

Analysis of Randomized Protocols for Conflict-Free Distributed Access*

Gopal Pandurangan[†] Gahyun Park[†]

Abstract

We study the following distributed access problem which arises naturally in many settings: given a set of data items shared among nodes in a distributed network, all nodes want to access all (or a subset of) the items residing on different nodes in a conflict-free manner. In addition, items may move from one node to the other during access. The goal is to design distributed protocols so that all nodes access all the desired items as quickly as possible, while at the same time not overloading the storage space of any one node. We show that a simple randomized distributed protocol (*Protocol 1* of the paper) performs *almost as well as an optimal (centralized) scheme* but with no coordination overhead. Our protocol takes $O(n)$ time with high probability to access all n items which is asymptotically the same compared to the optimal scheme. The protocol guarantees that the maximum load of any processor is at most $O(\log n / \log \log n)$ with high probability which is only slightly larger compared to the $\Omega(1)$ load of the optimal scheme. Our analysis involves a stochastic analysis of a “balls into bins” problem in a dynamic setting where balls (data items) move into bins (nodes) on request and we study the time and load requirements to move all the balls to the requested bins.

Keywords: Randomized Distributed Protocols; Conflict-Free Access.

*A short version of this paper appeared in the Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2005.

[†]Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA. E-mail: {gopal, gpark}@cs.purdue.edu.

1 Introduction and Motivation

In many distributed settings, data and resources are shared and there are constraints on their simultaneous access by participating nodes. For example, in ad hoc wireless networks, nodes have simultaneous wireless transmission constraints, and in peer-to-peer networks nodes have bandwidth constraints. Additionally, in such settings, communication and network management is expensive and hence *distributed protocols* with low communication and coordination overhead are preferred. Thus an important goal in such settings is to design *distributed conflict-free access protocols* so that all the competing nodes can access the shared resources in a coordinated manner. In this paper, we focus on the following distributed access problem which naturally arises in many settings: given a set of data items shared among nodes (computers/processors) in a distributed network, all nodes want to access all (or a subset of) the data items residing on different nodes. In many settings, there may be constraints on the number of items that can be accessed simultaneously by a node (e.g., due to limited bandwidth or storage space) or the number of different nodes that can access an item simultaneously (e.g., an item needs to be accessed exclusively by a node for maintaining data consistency at all times). The goal is to design a distributed access protocol so that all the nodes will access all the data as quickly as possible without any centralized coordination. To illustrate further, consider a concrete example of a Peer-to-Peer (P2P) network — a distributed ad hoc network of computers sharing data and resources. A large file (or a set of large files) may be distributed and stored across the network and a large set of nodes want to access all (or some number of) the distributed pieces of the file (or the set of files). In content-addressable P2P networks (e.g., Chord [38]), given a data item’s key any node can access the item efficiently in a distributed manner (even without knowing a priori where the item is located). However, bandwidth is usually a severe constraint, and hence it is not possible for a node to access all the pieces at the same time, nor it is possible for all the “home” nodes (the nodes where the data resides) to service many requests for the data at the same time. Broadcasting all the items to all the nodes may not be possible since the files are large and replicating data may be costly. Hence, the goal is to design an access schedule so that all the competing nodes can access all the distributed data in a coordinated manner. One way is to devise a centralized coordination scheme; although this can be time optimal, just to set up the centralized scheme in a fully distributed setting may be too costly.

There are also other applications where the distributed access problem naturally arises. For example, consider the Information Dispersal scheme of Rabin [34] in a resource-constrained setting. Here a file F of length $L = |F|$ is broken into n pieces F_i , each of length $|F_i| = L/m$, so that every m pieces suffice for reconstructing F . Thus, each node has some piece, all nodes wants to access at least m pieces ($m-1$ if each node has one piece to begin with) in a distributed manner. Our problem is relevant if there are constraints on simultaneous access due to bandwidth, storage, or exclusive access. Another interesting application is computing aggregates in a decentralized manner. Here each node wants to compute some (global) aggregate function (such as average, max) determined by all other nodes’ values. For example, the simple and elegant Push-Sum protocol of Kempe et al. [21] guarantees that in a complete network (i.e., every node can communicate with any other node in one step) within $O(\log n)$ parallel steps (with high probability) all nodes will have an accurate estimate of an aggregate function (such as average). This is a distributed protocol where in every

step, each node chooses some other node uniformly at random, and sends some value. This result does not assume any constraints on the number of items a node can receive at the same step. Our model and results are relevant when there is an constraint on the number of items a node can receive at the same step.

