term C_{p_s} in the same manner, and return documents in $A_{p_s}\,\mathbf{OR}\,C_{p_s}$. Thus the total number of postings examined is $|A_{p_s}| + |C_{p_s}| = 20M/15000 \simeq 1333$, so we have a factor of 15000 improvement. But now we need to be more careful: if a document contains only the term A then it is inserted in A_{p_s} with probability p_s, and, similarly, if it contains only the term C then it is inserted in C_{p_s} with probability p_s. But if a document contains both terms, the probability to be contained in either A_{p_s} or C_{p_s} is $2p_s - p_s^2$. Hence, every document containing both A and C and contained in $A_{p_s}\,\mathbf{OR}\,C_{p_s}$ must be rejected from the sample with probability $1 - p_s/(2p_s - p_s^2)$. This will ensure that every document in $A\,\mathbf{OR}\,C$ is included in the sample with probability exactly p_s.

4.2.2 Sampling Search Results for a General Query

We now generalize the examples of the previous section and show how to apply the same procedure for sampling query results to any search engine based on inverted indices and a *Document-at-a-time* retrieval strategy. This class includes Google [18], AltaVista [23], and IBM's Trevi [29].

Consider a query Q, which can be as simple as the prior examples, or a more complicated boolean expression (including **NOT** terms, but not exclusively **NOT** terms). It could even contain more advanced operators like phrases or proximity operators. Every such query contains a number of simple terms, say $T_1, T_2, \ldots, T_t$, to which the operators are applied, and each term is associated with a posting list. Although the exact details depend on the specific implementation, every search engine traverses those lists and evaluates Q over the documents in the lists and several heuristics and optimization techniques are applied to reduce the number of documents examined (so, for example, for an **AND** query the engine will ideally traverse only the most infrequent term). Recall that the total number of documents satisfying the query is m, and that we need a sample of size k, which means that every document satisfying the query should be sampled with probability $p_s = k/m$. Assume, moreover, for the moment that we know m, and therefore we know the sampling probability p_s—in Section 4.2.3 we show how to handle this.

The way to sample the results is simple in concept. In a nutshell, we use rejection sampling to sample uniformly from the union of the posting lists $T_1, T_2, \ldots, T_t$, conditional on the sample satisfying the query.

For every term T_i (but not for terms $\mathbf{NOT}\, T_i$) we create a *pruned posting list* of document entries that contains every document from the posting list of T_i with probability p_s, independently of anything else. The naive way to create the pruned list is to traverse the original posting list and insert every document into the pruned list with probability p_s. An efficient equivalent way is to skip over a random number X of documents, where X is distributed according to a geometric distribution with parameter p_s. We can create a geometrically distributed random variable with parameter p_s, in constant time, by using the formula

$$X = \left\lceil \frac{\ln(U)}{\ln(1 - p_s)} \right\rceil,$$

where U is a real random variable uniformly distributed in the interval $[0, 1]$ (see [25]).

The random skip is then performed by executing a *jump*$(r,\ X)$ operation, where r is the last document considered. (Recall from the discussion of Section 4.1.4 that the data structure used for postings allows for efficiently skipping documents in the posting lists and thus in our model the skip has unit cost.) We then insert the document into the pruned list and we skip another random number of documents, continuing until the posting list is completely traversed. Note that the pruned lists can be precomputed at the beginning of the query, or they can be created on the fly, as the documents are examined.

We now perform the query by considering only documents that contain at least one term in the pruned lists. This is equivalent to replacing the original query $Q(T_1, T_2, \ldots)$ with the query

$$Q(T_1, T_2, \ldots)\, \mathbf{AND} \big(T_{1,p_s}\, \mathbf{OR}\, T_{2,p_s}\, \mathbf{OR} \cdots\big).$$

By this construction, every document that appears in some posting list has probability at least p_s to be considered. There are, however, documents that originally appear in more than one posting list. Consider some document that appears in the posting lists of r terms that are also being pruned. Then this document has increased chances to appear in *some* pruned list, the probability being exactly $1 - (1 - p_s)^r$. Therefore, for every document that satisfies the query, we should also count the number r of posting lists subject to pruning, in which it originally appears. Then we insert the document into the sample with probability $p_s/(1 - (1 - p_s)^r)$, so that overall the probability that the document is accepted becomes exactly p_s.

There are several remarks to be made about this technique:

- First we want to stress its generality, which allows it to be incorporated in a large class of search engines.

- Second, the method is very clean and simple, since it does not require any additional nontrivial data structures; indeed, although the pruned lists can be precomputed (and, to improve response time, even stored on disk for common search terms and fixed pruning probabilities), the pruned lists can exist only at a conceptual level. When an iterator traverses a pruned list, in the actual implementation, it may traverse the original posting list and skip the necessary documents. Our implementation that we describe in detail in Section 4.3, demonstrates this approach. The only addition we require is the support of the *jump* operation described in Section 4.1.4, which is not significantly different from the *next* operation. Therefore from a programming point of view, the needed modifications are very transparent.

- Furthermore, the modern object-oriented approach to search engines is to view posting lists as streams that have a *next* method, and to build a *next* method for Boolean and other complex queries from the basic *next* method for primitive terms. Our geometric jumps method provides a method that samples term posting lists with probability p_s providing the primitive *sample-next*(p_s) method, and the approach described above provides a *sample-next*(p_s) method for arbitrary queries: we first advance to the minimum posting in all pruned posting lists via the primitive *sample-next*(p_s) method, we evaluate the query, and if we have a match, we perform the rejection method as described.

- Finally, we want to mention that the general mechanism can be appropriately modified and made more efficient for particular implementations. For example, in the **AND** example of the previous section, we saw that we need to create the pruned list of only one of the terms. In Section 4.3 we show how we apply the technique to the **WAND** operator used in IBM's Trevi [29] and JURU [24] search engines and gain similar benefits.

4.2.3 Estimating the Sampling Probability

During the previous discussion we assumed that we know the total number of documents m matching the query and hence that we can compute the sampling probability $p_s = k/m$. In reality we do not know m, and therefore we have to adjust the probability during the execution of the algorithm. The problem of sequential sampling (sample exactly k out of m elements that arrive sequentially) when m is unknown beforehand, has been considered in the past. Vitter [75] was first to propose efficient algorithms to address that problem, using a technique called *reservoir sampling*. The main idea is that when the ith item arrives we insert it into the sample (reservoir) with probability k/i (for $i \geq k$) replacing a random element already in the sample. This technique ensures that at every moment, the reservoir contains a random sample of the elements seen so far. Vitter and subsequent researchers proposed efficient algorithms to simulate this procedure, which instead of checking every element skip over a number of them (see, for example, [75, 47]).

It seems, however, that those techniques cannot be applied directly to our problem, because the list of matching documents represents the union or intersection of several lists. If we simply skip over a number of documents, we do not know how many skipped documents matched the query and, therefore, we cannot decide what the acceptance probability of the chosen document should be.

Instead we apply the following technique, related to the method used in [35] in the context of stream processing: We maintain a buffer of size $B > k$ (e.g., B can equal $2k$), and set the initial sampling probability equal to some upper bound for the correct sampling probability, p_0; trivially we can set $p_0 = 1$. In other words, we accept every document that satisfies the query with probability $p = p_0$. Whenever the buffer is full, that is, the number of documents accepted equals B (which indicates that p was probably too large) we set a new sampling probability $p' = \alpha \cdot p$, for some constant $k/B < \alpha < 1$. Then every already accepted document is retained in the sample with probability α and deleted from the sample with probability $1 - \alpha$, all random choices being independent. Thus the expected sample size becomes $B\alpha > k$ and a Chernoff bound shows that with high probability the actual size is close to $B\alpha$, if B is large enough. Subsequent documents that satisfy the query are inserted into the sample with probability $p = p'$ independently of all other documents and p is decreased again whenever the buffer becomes full.

Eventually, the algorithm goes over all the posting lists and it ends up with a final sampling probability equal to some value p^*, and with a final number of documents in the sample, K, where $K < B$ always, and $K \geq k$ with high probability. Assuming the latter holds, we can then easily sample without replacement from this set and extract a sample of exactly k documents.

To recapitulate, we present in Figure 4.1 a high-level description of the entire algorithm for sampling the results of a general query. Note that, for the sake of simplicity, the description does not give any details, nor does it present the most efficient implementation. For example, in Figure 4.1, in order to consider the next candidate document we consider only the pruned posting lists; in an actual implementation, we would consider both the pruned lists and the posting lists of the actual terms.

To estimate the running time of the algorithm, observe that the number of times that the sampling probability is decreased is bounded by

$$\frac{\ln(1/p^*)}{\ln(1/\alpha)} \approx \frac{\ln(m/k)}{\ln(1/\alpha)}.$$

```
1. Function getSample()
2.    /* First some initializations. */
3.    curDoc ← 0
4.    $p \leftarrow 1$
5.    /* We assume that initially the pointers of all
       the terms' posting lists point to DID 0 */
6.
7.    foreach (term $T_j$)
8.       create a term $T_{j,p}$ with the same posting
          list as $T_j$
9.
10.   repeat
11.      foreach ($j : T_{j,p}$.loc() = curDoc)
12.         $T_{j,p}$.nextPruned(curDoc)
13.      curDoc ← min{$T_{j,p}$.loc(), $j = 1, \ldots, t$}
14.      if (curDoc = lastID)
15.         return /* Finished with all the
             documents */
16.      foreach ($j : T_j$.loc() < curDoc)
17.         $T_j$.next(curDoc)
18.      $r \leftarrow |\{j : T_j$.loc() = curDoc$\}|$
19.      if (curDoc satisfies $Q$)
20.         with probability normalizedProbability($r$)
             addToSample(curDoc)
21.   end repeat

1. Function $T$.nextPruned($r$)
2.    $X \leftarrow$ Geometric($p$)
3.    $T$.jump($r$, $X$)

1. Function normalizedProbability($r$)
2.    return $p/(1-(1-p)^r)$

1. Function addToSample(DID)
2.    Add DID to the sample
3.    /* Let $B$ be the size of the buffer. */
4.    while (size of sample = $B$)
5.       /* we should take a smaller sample */
6.       $p' \leftarrow \alpha \cdot p$
7.       foreach ($i \in$ sample)
8.          keep $i$ with probability $\alpha = p'/p$
9.       $p \leftarrow p'$
```

Figure 4.1: The General Sampling Scheme. We assume that we want to sample a query Q where terms $T_1, T_2, \ldots, T_t$ appear nonnegated.