Also, in some applications, e.g., in distributed databases (e.g., see [31]) or in certain P2P systems [17], nodes may move data items around and update or modify those data items. For update consistency, it is desirable to have one copy of each item around and thus the node which accesses the data item becomes the new “home” node for the data, while the node from which it got the data relinquishes ownership (with corresponding increase and decrease in load of the respective nodes). Ideally, we would like the load to be as evenly distributed as possible during the algorithm and at the end of the algorithm. Again, this can be achieved by centralized coordination schemes, but here we are interested in fully distributed protocols which perform as well as these schemes with very little or no coordination overhead.

Motivated by the above discussion, we consider a simple and abstract distributed access problem where all nodes simultaneously want to access all (or a subset) of items in other nodes in a conflict-free manner. We study a simple and natural, randomized, fully distributed protocol that performs as well as an optimal centralized scheme, but has very little overhead.

1.1 Problem Statement and Protocols

Consider a network of n nodes (processors) and each processor has one (distinct) data item (for a total of n items). We assume that data items have unique keys (names) and each processor knows the respective keys of all the n data items. Each processor has a unique processor identification number. The goal of each processor is to access all (or some number $m < n$) of the data items located in the other processors. We will assume a synchronous setting, i.e., communication is synchronous and occurs in discrete time steps. In one step, a processor can request only one data item from one other processor. Thus each processor can distinguish the data items, but they may not know where each item is located. A processor will request a data item by specifying its key. We assume that a processor can locate (and access) an item in one step if it knows its key, no matter where this item is located currently. If more than one processor requests the same data item in the same step only one will be able to access it (ties can be broken arbitrarily). Success or failure to access an item will be known in the same time step. (We point out that, although the terminology “conflict-free” used in this paper suggests “exclusive access”, the model and the protocols studied here can be easily relaxed to allow an *upper bound* on simultaneous access. This extension is discussed in Section 4.)

We note that in this highly simplified model we ignore the delay (or equivalently, assume that all delays are the same) that will result in locating and accessing an item. In fact, we do not worry about how the item is located in the network and simply assume the existence of an underlying network protocol (e.g., similar to content-addressable networks [38, 35]) that will locate and return the item to the requested processor if the request is conflict-free. Our main goal here is to study a “clean” model that focuses only on the effects of conflict-free access. Thus we abstract away difficulties arising from other factors, such as delay, congestion, asynchrony, routing etc. (In a

similar vein, the simplified distributed computing model called *LOCAL* is used to study only the effects of *locality*, while ignoring other considerations [33]. We also note that simplified point-to-point “complete network” models in which every node can communicate with any other node in one time step has been used in theory to model overlay networks (e.g., [25]) and gossip-based communication, e.g., [21, 22].)

We would like to design a simple and natural distributed algorithm where there is no coordination of requests among nodes and every node acts independently in a distributed (and selfish) manner. Our *first goal* is to minimize (or reduce) the *time* taken for all the processors to access all (or a specified number of) the items. Suppose we assume further in our model that the item *moves* to the requested processor after (successful) access. That is, when processor *A* accesses an item from processor *B*, *A*’s load increases by 1 and *B*’s load decreases by 1. We will assume that the movement of an item will affect only the loads of the involved processors and will not affect the time needed to access the item in future time steps. In this setting, our *second goal* is to minimize the *load* of all the processors during the entire course of the algorithm. Also, on termination we are interested in minimizing the number of data items each node owns i.e., we want to achieve load balance after all requests have been serviced.

If the nodes can somehow coordinate among themselves (e.g., a centralized scheme that matches items to processors in a cyclical schedule that is conflict-free in each step) all nodes can access all the items in $n - 1$ steps storing only one item at any time and this obviously is optimal with respect to both time and load. Our goal is to design a distributed protocol that tries to perform as well as an optimal scheme. We want to leverage the fact that large number of nodes want to access a large number of items to schedule the accesses in a contention-free manner.

This paper is mainly devoted to the study of the following simple and natural, distributed, randomized protocol:

Protocol 1:

In each step every processor does the following:

- If there are items still to be accessed
- request an item chosen uniformly at random from among the items it has **not** yet accessed.

It is easy to implement the above protocol in our model. Each processor maintains its own list of items (identified by their unique keys) that it has accessed till now. Assume that the goal of each processor is to access all n (or only $m < n$) items. At the beginning of a step, each processor checks if it has accessed all n (or $m < n$) items. (Note that this is possible, since a processor knows the keys of all the n items.) If a processor has not accessed the required number of items then it chooses an item (say item p) randomly from among the items that it has not accessed till now (“sampling without replacement”). It then sends a “request” message for p (with p ’s key and its processor id in the message). If this is the only processor requesting this item in this step, then it will (successfully) access the item. We assume that success or failure to access is checked by the underlying network protocol and an appropriate notification is sent back to the processor in the same time step. The protocol terminates when all items have been accessed by the processor. In case of success, the item *moves* to the requested processor with corresponding changes in the

loads of the involved processors. In Section 2 we formally show the termination and correctness of Protocol 1 (cf. Theorem 2.1). We then analyze the performance of Protocol 1 with respect to the following measures (cf. Theorems 2.3, 3.1 and 3.2):

1. **Time:** The number of steps needed before *all* processors access all (or some m number of) items.
2. **Load:** The *maximum* load of any processor in any step during the execution of the protocol.

Although the algorithm is simple, its analysis is not, because of the complicated dependencies introduced soon after the first step. These dependencies make the analysis of Protocol 1 significantly more complicated than the analysis of the standard “balls into bins” paradigm. As is a common simplifying technique in analyzing randomized algorithms, one way is to analyze a somewhat less efficient protocol which uses “sampling with replacement” instead of “sampling without replacement” paradigm (used by Protocol 1). (We note that in some applications, e.g., finding a stable marriage [28] or finding Hamiltonian paths in random graphs [3], gives a bound which is asymptotically tight compared to the original algorithm which uses “sampling without replacement” paradigm. Here, as the reader will see, this is not the case: the simplified protocol below has asymptotically much worse performance compared to Protocol 1.)

Protocol 2:

In each step every processor does the following:

- If there are items still to be accessed
- request an item chosen uniformly at random from among **all** items.

Similar to Protocol 1, it is easy to implement Protocol 2. In every step, each processor checks the list of items accessed as in Protocol 1. In case the number of items accessed is less than n (or $m < n$) then the choosing of item is done uniformly at random from among all the n items (“sampling with replacement”). Protocol 2 is, obviously, naive to implement (since a node can unnecessarily request items which it has already accessed), however its expected time to terminate is easier to analyze and this gives an *upper bound* on the running time of Protocol 1. Using a coupon collector argument we show that the number of steps needed by Protocol 2 is $\Theta(n \log n)$ with very high probability (cf. Theorem 2.2), $\Theta(\log n)$ factor larger than the lower bound of $\Omega(n)$ (cf. Theorem 2.3). In contrast, we will show that Protocol 1 takes at most $O(n)$ steps *with very high probability (wvhp)*, i.e., with probability at least $1 - e^{-\delta n}$, for some $\delta > 0$. The load is also more difficult to analyze in Protocol 1 compared to Protocol 2. For Protocol 2, it is not difficult to show that the load is $\Theta(n)$, since the last processor alive (when all the other processors have accessed all the items and stopped) would need to sample $\Theta(n)$ items on the average. However, we will show that for Protocol 1, the maximum load of any processor is upper bounded by $O(\log n / \log \log n)$ *with high probability (whp)* i.e., with probability at least $1 - 1/n^\delta$, for some $\delta > 0$ (cf. Theorem 3.1). We will also show that the load in some processor will be at least $\Omega(\sqrt{\log n})$ whp in *every* step $t \geq \sqrt{\log n}$ (cf. Theorem 3.2). (Henceforth, we will use the abbreviations “wvhp” and “whp” in the sense defined above.)

We conclude this section by pointing out a quick application to the problem of computing aggregates discussed earlier. If each node accesses all the other values, then Protocol 1 takes at most $O(n)$ steps. However, in many situations (e.g., under the assumption that the values are drawn from the same distribution), sampling a small number of other nodes' values is enough to give an estimate of the aggregate (say, average) with high probability (see e.g., [9]). Thus, if only $\Theta(\log n)$ other values need to be accessed, then Protocol 1 terminates in $\Theta(\log n)$ steps whp (cf. Theorem 2.3).

Paper Organization. We next briefly discuss work that is most relevant to this work. The rest of the paper is mainly devoted to the analysis of Protocol 1. The correctness, termination, and time bounds are discussed in Section 2. The load bounds are stated and proved in Section 3. Extensions and open problems are discussed in Section 4.