Every time the probability is decreased the expected number of samples removed from the buffer is $(1-\alpha)B$. Thus, assuming that $B = \Theta(k)$, the expected total number of samples considered can be bounded by approximately

$$(1-\alpha)B\frac{\ln(m/k)}{\ln(1/\alpha)} + B = O\big(k\ln(m/k)\big). \tag{4.1}$$

Using this fact, and under independence assumptions that are common in information retrieval for the containment of terms in documents, we can show that the expected running time of this sampling scheme is bounded by approximately

$$O\big(k\ln(m/k)\big),$$

for any fixed query, and $k, m \to \infty$. The analysis is similar to the one that we present later in Section 4.3.3 so we omit it.

It is tempting to assert that the algorithm chooses independently every document with probability p^*. Unfortunately this is not the case: for every independent sampling probability p^* there is some probability that the sample will be larger than B; however, our algorithm never produces a sample larger than B. What holds is that, conditional on its size, the sample is uniform. Furthermore, we can use the final size and the final sampling probability to compute m, the size of the set that we sampled from. This is captured by Theorems 4.2.1, 4.2.3, and 4.2.4.

Theorem 4.2.1. *Assume that at the end of the sampling algorithm the actual size of the sample is K. Then the produced sample set is uniform over all sets of K documents that satisfy the query.*

Proof. We use a coupling (simulation) argument. Assume that each of the m documents that satisfy the query has an associated real random variable X_i, chosen independently uniformly at random in the interval $(0, 1]$.

We build a new algorithm that proceeds exactly as before except that whenever the buffer is full, p is reduced to p' and we keep in the buffer only those documents i that have $X_i < p'$. Every new document j is inserted in the buffer if and only if it has $X_j < p'$.

Let $S_p = \{i \mid X_i < p\}$. Then p^* is the largest value in the set $\{p_0, \alpha p_0, \alpha^2 p_0, \ldots\}$ such that

$$|S_{p^*}| = |\{i \mid X_i < p^*\}| < B,$$

and the final sample is S_{p^*}. Clearly the set S_{p^*} is uniform over all sets of size $K = |S_{p^*}|$. On the other hand the original algorithm and the new algorithm are in an obvious 1-1 correspondence, and thus, conditional on its size, the final sample is uniform. □

Notice that the algorithm does not know initially the number of documents that satisfy the query, a value that is usually hard to estimate. As we mentioned, an additional feature of the algorithm is that we can estimate the number of documents matching the query. Theorem 4.2.3 summarizes the result.

The main tool that we use in the proof of Theorem 4.2.3 is the concept of a martingale, and here we present the definition and the main result that we are using.

Definition 4.2.1. *Consider a sequence $\{X_t, t = 0, 1, \ldots\}$ of random variables, and a family of sets of random variables $\{\mathcal{H}_t, t = 0, 1, \ldots\}$, where $\mathcal{H}_{t-1} \subset \mathcal{H}_t$. We say that* the sequence $\{X_t\}$ forms a martingale with respect to $\{\mathcal{H}_t\}$ *if for every $t \geq 0$ the following three properties hold:*

1. *X_t is a function of $\mathcal{H}_t$.*
2. *X_t is integrable, that is, $\mathbf{E}[|X_t|] < \infty$.*
3. *$\mathbf{E}[X_{t+1} \mid \mathcal{H}_t] = X_t$.*

Intuitively, $\mathcal{H}_t$ corresponds to the history up to time t.

Definition 4.2.2. *A random variable T taking values in $\{0, 1, 2, \ldots\} \cup \{\infty\}$ is called a* stopping time *with respect to $\{\mathcal{H}_t\}$, if for every $t \in \{0, 1, 2, \ldots\}$ the indicator function of the event $\{T = t\}$ can be written as a function of the random variables in $\mathcal{H}_t$.*

This means that T is a stopping time if it is decidable whether or not $T = t$ with a knowledge only of the past and present, $\mathcal{H}_t$, and with no further information about the future.

Having defined a martingale and a stopping time, we are now able to present and prove a (nonstandard) version of the *Optional Sampling Theorem* that we use in our proof.

Theorem 4.2.2 (Optional Sampling Theorem)**.** *Consider a martingale $\{X_t, t = 0, 1, \dots\}$ with respect to $\{\mathcal{H}_t\}$ and assume that T is a stopping time, such that $\mathbf{Pr}(T < \infty) = 1$. Then we have that $\mathbf{E}[X_T] = \mathbf{E}[X_0]$ if there is a constant A independent of t, such that for every $t = 0, 1, 2, \dots$ we have that $\mathbf{E}[|X_{t\wedge T}|^2] < A$, where $t \wedge T = \min\{t, T\}$.*

Proof. Since the process $\{X_t\}$ forms a martingale with respect to $\{\mathcal{H}_t\}$, also the stopped process $\{X_{t\wedge T}\}$ forms a martingale with respect to $\{\mathcal{H}_t\}$, so, in particular, $\mathbf{E}[X_{t\wedge T}] = \mathbf{E}[X_0]$ for $t = 0, 1, 2, \dots$.

The fact that $\mathbf{E}[|X_{t\wedge T}|^2] < A$ implies that the sequence $\{X_{t\wedge T}\}$ is uniformly integrable, and the fact that $\mathbf{Pr}(T < \infty) = 1$ implies that $X_{t\wedge T} \to X_T$ almost surely. Therefore (see, for example, [77, page 131]),

$$\lim_{t\to\infty} \mathbf{E}[X_{t\wedge T}] = \mathbf{E}[X_T],$$

which concludes the proof. □

We are now in position to state and prove that our algorithm provides an unbiased estimator for the number of matches.

Theorem 4.2.3. *Assume that at the end of the algorithm the size of the sample is K, and the final sampling probability is p^*. Then the ratio K/p^* is an unbiased estimator for the number of documents m matching the query, that is, $\mathbf{E}[K/p^*] = m$.*

Proof. View the algorithm as performing two types of steps: if the buffer is full then the algorithm reduces the sampling probability and resamples the buffer with probability α; if the buffer is not full, the algorithm considers the next candidate document and inserts it with probability p.

Assume that after t steps there are K_t documents in the sample, the sampling probability is p_t, and we have considered m_t candidate documents. Thus if the algorithm stops after f steps, $K_f = K$, $p_f = p^*$, and $m_f = m$. If $m_t = m$, we also define $m_{t+1} = m_t = m$, $K_{t+1} = K_t$, and $p_{t+1} = p_t$. Now we define a sequence of random variables $\{X_t, t = 0, 1, \dots\}$ as follows. We let $X_0 = 0$ and for $t \geq 1$ we have

$$X_t = \frac{K_t}{p_t} - m_t.$$

We now show that the sequence $\{X_t\}$ is a martingale with respect to $\{\mathcal{H}_t\}$, where $\mathcal{H}_t = (K_0, p_0, m_0, K_1, p_1, m_1, \dots, K_t, p_t, m_t)$. This will finally imply that $\mathbf{E}[K_f/p_f] - \mathbf{E}[m_f] = 0$, which is what we want to prove.

It's clear that X_t is a a function of $\mathcal{H}_t$, while for the integrability notice that

$$\mathbf{E}[|X_t|] = \mathbf{E}\left[\left|\frac{K_t}{p_t} - m_t\right|\right] \leq \frac{B}{\alpha^t} + m < \infty,$$

where we used the fact that $K_t \leq B$, and $m_t \leq m$.

It remains to show that $\mathbf{E}[X_{t+1} \mid \mathcal{H}_t] = X_t$, the basic martingale property. If $m_t = m$, then the property holds trivially; if $m_t < m$ we consider two cases: First, if $K_t < B$ then the sampling probability does not change ($p_{t+1} = p_t$) but we consider a new document, which is inserted with probability $p_{t+1} = p_t$. Therefore, if we let Z be the indicator of the event that at time $t+1$ a document is accepted, we get

$$\begin{aligned}\mathbf{E}[X_{t+1} \mid \mathcal{H}_t] &= \mathbf{E}\left[\frac{K_t + Z}{p_{t+1}} - m_{t+1} \,\middle|\, \mathcal{H}_t\right] \\ &= \mathbf{E}\left[\frac{K_t + Z}{p_t} - (m_t + 1) \,\middle|\, \mathcal{H}_t\right] \\ &= \frac{K_t + p_t}{p_t} - m_t - 1 = X_t.\end{aligned}$$

On the other hand, if $K_t = B$ then every document already in the sample is resampled with probability $p_{t+1}/p_t = \alpha$ but we are not considering any new document, that is, $m_{t+1} = m_t$. Therefore

$$\begin{aligned}\mathbf{E}[X_{t+1} \mid \mathcal{H}_t] &= \mathbf{E}\left[\frac{\text{Binomial}(K_t, p_{t+1}/p_t)}{p_{t+1}} - m_{t+1} \,\middle|\, \mathcal{H}_t\right] \\ &= \mathbf{E}\left[\frac{\text{Binomial}(K_t, \alpha)}{\alpha p_t} - m_t \,\middle|\, \mathcal{H}_t\right] \\ &= \frac{K_t \alpha}{\alpha p_t} - m_t = X_t.\end{aligned}$$

Hence, we conclude that the sequence $\{X_t\}$ forms a martingale with respect to $\{\mathcal{H}_t\}$. We define the stopping time $f = \min\{t : m_t = m\}$. We will apply Theorem 4.2.2 for the martingale $\{X_t\}$ and the stopping time f, which will allow us to conclude that $\mathbf{E}[X_f] = \mathbf{E}[X_0] = 0$, therefore $\mathbf{E}[K_f/p_f] = E[m_f] = m$.

It is not hard to show that $\mathbf{Pr}(f < \infty) = 1$, but in order to apply Theorem 4.2.2 we have also to show that the second moment $\mathbf{E}[|X_{t\wedge f}|^2]$ is uniformly bounded (over t) by some constant. To this end, it helps to define for $t = 0, 1, 2, \ldots$ the random variable $Y_t = \log_\alpha p_t$. Then Y_t counts how many times the algorithm resampled from the buffer up to time t. We have

$$\begin{aligned}\mathbf{E}[|X_{t\wedge f}|^2] &= \mathbf{E}\left[\left|\frac{K_{t\wedge f}}{p_{t\wedge f}} - m_{t\wedge f}\right|^2\right] \\ &\le \mathbf{E}\left[\frac{K_{t\wedge f}^2}{p_{t\wedge f}^2}\right] + \mathbf{E}[m_{t\wedge f}^2] + 2\,\mathbf{E}\left[\frac{K_{t\wedge f}}{p_{t\wedge f}} m_{t\wedge f}\right] \qquad (4.2) \\ &\le B^2\,\mathbf{E}\left[\frac{1}{p_{t\wedge f}^2}\right] + m^2 + 2Bm\mathbf{E}\left[\frac{1}{p_{t\wedge f}}\right],\end{aligned}$$

where we used the fact that $K_{t\wedge f} \le B$, and $m_{t\wedge f} \le m$. We now show that the term $\mathbf{E}\left[\frac{1}{p_{t\wedge f}^2}\right]$ is uniformly bounded. We have

$$\begin{aligned}
\mathbf{E}\left[\frac{1}{p_{t\wedge f}^2}\right] &\le \sum_{i=0}^{\infty}\mathbf{Pr}\left(\frac{1}{p_{t\wedge f}^2} \ge i\right)\\
&= \sum_{i=0}^{\infty}\mathbf{Pr}\left(\frac{1}{p_{t\wedge f}} \ge \sqrt{i}\right)\\
&= \sum_{i=0}^{(1/\alpha)^{2m}-1}\mathbf{Pr}\left(\frac{1}{p_{t\wedge f}} \ge \sqrt{i}\right) + \sum_{i=(1/\alpha)^{2m}}^{\infty}\mathbf{Pr}\left(\frac{1}{p_{t\wedge f}} \ge \sqrt{i}\right)\\
&\le \left(\frac{1}{\alpha}\right)^{2m} + \sum_{i=(1/\alpha)^{2m}}^{\infty}\mathbf{Pr}\left(\left(\frac{1}{\alpha}\right)^{Y_{t\wedge f}} \ge \sqrt{i}\right)\\
&= \left(\frac{1}{\alpha}\right)^{2m} + \sum_{i=(1/\alpha)^{2m}}^{\infty}\mathbf{Pr}\left(Y_{t\wedge f} \ge \log_{1/\alpha}\sqrt{i}\right)\\
&\le \left(\frac{1}{\alpha}\right)^{2m} + \sum_{i=(1/\alpha)^{2m}}^{\infty}\mathbf{Pr}\left(Y_{f} \ge \log_{1/\alpha}\sqrt{i}\right),
\end{aligned}$$

since Y_t is increasing with t.

We will now bound the probability of the event $\mathcal{E}_i = \{Y_f \ge \log_{1/\alpha}\sqrt{i}\}$. Recall that Y_t counts how many times the algorithm resampled from the buffer up to time t, so, since there are m documents in total, event $\mathcal{E}_i$ implies that in at least $\log_{1/\alpha}\sqrt{i} - m$ resample steps no document was evicted from the buffer. Since in every resampling step each document that is in the buffer stays in the buffer with probability α, and since the buffer contains B documents at every resampling step, the probability of event $\mathcal{E}_i$ is bounded by $\alpha^{B(\log_{1/\alpha}\sqrt{i}-m)}$. Therefore we get

$$\begin{aligned}
\mathbf{E}\left[\frac{1}{p_{t\wedge f}^2}\right] &\le \left(\frac{1}{\alpha}\right)^{2m} + \left(\frac{1}{\alpha}\right)^{Bm}\sum_{i=(1/\alpha)^{2m}}^{\infty}\alpha^{\log_{1/\alpha} i^{B/2}}\\
&= \left(\frac{1}{\alpha}\right)^{2m} + \left(\frac{1}{\alpha}\right)^{Bm}\sum_{i=(1/\alpha)^{2m}}^{\infty}\frac{1}{i^{B/2}},
\end{aligned}$$

which is bounded for $B > 2$.

This implies that also $\mathbf{E}\left[\frac{1}{p_{t\wedge f}}\right]$ is uniformly bounded, and by Equation (4.2) the expectation $\mathbf{E}[|X_{t\wedge f}|^2]$ is also uniformly bounded. So, we can apply Theorem 4.2.2 and get that $\mathbf{E}[X_f] = \mathbf{E}[X_0] = 0$, which finally implies that

$$\mathbf{E}\left[\frac{K}{p^*}\right] = \mathbf{E}\left[\frac{K_f}{p_f}\right] = \mathbf{E}[m_f] = m.$$

Hence, K/p^* is an unbiased estimator for the number of documents satisfying the query. □

Besides having the correct expectation, a good estimator should be close to the correct value with high probability.

Definition 4.2.3. *An* (ϵ,δ)-approximation scheme *for a quantity X is defined as a procedure that given any positive $\epsilon < 1$ and $\delta < 1$ computes an estimate $\hat{X}$ of X that is within relative error of ϵ*

with probability at least $1-\delta$*, that is,*

$$\mathbf{Pr}(|\hat{X}-X| \le \epsilon X) \ge 1-\delta.$$

The following theorem shows that our sampling procedure, using a buffer size quadratic in $1/\epsilon$ and logarithmic in $1/\delta$, is in fact an (ϵ,δ)-approximation scheme.

Theorem 4.2.4. *There are constants* c_1, c_2 *such that for any positive* $\epsilon < 1$ *and* $\delta < 1$*, the algorithm above with a buffer size* $B = \frac{c_1}{\epsilon^2}\ln\frac{c_2}{\delta}$ *is an* (ϵ,δ)*-approximation scheme, that is, if at the end of the algorithm the size of the sample is* K *and the final sampling probability is* p^* *we have:*

$$\mathbf{Pr}\left(\left|\frac{K}{p^*}-m\right| \le \epsilon m\right) \ge 1-\delta.$$

Proof. The proof is similar to that of Theorem 3 in [35]. We assume that the initial sampling probability is $p_0 = 1$ and we have $p_i = \alpha p_{i-1}$, so that $p_i = \alpha^i$. In order to simplify some calculations we assume that $\alpha \le 3/4$. We set the buffer size to be

$$B = \frac{1+\epsilon}{\alpha}\cdot\frac{3}{\epsilon^2}\ln\frac{8}{\delta}.$$

We use the coupling argument (the one used in Theorem 4.2.1) and think of the algorithm as sampling from all documents and working by levels. Let Y_0 be the number of all the matching documents, Y_1 be a random variable counting the number of documents that got sampled with probability α, Y_2 be a random variable that counts the documents that further got sampled with probability α and so on. Then Y_i is distributed as Binomial(m, α^i).

Here is the main idea. The final outcome size K equals one of the Y_i's (the one for which we have $Y_{i-1} \ge B$ and $Y_i < B$) and then p^* equals α^i. The idea is that for small i, with good probability, the estimates Y_i/α^i are accurate and if K equals one of those then the algorithm provides a good estimate. Otherwise K equals one of the later Y_i's, but this has small probability.

We define the events

$$\mathcal{B}_i = \left\{\left|\frac{Y_i}{\alpha^i}-m\right| > \epsilon m\right\} = \{|Y_i - m\alpha^i| > \epsilon m\alpha^i\}.$$

We also define

$$\ell = \max\ i \quad \text{s.t.} \quad m\alpha^i \ge \frac{3}{\epsilon^2}\ln\frac{8}{\delta},$$

and therefore we have

$$m\alpha^{\ell+1} < \frac{3}{\epsilon^2}\ln\frac{8}{\delta}.$$

So, the probability that the estimator fails to be within a factor of ϵ close to m is bounded by

$$\sum_{i=1}^{\ell}\mathbf{Pr}(\mathcal{B}_i) + \mathbf{Pr}(Y_\ell \ge B).$$

By applying a Chernoff bound we have that for $i \le \ell$

$$\mathbf{Pr}(\mathcal{B}_i) \le 2e^{-\frac{\epsilon^2}{3}m\alpha^i}.$$

If $f(i) = e^{-\frac{\epsilon^2}{3} m\alpha^i}$, then for $i \leq \ell$ we have $f(i)/f(i-1) \geq (8/\delta)^{(1-\alpha)/\alpha}$, which for $\alpha \leq 3/4$ is at least 2, for any $\delta < 1$. So for the summation we have

$$\begin{aligned}\sum_{i=1}^{\ell} \mathbf{Pr}(\mathcal{B}_i) &\leq \sum_{i=1}^{\ell} 2e^{-\frac{\epsilon^2}{3} m\alpha^i} \\ &\leq 4e^{-\frac{\epsilon^2}{3} m\alpha^\ell} \\ &\leq 4e^{-\frac{\epsilon^2}{3}\frac{3}{\epsilon^2}\ln\frac{8}{\delta}} \\ &= \frac{\delta}{2}.\end{aligned}$$

For $\mathbf{Pr}(Y_\ell \geq B)$ we have

$$\begin{aligned}\mathbf{Pr}(Y_\ell \geq B) &= \mathbf{Pr}\left(Y_\ell \geq \frac{1+\epsilon}{\alpha}\frac{3}{\epsilon^2}\ln\frac{8}{\delta}\right) \\ &\leq \mathbf{Pr}\left(Y_\ell \geq \frac{1+\epsilon}{\alpha} m\alpha^{\ell+1}\right) \\ &= \mathbf{Pr}\left(Y_\ell \geq (1+\epsilon)\cdot m\alpha^\ell\right) \\ &\leq e^{-\frac{\epsilon^2}{3} m\alpha^\ell} \\ &\leq \frac{\delta}{8}.\end{aligned}$$

Therefore, the probability that the algorithm fails is less than δ. □

4.3 Efficient Sampling of the WAND Operator

Although we described a general sampling mechanism that can be applied to diverse settings, we have also seen that when we specialize to some particular operator such as **AND** we can achieve improved performance. In this section we describe the operator **WAND**, introduced in [20], which generalizes **AND** and **OR**, and we present an efficient implementation for sampling the results of **WAND**.

4.3.1 The WAND Operator

Here we briefly describe the **WAND** operator that was introduced in [20] as a means to optimize the speed of search queries. **WAND** stands for Weak **AND**, or Weighted **AND**. It takes as arguments a list of Boolean variables $X_1, X_2, \ldots, X_k$, a list of associated positive *weights*, $w_1, w_2, \ldots, w_k$, and a threshold θ. By definition, $\mathbf{WAND}(X_1, w_1, \ldots X_k, w_k, \theta)$ is true iff

$$\sum_{1 \leq i \leq k} x_i w_i \geq \theta, \tag{4.3}$$

where x_i is the indicator variable for X_i, that is,

$$x_i = \begin{cases} 1, & \text{if } X_i \text{ is true} \\ 0, & \text{otherwise.} \end{cases}$$

Observe that **WAND** can be used to implement **AND** and **OR** via

$$\mathbf{AND}(X_1, X_2, \dots X_k) \equiv \mathbf{WAND}(X_1, 1, X_2, 1, \dots X_k, 1, k),$$

and

$$\mathbf{OR}(X_1, X_2, \dots X_k) \equiv \mathbf{WAND}(X_1, 1, X_2, 1, \dots X_k, 1, 1).$$

For the purposes of this paper we shall assume that the goal is simply to sample the set of documents that satisfy Equation (4.3) with X_i indicating the presence of query term T_i in document d. We note however that the situation considered in [20] is more complicated: there each term T_i is associated with an upper bound on its maximal contribution to any document score, UB_i, and each document d is subject to a preliminary filtering given by

$$\mathbf{WAND}(X_1, UB_1, X_2, UB_2, \dots, X_k, UB_k, \theta),$$

where X_i again indicates the presence of query term T_i in document d. If **WAND** evaluates to true, then the document undergoes a full evaluation, hence a document that matches **WAND** does not necessarily match the query. We can deal with this approach by doing a full evaluation on every document that we would normally insert into the buffer (that is, a document that won the coin toss). The document is then inserted into the buffer only if it passes the full evaluation. This insures that p is reduced only as needed. Further refinements considered in [20], such as varying the threshold θ during the algorithm, are meant to increase the efficiency of finding the top k results and thus are beyond the scope of this paper.