1.2 Related Work

Conflict-free access protocols have been studied extensively in the context of multiaccess channels (see e.g., an early survey by Gallager [12]), but these are not directly related to our model. In a multiaccess channel problem, a common channel is shared by many competing processors. Each processor needs the channel exclusively to communicate and if more than one processor communicates in the same time step then none will succeed. The goal is to devise a local protocol that schedules the accesses in a conflict-free manner. Typically, the assumption is that a processor does not have knowledge of the intentions or the history of other processors and has to rely solely on its local history to design its schedule. Randomized *backoff* protocols that repeatedly retries sending a message after a suitably chosen random time interval have been extensively studied [15, 36, 13, 14]. Randomization has also been used to design efficient collision-resolution protocols in the multiple access channel model of radio networks. In this setting, the basic assumption is that each node can send a message to all its neighbors in one step, but a neighbor w can hear the message only if no other neighbor of w is transmitting at the same step [5]. We refer to the survey by Chlebus [6] for overview of work on randomized communication in radio networks.

There is also work on randomized protocols for gossip/rumor propagation in networks, but these usually assume that data (typically of small size) can be replicated and propagated simultaneously. In many of these applications, the motivation for using randomization is that it naturally provides robustness, simplicity, and scalability. For example, Feige et al. [11] studies the following randomized protocol for broadcasting an item from a source node to the entire network. Every node that receives a copy of the item (starting with the the source node) chooses a random neighbor and sends a copy of the item to it. The main result is the analysis of the time needed for a copy to reach all the nodes in the network. Another related work is the work on randomized rumor spreading by Karp et al [18]. This work assumes a model in which there are n nodes that communicate in parallel rounds in each of which every player calls a randomly selected communication partner and exchanges a rumor. Starting with a single rumor, the goal is to propagate the rumor to all n nodes. Specifically, two classes of algorithms can be defined based on the direction of exchange: *push* and *pull*. Assuming u call v , the rumor is said to be pushed if u tells v the rumor, and the rumor is said to be pulled if v tells u the rumor. The main results of the work is the design of a protocol

that uses both push and pull to design an algorithm that takes $O(\log n)$ rounds and $O(n \log \log n)$ transmissions to spread the rumor to all nodes. It is also shown that this is essentially asymptotically optimal in the sense that any *address-oblivious* algorithm needs $\Omega(n \log \log n)$ transmissions for each rumor regardless of the number of rounds. It also shows that there exists a fundamental performance gap between rumor spreading algorithms based on random interconnections and deterministic broadcasting schemes. We note that protocols of this paper can be considered as “pull” protocols in the sense that the item tries to pull the data item from a node having the item. Other recent works on using randomization in decentralized gossip-based protocols include [21, 22, 20, 19].

There is a vast literature on the balls and bins problems (also called “occupancy” problems) where m balls are thrown independently and randomly into n bins and the goal is to study questions such as the maximum number of balls in a bin, the expected number of bins with k balls etc. Comprehensive treatises on these problems are the books by Johnson and Kotz [16] and by Kolchin et al [23]. A coordinated placement of balls will result in 1 ball in every bin and this gives the optimal load distribution. Throwing balls randomly, requires no coordination, and yet achieves a maximum load of at most $O(\log n / \log \log n)$ with high probability ([28]). In fact, this can be shown to be optimal in the sense that some bin will have this load. The work of Azar et al. [4] builds on this basic occupancy problem and introduces the *two choices* version of the problem where each ball is thrown into the least loaded of *two* randomly chosen bins. This leads to even more balanced allocation: the maximum load in any bin is now $O(\log \log n)$. A comprehensive survey of the two choices paradigm and its applications is the survey by Mitzenmacher et al. [26]. The “balls into bins” model, although quite simple, has played a fundamental role in many algorithmic applications — randomized load balancing, hashing, routing, analyzing random graph algorithms to name a few (e.g., [28, 27, 4, 26]). In particular, “balls into bins” paradigm have been used in the design and analysis of parallel and distributed algorithms for load balancing (e.g., see [1, 29, 26] and the references therein), edge-coloring [32], fault-tolerant decentralized computing (e.g., the survey of [29] and the more recent work of Chlebus and Kowalski [7] and references therein), and packet routing (the survey of [37]). The paper of Dubhashi and Ranjan [10] studies negative dependence in the analysis of balls and bins and discusses useful analytical tools such as the FKG inequality (cf. Section 3).