4.3.2 Sampling WAND Results

In the **AND** example that we saw previously, we can sample only the rarest term, and hence minimize the total number of *next*, *jump*, and *sample-next* operations. In contrast, in the **OR** example, we must sample the posting lists of all terms. Since the **WAND** operator varies between **OR** and **AND**, a good sampling algorithm must handle efficiently both extremes.

Let $T = \{T_1, \dots, T_t\}$ be the set of the query terms with associated positive weights, $w_1, w_2, \dots, w_t$. Our goal is to sample uniformly at random documents from the set of documents $\{j\}$ that satisfy the inequality

$$\sum_{1 \le i \le t} x_{i,j} w_i \ge \theta, \tag{4.4}$$

where $x_{i,j}$ is given by

$$x_{i,j} = \begin{cases} 1, & \text{if document } j \text{ contains } T_i \\ 0, & \text{otherwise.} \end{cases}$$

We divide T it into two subsets, the set S that contains the terms that must be sampled, and the set S^c that contains the rest of the terms in T. In the **AND** example, the set S contains only the least frequent term, while in the **OR** example the set S contains all the terms.

The first issue is how to select the set S. We will discuss the optimal way to do it, after discussing the running time of the algorithm. For the time being, assume that we choose the set S arbitrarily such that

$$\sum_{i \in S^c} w_i < \theta.$$

Hence S is such that any document that satisfies Equation (4.4) must contain at least one term from S. It is easy to check that the **AND** and the **OR** examples expressed as **WAND** obey this inequality for their respective choices of S.

Following the description in Section 4.2.2, we create *pruned lists* for the terms in S (but not for the terms in S^c), and again as before, a document in the posting list of a term is included in the pruned list of that term with probability p_s, independently of other documents and other terms.

Of course the algorithm does not know p_s beforehand, so it initially starts accepting all the documents with some probability $p = p_0$, maybe $p = 1$, and it reduces p over time, using the process described in Section 4.2.3.

The algorithm guarantees that every document that contains at least one term in S has probability at least p to be selected. If it becomes selected and it satisfies **WAND**, we normalize the probability to be exactly p using the rejection method described in Section 4.2.2. If a document does not contain any term from S, its total weight is strictly smaller than θ and, therefore, it does not satisfy **WAND**.

We now give a high-level description of the sampling algorithm. The details appear in Figure 4.2, while a complete description and a formal proof of correctness can be found in Section 4.3.4; Figure 4.3 contains a visual example.

Every term in the set S is associated with a *producer*, which is an iterator traversing the pruned list, selecting documents for evaluation against the query. Furthermore, in order to perform the evaluation, every term in the query is also associated with a *checker* that traverses the original posting list. At one iteration of the algorithm we advance the producers that point to the document with the smallest DID, and some document is selected (with probability p) by some of them. Then the checkers will determine the terms that are contained in the document and if the sum of their weights exceeds the threshold θ, then the document becomes a candidate to be selected for the sample. Like in the general approach, the pruned list may exist only at the conceptual level, and the producers may traverse the original posting lists and jump over a random number of documents, which is geometrically distributed.

Once a document, whose DID is held in the variable *global*, is selected for consideration, we use the checkers to determine if *global* satisfies the query. Some checkers point to documents with DID smaller than *global* and these are terms that, as far as we know at this point, might be contained in the document with DID=*global*. The algorithm maintains an upper bound equal to the sum of the weights of the terms whose checkers point to a document with DID not greater than *global*. As long as the upper bound exceeds the threshold θ (and therefore *global* might satisfy the query), we advance some term's checker to the first document with DID $\geq$ *global*. Assume its DID is *doc*. If *doc* = *global* then the term is contained in *global*. We continue by advancing the rest of the

```
1.  **Function** getWANDSample()
2.    /* First some initializations. */
3.    curDoc ← 0
4.    global ← 0
5.    $p \leftarrow 1$
6.    **foreach** (term $i$)
7.      checker[i].next(0)
8.    **foreach** (term $i \in S$)
9.      producer[i].nextPruned(0)
10.
11.   **repeat**
12.     advance global to smallest DID for which
          $\sum_{i:\text{checker[i].DID} \le \text{global}} w_i \ge \theta$
13.     **if** (global < min DID of producers)
14.       global ← min DID of producers
15.     /* Now at least one producer is ≤ global. */
16.     $A \leftarrow$
          {terms $i \in S$ s.t. producer[i].DID < global}
17.     **while** ($A \neq \emptyset$ && no producer points to
          global)
18.       pick $i \in A$
19.       producer[i].nextPruned(global)
20.     **if** (no producer points to global)
21.       global ← min DID of producers
22.     **if** (global = lastID)
23.       **return** /* Finished with all the
            documents */
24.     /* Now the global points to a DID that
          exists in some pruned list, and such that
          the accumulated weight behind it is at
          least θ. */
25.     $B \leftarrow$
          {terms $i \in T$ s.t. checker[i].DID ≤ global}
26.     /* B contains the terms that contribute to
          the upper bound */
27.     **if** (global ≤ curDoc)
28.       /* document at global has already been
            considered */
29.       pick $i \in B$
30.       /* it is probably best to pick an
            $i \in B \cap S$ */
31.       checker[i].next(curDoc + 1)
32.     **else** /* global > curDoc */
33.       **if** ($\sum_{i \in B:\text{checker[i].DID} = \text{global}} w_i \ge \theta$)
34.         /* Success, we have enough mass on
              global. */
35.         curDoc ← global
36.         /* We consider curDoc as a candidate.
              Now we must count exactly how many
              posting lists in S contain global in
              order to perform the probability
              normalization correctly. */
37.         **foreach**
              ($i \in S \cap B$ s.t. checker[i].DID < curDoc)
38.           checker[i].next(curDoc)
39.         $D \leftarrow$
              {terms $i \in S$ s.t. checker[i].DID = global}
40.         with probability
              normalizedProbability(|D|)
              addToSample(curDoc)
41.       **else** (of line 33)
42.         /* Not enough mass yet on global,
              advance one of the preceding terms. */
43.         pick $i \in B$ s.t. checker[i].DID < global
44.         /* it is probably best to pick an
              $i \in B \cap S$ */
45.         checker[i].next(global)
46.   **end repeat**

1.  **Function** producer[i].nextPruned($r$)
2.    $X \leftarrow$ Geometric($p$)
3.    producer[i].jump($r$, $X$)

1.  **Function** normalizedProbability($r$)
2.    **return** $p/(1-(1-p)^r)$

1.  **Function** addToSample(DID)
2.    Add DID to the sample
3.    /* Let B be the size of the buffer. */
4.    **while** (size of sample = $B$)
5.      /* we should take a smaller sample */
6.      $p' \leftarrow \alpha \cdot p$
7.      **foreach** ($i \in$ sample)
8.        keep $i$ with probability $\alpha = p'/p$
9.      $p \leftarrow p'$
```

Figure 4.2: Sampling **WAND**.

checkers that are behind *global* until either the total sum of weights of the terms whose checkers are in positions ≤ *global* is less than the threshold θ, in which case the document does not satisfy the query, or until the sum of the weights of the terms that were found to be contained in *global* exceeds the threshold θ, in which case the document becomes a candidate to be selected for the

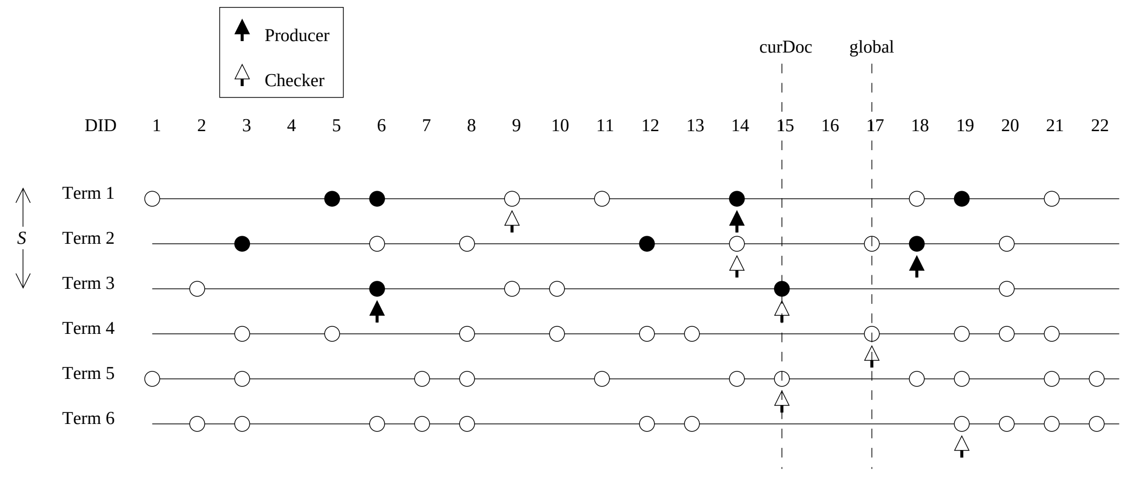


Figure 4.3: An example of the posting lists. A bullet indicates that the term exists in the corresponding document. A black bullet indicates that the document was sampled (or will be), hence it exists in the pruned list.

sample. In the latter case, the next step is to count the exact number of terms in S that are contained in the document. Each of these terms offers a chance to the document to be inserted to the corresponding pruned list, therefore, by counting the terms in S that are contained in the document we can apply the rejection method, described in Section 4.2.2, and accept the document with the correct probability (i.e., with probability p).

Notice that the algorithmic description leaves some details unspecified. For instance, whenever some checker has to be advanced there is usually more than one choice. The goal is to select the checker that will advance the farthest possible, and a simple heuristic is to select the checker of the most infrequent term. This problem appears in the general context of query constraints satisfaction for posting list iterators and there are more advanced heuristics that try to guess the best move based on the results seen so far (see [23]). In our particular case, at some point during the execution of the algorithm, there is even more flexibility: we can either advance a checker or a producer (e.g., at line 31 we can advance a producer instead of a checker). Hence in principle, we can select whether it is better to advance a producer or a checker, based on our experience so far and the expected benefit of the choice and, indeed, our implementation uses this heuristic.