2 Time Bound and Analysis

We first formally show that Protocol 1 terminates and is correct, i.e., it terminates in a finite number of steps (with certainty) and on termination all the n processors would have accessed all the n items. The same proof will work if only $m < n$ items need to be accessed by all processors.

Theorem 2.1 *Protocol 1 is correct and terminates in at most n^2 steps.*

Proof: The main argument for termination is that, in every step, *some* processor will make progress as explained below. Note that if two processors access the same item in the same time step, one of them will succeed. Once an item is successfully accessed by a processor, it will not again be chosen for access by the same processor. Define the stochastic process $Z = Z_0, Z_1, \dots, Z_t, \dots$,

where the random variable Z_t counts the *total* number of successful item accesses by *all* processors up to step t (starting from step 0). In the beginning, each processor has only its own item, and thus $Z_0 = n$. The protocol will terminate (for all processors) when $Z_t = n^2$. Since, at least one item is successfully accessed by some processor in each step, the stochastic process is monotonically increasing, i.e., $Z_{t+1} \geq Z_t + 1$, for all t . Hence the protocol will terminate correctly before $t \leq n^2$ steps. ■

Note that the n^2 termination time bound is a worst-case bound. We note that $\Omega(n)$ is an obvious lower bound. As discussed in Section 1.1, we show that Protocol 2 correctly terminates in $\Theta(n \log n)$ steps with very high probability. This gives an (non-tight) upper bound on the termination of Protocol 1, since it is easy to see that the running time of Protocol 2 stochastically dominates the running time of Protocol 1.

Theorem 2.2 *Protocol 2 correctly terminates in $\Theta(n \log n)$ steps in expectation and wvhp.*

Proof: Protocol 2 terminates only when all processors have accessed all the items. We now bound the number of steps needed for termination with high probability.

We first get a high probability bound on the number of steps needed by a particular processor (say processor 1) to access all items.

The probability that processor 1 did *not* access a particular item in some step is at most

$$((1 - 1/n) + (1/n)(1 - (1 - 1/n)^n)) \leq 1 - 1/n(1 - 1/e).$$

In the above formula, the first term is the probability that this particular item is *not chosen* to be requested and the second term represents the probability that the item is chosen and was not accessed (since some other processor accessed the same item in this step). Thus the probability that this particular item is not accessed in $cn \log n$ steps is at most

$$(1 - 1/n(1 - 1/e))^{cn \log n} \leq 1/n^3$$

for a suitably large constant c .

Hence using the union bound ([27][Lemma 1.2]), the probability that there exists some item that is not accessed by processor 1 in $O(n \log n)$ steps is at most $1/n^2$. Using the union bound again, the probability that there is some processor which has not accessed all the items is at most $1/n$.

The lower bound follows directly from the coupon collector argument: A processor needs $\Omega(n \log n)$ steps wvhp even to *choose* all n items to request ([28][Theorem 3.8]). ■

The main result of this section is the following theorem which gives an asymptotically tight bound on the running time of Protocol 1.

Theorem 2.3 *1. Let T be the number of steps before all the n processors access all the n items in Protocol 1. Then $T = O(n)$ with probability at least $1 - e^{-\delta n}$, for some $\delta > 0$.*

2. Let T_m be the number of steps needed before all n processors access (any) m ($< n$) items. Then the following holds.

- (a) *For $m = \Omega(\log n)$, $T_m = O(m)$ whp.*
- (b) *For $m = o(\log n)$, $T_m = O(\log n)$ whp.*

Proof: We will prove Part 1; Part 2 will be a straightforward generalization.

The main idea is to show that after cn steps (for some constant $c > 0$) each processor has accessed at least dn of the items (wvhp) and each item has been accessed at least dn of the processors (wvhp), for some constant $d < 1$, and then we use this repeatedly to get a geometric sum. We now formalize this idea as follows.

Consider the following less efficient protocol where *sampling with replacement* is combined with *sampling without replacement*. Partition a set of steps into rounds such that k -th round consists of $c_k n$ steps, where c_k is a parameter that is independent of n (but dependent on k) that will be fixed later. More precisely, first round consists of step 1 through $c_1 n$, and k -th round, for $k > 1$, consists of step $\sum_{i=1}^{k-1} c_i n + 1$ through $\sum_{i=1}^k c_i n$.

Protocol 3:

In each step of the k th round every processor does the following:

- if there are items still to be accessed
- request an item chosen uniformly at random from among the items
- it has not yet accessed in previous rounds.