4.3.3 Running-Time Estimation and the Choice of the Set S

We now bound the running time of the algorithm, assuming that we know the correct value of the sampling probability $p_s = k/m$. Consider a query with t terms, and recall that N_i is the total number of documents containing the ith term and that w_i is the weight of the ith term in the **WAND** operator. In order to obtain an upper bound for the number of pointer advances, we note that whenever we advance a checker we advance it to at least past a producer, since during the execution of the algorithm the document under consideration (*global*) has been originally selected by some producer. Therefore, the total number of each checker's advances is bounded by the total number of producer advances, which is expected to be $p_s \sum_{i\in S} N_i$. Therefore, the running time is expected to be

$$O\Big(tp_s \sum_{i\in S} N_i\Big) = O\Big(t\frac{k}{m}\sum_{i\in S} N_i\Big). \tag{4.5}$$

If the sampling probability is not known in advance, then in the worst case sampling will not help much. For instance if the standard search **WAND** spends a large amount of time getting the first B matches and then starts producing matches very fast, the sampling **WAND** will spend an equal amount of time until the first decrease of p from 1 to α. This is of course unlikely but entirely possible.

Hence for the average case we need to assume that the results are uniformly distributed with respect to DID numbers. To this end we assume the often-used probability model in IR, that is, we assume that each document contains the query terms independently with certain probabilities. In this case, conditional on a document d containing a term $t_i \in S$, there is a fixed probability π_i that d satisfies the query. Similarly there are fixed probabilities, $\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,s}$ that d satisfies the query and contains exactly $1, 2, \ldots, s$ terms from S, where $s = |S|$. Now consider the first time a document d is selected by a producer, say for the term t_i. Assume that at that time the sampling probability was p. In view of the above, the probability that d satisfies the query and also passes the rejection procedure is

$$\sum_{j=1}^{s} \frac{\pi_{i,j}p}{1-(1-p)^j} \geq \sum_{j=1}^{s} \frac{\pi_{i,j}}{j} \doteq \rho_i.$$

On the other hand, in view of Equation (4.1), we know that the expected total number of samples ever inserted in the buffer is bounded by approximately $O\big(k\ln(m/k)\big)$. Hence the expected number of occurrences of the term t_i selected by its producer is bounded by approximately

$$O\Big(\frac{k}{\rho_i}\ln(m/k)\Big),$$

and therefore the expected total number of moves (producers and checkers) is approximately

$$O\Big(tk\ln(m/k)\sum_{i\in S}\frac{1}{\rho_i}\Big) = O\big(k\ln(m/k)\big), \tag{4.6}$$

for any fixed query, and $k, m \to \infty$.

In order to minimize the running time of the algorithm, we want to select S so that the sum $\sum_{i\in S}\rho_i^{-1}$ is minimized. Of course ρ_i is not known in advance, but it can be estimated as the query progresses. Another approach, for $m \ll N_i$, is to make the rough estimate $\rho_i \approx m/N_i$. Then Equation (4.6) again suggests that a good choice for S is to try to minimize $\sum_{i\in S} N_i$.

A simple way to achieve a good selection for S in this vein is to sort the terms in increasing order of frequencies (and decreasing order of weights in case of ties), and let

$$s = \min_i \text{s.t.}: \quad \sum_{j=i+1}^{t} w_j < \theta.$$

Then let $S = \{1, 2, \ldots, s\}$. Notice that this greedy approach includes both the examples of **AND** and **OR** as special cases.

The optimal choice for the set S to minimize $\sum_{i \in S} N_i$ is obtained by solving the following integer program:

$$\begin{aligned} \min \quad & \sum_{i \in S} N_i \\ \text{s.t. :} \quad & \sum_{i \in S^c} w_i < \theta, \end{aligned}$$

or, equivalently,

$$\begin{aligned} \max \quad & \sum_{i \in S^c} N_i \\ \text{s.t. :} \quad & \sum_{i \in S^c} w_i < \theta, \end{aligned}$$

which can be interpreted as a Knapsack problem. Since the values N_i are integral we can solve it exactly in polynomial (in t and N) time through dynamic programming, but since we have a small number of terms we can solve it much more efficiently by brute force. Sometimes we have some flexibility in assigning weights (usually we want terms with low frequency to have large weight), in which case the greedy approach will suffice to obtain an optimal solution.

The analysis above is based on the independence assumptions for the containment of terms in documents; in reality, however, the running time will depend on the actual joint distribution of the query terms, which generally changes as the algorithm iterates through the posting lists. In practice we can achieve better performance by observing the performance of each producer and dynamically changing the set S as the algorithm progresses. We want to insert terms that both produce large jumps and are well correlated with successful samples so that the sampling probability will go down quickly.

4.3.4 Detailed Description of the Algorithm and Proof of Correctness

Here we present a detailed description as well as a formal proof of the correctness of the algorithm presented in Figure 4.2. The essence of the proof is to show that a set of four invariants is maintained throughout the execution of the protocol. For $i \in S$, let

$$\mathcal{C}_i = \{\text{DIDs that appear in the pruned list of term } i\},$$

and $\mathcal{C} = \bigcup_{i \in S} \mathcal{C}_i$. The four invariants are the following:

1. All documents with DID $\leq$ *curDoc* have either been considered as candidates, or do not belong to $\mathcal{C}$.

2. For any term $i \in T$, any document containing i with DID $<$ $\mathit{checker}[i].\mathit{DID}$ has either been considered as a candidate, or does not belong to $\mathcal{C}$.

3. At every given time point, every document with DID $<$ global has either been considered or does not belong to $\mathcal{C}$.

4. For all terms $i \in S$, every document containing i with DID $<$ $\mathit{producer}[i].\mathit{DID}$ has either been considered or does not belong to $\mathcal{C}_i$.

It is easy to verify that all for of them are true after the initializations (line 9).

We next try to find the next candidate document. We increase *global* to equal the minimum DID that could be a potential candidate, which happens when the sum of the weights of terms whose *checker*s are behind (i.e., point to documents with DID smaller than or equal to) *global* reaches the threshold θ. Since we increase *global* we must verify that invariant 3 is maintained after the execution of line 12. This is indeed true by invariant 2. $[2 \Rightarrow 3]$

The next candidate document must be pointed to by some *producer*. Therefore, at lines 13–14, if all the *producer*s are ahead of *global* we increase *global* to the smallest *producer*. The validity of invariant 3 follows from invariant 4. $[4 \Rightarrow 3]$

The next candidate document is the one with the smallest DID $\geq$ *global* that exists in some pruned list. In order to discover it, we advance all the *producer*s that are behind *global* in their pruned list to their first document with DID $\geq$ *global* (lines 16–19). Notice that we can stop if we discover a pruned list that contains *global*. $[3 \Rightarrow 4]$

By line 20, either some *producer* points to *global*, or we have advanced all the *producer*s past *global*. In the latter case (lines 20–21) we increase *global* to the smallest of the *producer*s. $[4 \Rightarrow 3]$

At lines 22–23 we check whether we have traversed all the documents and in that case the sampling is finished. Otherwise *global* points to the next candidate document. Notice that if the DID of that document is $\leq$ *curDoc*, then the document has been considered in the past (invariant 1) so we can advance one of the *checker*s past *curDoc* (lines 27–31). (Actually *global* cannot be strictly smaller than *curDoc*.) $[1 \Rightarrow 2]$

Otherwise we have a candidate new document. The next step (line 33) is to check whether we have discovered enough terms contained in the document, that is, we check whether the sum of the weights of terms whose *checker*s point to *global* exceeds the threshold θ. If this is the case, then we consider the new document as a candidate and we set *curDoc* to its DID $[3 \Rightarrow 1]$. Now the document should be inserted to the sample after applying the rejection scheme described in Section 4.2.3. Since we want the probability of the document being sampled to be exactly p, we must compute the probability that at least some pruned list contains the document. Consider all the terms S that have *producer*s (and associated pruned lists). Assume that the document appears in the posting lists of r such terms. Then for each of those, the probability to appear to the corresponding pruned list is p, therefore the probability to appear in *some* pruned list equals $1-(1-p)^r$. Hence we must normalize and accept the document with probability $p/(1-(1-p)^r)$ (lines 39–40). Previously, at lines 37–38, we advance the *checker*s of all terms in S in order to compute r. $[1 \Rightarrow 2]$

In the case that the sum of the weights of terms whose *checker*s point to *global* is less than θ (i.e., in the case that the **if** clause of line 33 evaluates to false) we chose one of the terms whose *checker* is behind *global* (and therefore contributed to the upper bound at line 12) and we advance the corresponding *checker* to the first smallest DID $\geq$ *global*. $[1 \Rightarrow 2]$

When the execution of the algorithm is over, invariant 1 proves that every document in $\mathcal{C}$ gets at some point to be considered as a candidate. At line 39, the set D contains exactly those terms that exist in set S and are contained in document *global*. This follows from the definition of the set D and from invariant 2, combined with the **foreach** loop at line 37. (If a posting list contains the document *global* then its *checker* cannot be ahead of *global*, otherwise invariant 2 would have been violated. It is therefore behind, and the **foreach** loop makes sure that the *checker* will point exactly to *global*.) Therefore we count all the terms in S that exist in document *global*, hence (because of the normalization) the document becomes accepted with probability exactly p.

4.4 An Alternative Sampling Scheme

In this section we present a different way for adjusting the sampling probability. As we will see, this technique is faster theoretically, but in practice it may not be as efficient, as it may require traversing a term's posting list more than once.

Here is the main idea. For concreteness, we present the algorithm for sampling a **WAND** query, although the same method can be applied for a general query (where the set S will include all the query terms that appear in the query and are not negated, as in Section 4.2.2). Recall that N_i is the number of documents in the posting list of the ith term. We let $N = \sum_{i \in S} N_i$. By Equation (4.5), whenever we sample with probability p, the expected running time is bounded by $O(tpN)$. In this scheme we sample initially with some small probability $p_0 = 1/N$. Then the expected time needed to scan all the lists is $O(t)$. If we sample at least k documents, then we stop and we have a uniform sample (like previously, if we end with more than k documents then we further select a sample of exactly k documents, by sampling from the final set without replacement). Otherwise, we set $p_1 = 2p_0$, and repeat. In general, if in the ith step the buffer did not become full, then we increase the sampling probability $p_{i+1} = 2p_i$ (so $p_i = 2^i/N$), and we restart. Let $\ell = \log_2 \frac{kN}{m}$. We are expected to finish when $p_i \simeq p_\ell = k/m$, so we expect the total time to be about

$$O\left(t\left(\sum_{i=0}^{\ell} p_i N\right)\right) = O\left(t\left(1 + 2 + 4 + \cdots + \frac{k}{m}N\right)\right) = O\left(t\frac{k}{m}N\right).$$

Let us try now to analyze the running time and the performance more rigorously. We prove the following lemma, which bounds the total number of producer advances.

Lemma 4.4.1. *The expected number of producer advances is bounded by* $6kN/m$.

Proof. We call the traversing of the posting list with probability p_i *round* i. Notice that for the ith round, the total number of producer advances is stochastically dominated by a binomial random variable, Binomial(N, p_i). Let the number of producer advances at round i be X_i (X_i equals 0 if the algorithm stopped before round i) and the total number of producer advances be $X = \sum_{i=0}^{\infty} X_i$.

We then have

$$\mathbf{E}[X] = \sum_{i=0}^{\ell+1} \mathbf{E}[X_i] + \sum_{i=\ell+2}^{\infty} \mathbf{E}[X_i] \le \sum_{i=0}^{\ell+1} \mathbf{E}[\text{Binomial}(N, p_i)] + \sum_{i=\ell+2}^{\infty} \mathbf{E}[X_i].$$

We denote by $\mathcal{A}_i$ the event that "the algorithm has not terminated up to (and including) round i." First notice that conditioning on $\overline{\mathcal{A}_{i-1}}$ we have $X_i = 0$. Also, event $\mathcal{A}_i$ implies that at round i there were fewer than k documents sampled. Since at level i the number of documents that become sampled is distributed as Binomial(m, p_i), the expected number of sampled documents is $m2^i/N$, and for $i \ge \ell + 1$ we get by a Chernoff bound

$$\begin{aligned}
\Pr(\mathcal{A}_i) &= \mathbf{Pr}(\text{Binomial}(m, 2^i/N) < k) \\
&= \mathbf{Pr}\left(\text{Binomial}(m, 2^i/N) < \frac{k}{m2^i/N} m2^i/N\right) \\
&\le e^{-\frac{1}{2}\frac{m2^i}{N}\left(1-\frac{k}{m2^i/N}\right)^2} \\
&\le e^{-\frac{1}{8}\frac{m2^i}{N}}.
\end{aligned}$$

From the previous calculation and from the fact that conditional on event $\mathcal{A}_{i-1}$ the random variable X_i is stochastically dominated by a Binomial(N, p_i), we get for $i \ge \ell + 2$

$$\begin{aligned}
\mathbf{E}[X_i] &= \mathbf{E}[X_i \mid \mathcal{A}_{i-1}] \cdot \mathbf{Pr}(\mathcal{A}_{i-1}) + \mathbf{E}[X_i \mid \overline{\mathcal{A}_{i-1}}] \cdot \mathbf{Pr}(\overline{\mathcal{A}_{i-1}}) \\
&\le N\frac{2^i}{N} e^{-\frac{1}{8}\frac{m2^{i-1}}{N}} + 0,
\end{aligned}$$

and so

$$\begin{aligned}
\mathbf{E}[X] &\le \sum_{i=0}^{\ell+1} 2^i + \sum_{i=\ell+2}^{\infty} 2^i e^{-\frac{1}{8}\frac{1}{N}m2^{i-1}} \\
&\le 2^{\ell+2} + 2^{\ell} \sum_{i=\ell+2}^{\infty} 2^{i-\ell} e^{-\frac{1}{8}\frac{1}{N}m2^{i-1}} \\
&= \frac{4kN}{m} + \frac{kN}{m} \sum_{j=2}^{\infty} 2^j e^{-\frac{1}{8}\frac{1}{N}m2^{\ell+j-1}} \\
&= \frac{4kN}{m} + \frac{kN}{m} \sum_{j=2}^{\infty} 2^j e^{-\frac{1}{16}k2^j} \\
&\le \frac{6kN}{m},
\end{aligned}$$

for $k \ge 6$. □

Since, as we argued right before Equation (4.5), the number of advances that each checker performs is bounded by the total number of producer advances, and by making use of Lemma 4.4.1, we have proven the following theorem.

Theorem 4.4.1. *The expected running time of the algorithm is $O(tkN/m)$.*

Let us compare now the running time of this scheme with the one of the first algorithm. According to Equation (4.6) the expected running time of the first algorithm under independence assumptions is approximately

$$O\Big(tk \ln(m/k) \sum_{i \in S} \frac{1}{\rho_i}\Big).$$

Recall from the discussion after Equation (4.6) that $\rho_i \approx m/N_i$, therefore the expected running time is approximately

$$O\left(tk \ln\left(\frac{m}{k}\right) \frac{N}{m}\right).$$

Therefore, we can see that the first scheme is slower by a logarithmic factor than the second scheme. More importantly, Theorem 4.4.1 holds without the independence assumptions that are introduced in the analysis of the first scheme. Nevertheless, as we mentioned previously, we expect the second scheme to be less efficient in practice, since it requires accessing the terms' posting lists several times.

Now we show that the second algorithm also provides an (ϵ, δ)-approximation scheme to the number of documents that match the query.

Theorem 4.4.2. *There are constants c_1, c_2 such that for any positive $\epsilon < 1$ and $\delta < 1$, the algorithm above with a requested sample size $k = \frac{c_1}{\epsilon^2} \ln \frac{c_2}{\delta}$ is an (ϵ, δ)-approximation scheme, that is, if at the end of the algorithm the size of the sample is K and the final sampling probability is p^* we have:*

$$\mathbf{Pr}\left(\left|\frac{K}{p^*} - m\right| \le \epsilon m\right) \ge 1 - \delta.$$

Proof. The main idea is to show that the algorithm will not terminate in the first rounds (up to round $\ell - 1$), while if it terminates later it provides a good approximation to the number of documents matching the query.

Let $\mathcal{B}_i$ be the event that "the algorithm terminated at round i," and $\mathcal{C}_i$ the event that "the algorithm terminated at round i and failed to provide an estimation within ϵ to m." Finally, let $\mathcal{B}$ be the event that "the algorithm failed." Then we have

$$\mathbf{Pr}(\mathcal{B}) \le \sum_{i=0}^{\ell-1} \mathbf{Pr}(\mathcal{B}_i) + \sum_{i=\ell}^{\infty} \mathbf{Pr}(\mathcal{C}_i).$$

In order to bound $\mathbf{Pr}(\mathcal{B}_i)$, we notice that the number of matches at the ith round is a random variable distributed as a Binomial$(m, 2^i/N)$. Therefore,

$$\begin{aligned}\mathbf{Pr}(\mathcal{B}_i) &\le \mathbf{Pr}(\text{Binomial}(m, 2^i/N) \ge k) \\ &\le \mathbf{Pr}\left(\text{Binomial}(m, 2^i/N) \ge \frac{k}{2^i m/N} \frac{2^i m}{N}\right) \\ &\le e^{-\frac{1}{3}\frac{2^i m}{N}\left(\frac{kN}{2^i m} - 1\right)^2}.\end{aligned}$$

Notice that if $f(i) = e^{-\frac{1}{3}\frac{2^i m}{N}\left(\frac{kN}{2^i m} - 1\right)^2}$, then for $i \le \ell - 1$ and $k \ge 2$ we have $f(i)/f(i-1) \ge e^{7k/12} > 2$. Therefore,

$$\sum_{i=0}^{\ell-1} \mathbf{Pr}(\mathcal{B}_i) < 2 \cdot \mathbf{Pr}(\mathcal{B}_{\ell-1}) \le 2e^{-\frac{k}{6}}.$$

Also,

$$\begin{aligned}\mathbf{Pr}(\mathcal{C}_i) &\le \mathbf{Pr}\left(\left|\frac{\text{Binomial}(m, 2^i/N)}{2^i/N} - m\right| > \epsilon m\right)\\ &= \mathbf{Pr}\left(\left|\text{Binomial}(m, 2^i/N) - \frac{2^i m}{N}\right| > \epsilon\frac{2^i m}{N}\right)\\ &\le 2e^{-\frac{1}{3}\frac{2^i m}{N}\epsilon^2}.\end{aligned}$$

So,

$$\begin{aligned}\sum_{i=\ell}^{\infty}\mathbf{Pr}(\mathcal{C}_i) &\le 2\cdot\sum_{j=0}^{\infty} e^{-\frac{1}{3}\frac{2^{\ell+j}m}{N}\epsilon^2}\\ &\le 2\cdot\sum_{j=0}^{\infty} e^{-\frac{1}{3}2^j k\epsilon^2}\\ &\le 3e^{-\frac{1}{3}k\epsilon^2}.\end{aligned}$$

Putting everything together, we get

$$\mathbf{Pr}(\mathcal{B}) \le 2e^{-\frac{k}{6}} + 3e^{-\frac{1}{3}k\epsilon^2},$$

which for

$$k = \frac{6}{\epsilon^2}\ln\frac{3}{\delta}$$

is less than δ. □

4.5 Experiments

We implemented the sampling mechanism for the **WAND** operator and performed a series of experiments to test the efficiency of the approach as well as the accuracy of the results. We used the JURU search engine developed by IBM [24].

The data consisted of a set of 1.8 million Web pages, consisting of a total of 1.1 billion words (18 million total distinct words). Each document was classified according to its content to several categories. The taxonomy of the categories, as well as the classification of the documents to categories, were performed by IBM's Eureka classifier described in [1]. We used a total of 3000 categories, and each document belonged to zero, one, or more categories. Eureka's taxonomy contains additionally a number of broader super-categories that form a hierarchical structure. Although we did not make use of this structure in our experimental evaluation, we argue later in this section that it can be used to provide more meaningful results for the category-suggestion problem.

In order to estimate the gain in run-time efficiency, we count the number of times a pointer is advanced (via *next*, *jump*, or *sample-next*) over the terms' posting lists. As we argued previously, the total running time depends heavily on the number of those advances, since the posting lists are usually stored on secondary storage and accessing them is the main bottleneck in the query response time.

#	Query
Q_1	Schumacher AND (Joel OR Michael)
Q_2	Olympic AND (Airline OR Games OR Gods)
Q_3	Turkey AND Customs
Q_4	Long AND Island AND Tea
Q_5	Schwarzenegger AND (California OR Terminator)
Q_6	Taxi AND Driver
Q_7	Dylan AND (Musician OR Poet)
Q_8	Football AND (Lazio OR Patriots)
Q_9	Indian AND (America OR Asia)

Table 4.1: The queries that we inserted to the sampling algorithm.

We experimented by creating nine ambiguous queries depicted in Table 4.1 chosen to produce results in many different categories. For each query we created different samples of sizes $k = 50$, 200, and 1000. In all the experiments the resampling probability equals $\alpha = 3/4$ and the buffer size is $B = 2k$. In Table 4.2 we compare the number of pointer advances for different sample sizes and in Figure 4.4 we plot a summary of those results. Notice that even though the total number of matching documents is small (in the order of several thousands, while the motivation for our techniques is for applying them to queries with result sizes in the millions) we show a significant gain for small sample sizes. In order to further establish this point we performed additional queries using artificially created documents built from random sequences of numbers, such that the result sets would be larger. We present the results in Table 4.3.

Query	Matches	No Sampling	50	200	1000
Q_1	587	3275	2627	4297	4561
Q_2	5109	31121	4323	12716	31231
Q_3	3111	33849	13841	24192	35461
Q_4	1111	28604	12120	28547	40151
Q_5	407	2497	1532	3278	3314
Q_6	1028	6491	3783	6401	7475
Q_7	356	3678	3173	4967	4967
Q_8	566	8796	5060	8699	9123
Q_9	15721	96997	6437	19423	55248

Table 4.2: Number of pointer advances for the nine queries. The second column contains the total number of pages matching each query. The rest of the columns contain the number of pointer advances performed without sampling, and for samples of 50, 200, and 1000 pages.

From the two tables it is clear that sampling is justified if the sampling size k is at least 2 orders of magnitude smaller than the actual result size m. In this case the total time can be reduced by a factor of 10, 100, or even more, depending on the ratio k/m, as well as on the query type. On the other hand, if k is comparable to m, the overhead of the sampling (due to more than one pointer for each term) might even increase the total time.

Query	Matches	No Sampling	10	100
T_1 AND T_2	13011	104087	977	7161
T_3 OR T_4	57046	120102	566	4392
T_3 OR T_4 OR T_5	62890	134874	715	5351

Table 4.3: Comparison of pointer advances for queries performed on artificially created documents with samples of sizes 10 and 100.

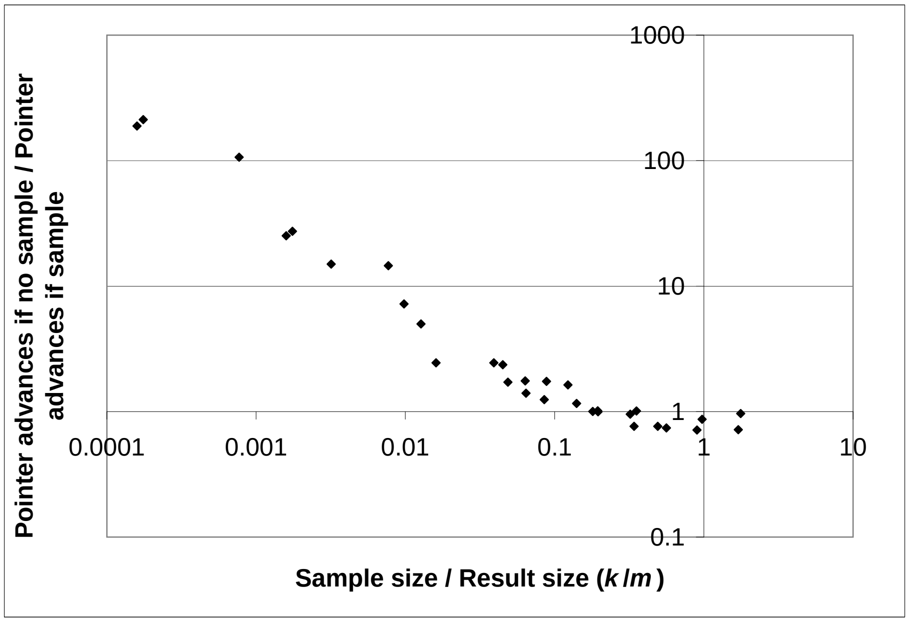


Figure 4.4: A log − log plot of the pointer advances as a function of the ratio k/m. Summary of the data of Tables 4.2 and 4.3.

4.5.1 Estimating the Most Frequent Categories of the Search Results

We also evaluated the suitability of our approach for a particular application, namely the discovery of the most frequent categories spanned by the set of documents matched by a given query. We emphasize that we are not testing the uniformity of the samples: our samples are provably uniform; what we test here is whether uniform samples capture popular categories, something which of course depends on the distribution of categories over the result set. To this end, we consider the same queries of Table 4.2. Each of these query results induces a set of categories from the Eureka taxonomy. In order to determine whether the sampling succeeds in discovering the most frequent categories, we measured, for each sample size, how many of the 10 most frequent categories in the full result set are present in the sample; we show the results in Table 4.3(a) and a summary in Figure 4.5(a).

Furthermore, it is desirable for the frequent categories in the full result set to be also frequent in the sample so that we can identify them. For that, for each query, we check how many of the top-10 frequent categories in the result set are present within the top-10 frequent categories according to the sample, and we show the results in Table 4.3(b) and in Figure 4.5(b).

There are some facts worth noticing with respect to the results of sampling, some of which are not revealed in the tables. First observe that in most cases, even small sample sizes succeed in sampling documents from the frequent categories (Table 4.3(a)) but a somehow larger sample size is needed in order to ensure that the frequent categories are frequent in the sample as well (Table 4.3(b)). It also seems that a sample of size 1000 is always successful in our examples, but this is somewhat misleading since in some of the examples the total number of documents is small, and therefore the sampling extracts all the original categories.

(a) Number of the top-10 frequent categories that appear in the samples.

Query	50	200	1000
Q_1	10	10	10
Q_2	7	10	10
Q_3	7	10	10
Q_4	4	8	10
Q_5	5	10	10
Q_6	7	9	10
Q_7	7	10	10
Q_8	8	10	10
Q_9	2	9	10

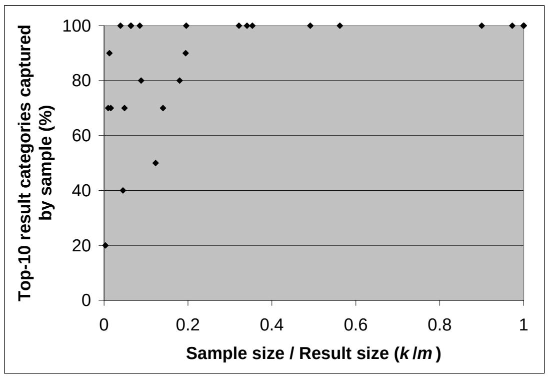
(b) Number of the top-10 frequent categories that appear in the 10 most frequent sample categories.

Query	50	200	1000
Q_1	7	8	10
Q_2	6	5	8
Q_3	4	6	9
Q_4	3	6	10
Q_5	3	7	10
Q_6	3	4	10
Q_7	5	10	10
Q_8	7	8	10
Q_9	0	3	7

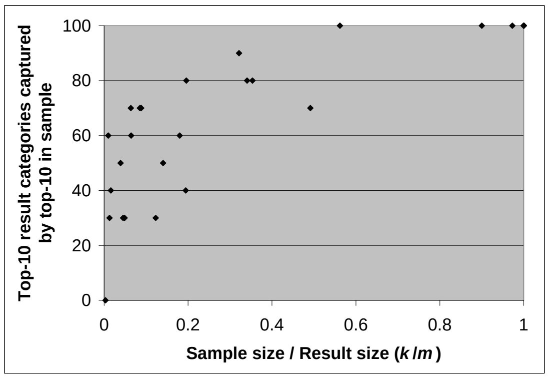
Table 4.4: Results from experiments on discovering popular categories using a random sample.

A final important remark, explains the poor performance in most of the cases of Table 4.3(b), compared with Table 4.3(a). Let us focus, for concreteness, on Q_9 (corresponding to the query "Indian AND (America OR Asia)"). The total number of matching documents is 15721, and the sample of size 50 fails completely to identify the frequent categories, while the sample of size 200 also fails to spot out the most frequent categories in Table 4.3(b) (although, notice in Table 4.3(a) that it does manage to sample some documents related to 9 out of the 10 frequent categories). This is due to the Eureka categorization: the 3000 categories used to tag the documents are very fine, resulting in documents matching very specific categories. For query Q_9, the 15721 matching documents were found to be related to 1935 categories, from which we tried to extract the top 10. Each of these categories contains a rather small number of documents: the most frequent one contains 125 documents, the 10th most frequent contains 54; the accumulated mass in the top 10 categories (sum of the number of documents contained within the top 10 categories) is 753, while the total mass is 9404. Therefore, each of the 50 sampled documents, has less than 10% chance to be a document contained within the top-10 categories, and negligible probability (0.57%) to be contained within the top 10th category.

The solution to this categorization artifact is straightforward: after obtaining the samples, we must aggregate the categories to coarser super-categories according to the taxonomy (e.g., the categories Lions, Cheetahs and Monkeys can be aggregated to Mammals, or Animals). Then the final result is a sample of a smaller number of categories each with a large mass, in which case even a



(a) Number of the top-10 frequent categories that appear in the samples.



(b) Number of the top-10 frequent categories that appear in the 10 most frequent sample categories.

Figure 4.5: Results from experiments on discovering popular categories using a random sample.

small sample size can efficiently discover the popular super-categories and present them to the user. Since the emphasis of our work lies mainly on the method for sampling, we have not pursued this line of research any further.

4.5.2 Estimating the Size of the Result Set

Finally we evaluate the quality of the estimator for the size of the result set. Table 4.5 shows the estimates and the relative errors. We mention again that many commercial Web search engines fail to provide an accurate estimation of the number of results. In contrast, notice that for even the smallest sampling size the error never exceeds 15%, and usually it is negligible for a sample size greater than 200. Here, however, the fact that the result sets are rather small plays to our advantage: our samples are relatively large, occasionally larger than the result set. Further research is needed to elucidate the case of very large result sets and to compare against the current performance of commercial search engines.

Query	Match Count	50		200		1000	
		Est.	Err.	Est.	Err.	Est.	Err.
Q_1	587	562	4.3	597	1.7	587	0
Q_2	5109	5388	5.5	5088	0.4	5050	1.1
Q_3	3111	2652	14.8	3376	8.5	3150	1.3
Q_4	1111	1119	0.7	1150	3.5	1111	0
Q_5	407	433	6.4	395	2.9	407	0
Q_6	1028	1172	14.0	989	3.8	1028	0
Q_7	356	316	11.2	356	0	356	0
Q_8	566	545	3.7	596	5.3	566	0
Q_9	15721	17028	8.3	15448	1.7	15902	1.2

Table 4.5: Evaluation of the estimates for the sizes of the query results. The table shows the actual value, and for each sampling size the estimate and the percentage of the error.

Bibliography

[1] C. C. Aggarwal, S. C. Gates, and P. S. Yu. On using partial supervision for text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):245–255, 2004.

[2] W. Aiello, B. Awerbuch, B. M. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC '93)*, pages 632–641, May 1993.

[3] E. Amitay, D. Carmel, R. Lempel, and A. Soffer. Scaling IR-system evaluation using term relevance sets. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '04)*, pages 10–17. ACM Press, 2004.

[4] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *Proceedings of the 14th International World Wide Web Conference 2005 (WWW '05)*, pages 245–256, 2005.

[5] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Stability and efficiency of a random local load balancing protocol. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 472–481, Oct. 2003.

[6] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Load balancing in arbitrary network topologies with stochastic adversarial input. *SIAM Journal on Computing*, 34(3):616–639, June 2005.

[7] A. Anagnostopoulos, I. Kontoyiannis, and E. Upfal. Steady state analysis of balanced-allocation routing. *Random Structures and Algorithms*, 26(4):446–467, 2005.

[8] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS '96)*, pages 380–389. IEEE, 1996.

[9] E. Anshelevich, D. Kempe, and J. Kleinberg. Stability of load balancing in dynamic adversarial systems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 399–406, New York, May 2002.

[10] G. R. Ash, R. H. Cardwell, and R. P. Murray. Design and optimization of networks with dynamic routing. *The Bell System Technical Journal*, 60, 8(8):1787–1820, 1981.

[11] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. In *Proceedings of the 26th ACM Symposium on the Theory of Computing (STOC '94)*, pages 593–602. ACM Press, 1994.

[12] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, Feb. 2000.

[13] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, pages 633–634. Society for Industrial and Applied Mathematics, 2002.

[14] J. P. Bagrow and D. ben-Avraham. On the *Google*-fame of scientists and other populations. In *Proceedings of the 8th Granada Seminar on Computational and Statistical Physics, "Modeling Cooperative Behavior in the Social Sciences"*, pages 81–89, 2005.

[15] P. Berenbrink, T. Friedetzky, and L. A. Goldberg. The natural work-stealing algorithm is stable. *SIAM Journal on Computing*, 32(5):1260–1279, Oct. 2003.

[16] J. E. Bertram and P. E. Sarachik. Stability of circuits with randomly time-varying parameters. *IRE Trans. Circuit Theory*, 6:260–270, 1959.

[17] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, Jan. 2001.

[18] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *WWW7/Computer Networks and ISDN Systems*, 30:107–117, April 1998.

[19] A. Z. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[20] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM '03)*, pages 426–434. ACM Press, 2003.

[21] A. Z. Broder, A. Frieze, C. Lund, S. Phillips, and N. Reingold. Balanced allocations for tree-like inputs. *Information Processing Letters*, 55(6):329–332, Sept. 1995.

[22] A. Z. Broder, A. M. Frieze, and E. Upfal. A general approach to dynamic packet routing with bounded buffers. *Journal of the ACM*, 48(2):324–349, 2001.

[23] M. Burrows. Sequential searching of a database index using constraints on word-location pairs. United States Patent 5 745 890, 1998.

[24] D. Carmel, E. Amitay, M. Herscovici, Y. S. Maarek, Y. Petruschka, and A. Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proceedings of the Tenth Text REtrieval Conference (TREC-10)*. National Institute of Standards and Technology (NIST), 2001.

[25] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[26] D. Down, S. P. Meyn, and R. Tweedie. Exponential and uniform ergodicity of Markov processes. *The Annals of Probability*, 23(4):1671–1691, 1996.

[27] D. Fallows, L. Rainie, and G. Mudd. The popularity and importance of search engines, August 2004. The Pew Internet & American Life Project, `http://www.pewinternet.org/pdfs/PIP_Data_Memo_Searchengines.pdf`.

[28] W. Feller. *An Introduction to Probability Theory and its Application, Vol. 2.* John Wiley and Sons, New York, second edition, 1971.

[29] M. Fontoura, E. J. Shekita, J. Y. Zien, S. Rajagopalan, and A. Neumann. High performance index build algorithms for intranet search engines. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB '04)*, pages 1158–1169. Morgan Kaufmann, 2004.

[30] F. G. Foster. On the stochastic matrices associated with certain queuing processes. *Annals of Mathematical Statistics*, 24:355–360, 1953.

[31] B. Ghosh, F. T. Leighton, B. M. Maggs, S. Muthukrishnan, C. G. Plaxton, R. Rajaraman, A. W. Richa, R. E. Tarjan, and D. Zuckerman. Tight analyses of two local load balancing algorithms. *SIAM Journal on Computing*, 29(1):29–64, Feb. 2000.

[32] B. Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *Journal of Computer and System Sciences*, 53(3):357–370, Dec. 1996.

[33] R. J. Gibbens, P. J. Hunt, and F. P. Kelly. Bistability in communication networks. In G. R. Grimmet and D. J. A. Welsh, editors, *Disorder in Physical Systems*, pages 113–128. Oxford Univ. Press, New York, 1990.

[34] R. J. Gibbens, F. P. Kelly, and P. B. Key. Dynamic alternative routing. In M. E. Steenstrup, editor, *Routing in Communications Networks*, pages 13–47. Prentice Hall, 1995.

[35] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '01)*, pages 281–291. ACM Press, 2001.

[36] L. A. Goldberg and P. D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. *Journal of Computer and System Sciences*, 58(1):232–258, 1999.

[37] J. Goodman, A. G. Greenberg, N. Madras, and P. March. Stability of binary exponential backoff. *Journal of the ACM*, 35(3):579–602, July 1988.

[38] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a WebFountain: An architecture for very large-scale text analytics. *IBM Systems Journal*, 43(1), 2004.

[39] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web (WWW '05)*, pages 902–903. ACM Press, 2005.

[40] P. J. Haas, J. F. Naughton, and A. N. Swami. On the relative cost of sampling for join selectivity estimation. In *Proceedings of the Thirteenth ACM Symposium on Principles of Database Systems (PODS '94)*, pages 14–24. ACM Press, 1994.

[41] J. Håstad, T. Leighton, and B. Rogoff. Analysis of backoff protocols for multiple access channels. *SIAM Journal on Computing*, 25(4):740–774, Aug. 1996.

[42] P. J. Hunt and C. N. Laws. Asymptotically optimal loss network control. *Mathematics of Operations Research*, 18(4):880–900, 1993.

[43] S. Karlin and H. M. Taylor. *A First Course in Stochastic Processes.* Academic Press, New York, second edition, 1975.

[44] I. Y. Kats and N. N. Krasovskii. On stability of systems with random parameters. *Prikl. Mat. Mekh.*, 24:809–823, 1960. (In Russian).

[45] F. P. Kelly. Loss networks. *Annals of Applied Probability*, 1(3):319–378, 1991.

[46] H. J. Kushner. *Stochastic Stability and Control.* Academic Press, 1967.

[47] K.-H. Li. Reservoir-sampling algorithms of time complexity $O(n(1 + \log(N/n)))$. *ACM Transactions on Mathematical Software*, 20(4):481–493, 1994.

[48] M. J. Luczak, C. McDiarmid, and E. Upfal. On-line routing of random calls in networks. *Probability Theory and Related Fields*, 125:457–482, 2003.

[49] M. J. Luczak and E. Upfal. Reducing network congestion and blocking probability through balanced allocation. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS '99)*, pages 587–595, 1999.

[50] R. Lüling and B. Monien. A dynamic distributed load balancing algorithm with provable good performance. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, pages 164–172, July 1993.

[51] J. Martin and Y. Suhov. Fast Jackson networks. *Annals of Applied Probability*, 9(3):854–870, 1999.

[52] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (ICALP '93)*, pages 398–409, July 1993.

[53] S. P. Meyn and R. Tweedie. Stability of Markovian processes III: Foster-Lyapunov criteria for continuous-time processes. *Advances in Applied Probabability*, 25:518–548, 1993.

[54] S. P. Meyn and R. Tweedie. A survey of Foster-Lyapunov techniques for general state space Markov processes. In *Proceedings of the Workshop on Stochastic Stability and Stochastic Stabilization, Metz, France, June 1993*. Springer-Verlag, 1994.

[55] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Communications and Control Engineering Series. Springer-Verlag, London, New York, 1993.

[56] M. Mitzenmacher. Load balancing and density dependent jump Markov processes. In *Proceedings of 37th Conference on Foundations of Computer Science (FOCS '96)*, pages 213–222, Oct. 1996.

[57] M. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, August 1996.

[58] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, pages 292–301. ACM Press, June 22–25, 1997.

[59] M. Mitzenmacher. Analyses of load stealing models based on differential equations. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '98)*, pages 212–221, June 1998.

[60] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[61] S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the Fourteenth Annual ACM Symposium on Discrete Algorithms (SODA '03)*, pages 413–413. ACM Press, 2003.

[62] S. Muthukrishnan and R. Rajaraman. An adversarial model for distributed dynamic load balancing. In *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '98)*, pages 47–54, June 1998.

[63] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18(2):229–243, Apr. 1989.

[64] R. Pemantle and J. S. Rosenthal. Moment conditions for a sequence with negative drift to be uniformly bounded in L^r. *Stochastic Processes and their Applications*, 82:143–155, 1999.

[65] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load balancing schemes. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS '98)*, pages 694–705, Nov. 1998.

[66] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan, and J. Prager. Mining the web for answers to natural language questions. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM '01)*, pages 143–150. ACM Press, 2001.

[67] S. M. Ross. *Applied Probability Models with Optimization Applications.* Dover Publications, Reprint, 1970.

[68] S. M. Ross. *Applied Probability Models with Optimization Applications.* Holden-Day, 1970.

[69] S. M. Ross. *A First Course in Probability.* Macmillan, London, 5th edition, 1998.

[70] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal. A simple load balancing scheme for task allocation in parallel machines. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '91)*, pages 237–245, July 1991.

[71] J. Shaler Stidham. A last word on $L = \lambda W$. *Operations Research*, 22(2):417–421, 1974.

[72] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[73] Y. Suhov and N. Vvedenskaya. Fast Jackson networks with dynamic routing. *Problems of Information Transmission*, 38(2):136–153, 2002.

[74] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

[75] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[76] N. Vvedenskaya, R. Dobrushin, and F. Karpelevich. A queueing system with a choice of the shorter of two queues – an asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.

[77] D. Williams. *Probability with Martingales.* Cambridge University Press, 1991.

[78] C. Xu and F. Lau. Iterative dynamic load balancing in multicomputers. *Journal of the Operational Research Society*, 45(7), July 1994.

[79] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 401–408. ACM Press, 2003.

[80] W. Zhu, C. Steketee, and B. Muilwijk. Load balancing and workstation autonomy on Amoeba. *Australian Computer Science Communications*, 17(1):588–597, 1995.