In particular, note that each processor requests an item from among all items in the first round. Note that this can be considered as a hybrid of Protocols 1 and 2. There is sampling with replacement in every round (that consists of a finite number of steps) as in Protocol 2. And in every round, the sampling is restricted to items that have been accessed in previous rounds — sampling without replacement as in Protocol 1. Correctness is obvious because the protocol terminates (for every processor) only when all items are accessed by all processors. Let M' be the number of steps before all the n processors access all the n items in the above protocol. We will now show that $M' = O(n)$ wvhp. It is not difficult to see that M' stochastically dominates M , the number of steps taken by Protocol 1 and thus we bound M by bounding M' .

We assume that at the beginning, each processor has no items (having one item already does not change the analysis) and it starts requesting an item from step 1. For the first round, let $I_1 = \{1, 2, \dots, n\}$ be the set of all items. Let $X_{ij}, j \in I_1$ be a random variable defined to be 1 if processor i accesses item j during the first round and 0 otherwise. Let X_i be the total number of items accessed by processor i during the first round. The probability that processor i accesses item j at some step $s < c_1 n$ is at least

$$\frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{en}.$$

Thus we have

$$\Pr(X_{ij} = 0) \leq \left(1 - \frac{1}{en}\right)^{c_1 n} \leq e^{-c_1/e}.$$

Since $X_i = \sum_{j \in I_1} X_{ij}$, we have

$$\mathbf{E}[X_i] \geq d_1 n,$$

where $d_1 = 1 - e^{-c_1/e}$. Let Y_j be the number of processors that have accessed item j during the first round. Using the same arguments, we can show that

$$\mathbf{E}[Y_j] \geq d_1 n.$$

To show concentration of X_i which is a sum of *dependent* random variables, we use a martingale analysis. Specifically, we set up a Doob martingale ([28, Chapter 4]) to show that with wvhp

$$X_i \geq d_\epsilon n$$

where $d_\epsilon = (1 - \epsilon)d_1$, for some small constant $\epsilon > 0$. Let $Z_0 = \mathbf{E}[X_i]$, $Z_1 = \mathbf{E}[X_i \mid X_{i1}]$, \dots , $Z_n = X_i = \mathbf{E}[X_i \mid X_{i1}, X_{i2}, \dots, X_{in}]$. Then the sequence Z_0, Z_1, \dots, Z_n is a Doob martingale ([28, Chapter 4]). We would like to show concentration of X_i by applying Azuma's inequality ([28, Theorem 4.16]):

$$\Pr(|Z_n - Z_0| \geq \lambda) \leq 2 \exp\left(-\frac{\lambda^2}{2\gamma^2 n}\right)$$

provided $|Z_j - Z_{j-1}| \leq \gamma$, for all j (called the ‘‘bounded difference property’’).

We next prove that $\gamma = 1$, i.e., that $|Z_j - Z_{j-1}| \leq 1$ for $j \in I_1$.

Claim 2.1 $|Z_{j+1} - Z_j| \leq 1$ for $j \in I_1$.

Proof: For notational convenience, let $U_j = U_{i,j}$, that is, U_j equals to 1 if processor i accesses item j during the first round, and let $U = \sum_{j=1}^n U_j$ be the number of items that processor i accesses in the first round. We define

$$f := f(U_1, \dots, U_{j+1}) = |Z_{j+1} - Z_j| = \mathbf{E}[U \mid U_1, \dots, U_j, U_{j+1}] - \mathbf{E}[U \mid U_1, \dots, U_j].$$

Then f becomes

$$f = U_{j+1} - \Pr(U_{j+1} = 1 \mid U_1, \dots, U_j) + \sum_{m=j+2}^n g_m,$$

where $g_m := \Pr(U_m = 1 \mid U_1, \dots, U_j, U_{j+1}) - \Pr(U_m = 1 \mid U_1, \dots, U_j)$. Since X_j s are identically distributed (by symmetry), we have

$$g_m := g = \Pr(U_n = 1 \mid U_1, \dots, U_j, U_{j+1}) - \Pr(U_n = 1 \mid U_1, \dots, U_j),$$

for all $m = j+2, \dots, n$. Thus we have

$$f = U_{j+1} - \Pr(U_{j+1} = 1 \mid U_1, \dots, U_j) + (n - j - 1)g. \quad (1)$$

We show that

$$-1 \leq f \leq 1, \quad \text{if } U_{j+1} = 1, \quad (2)$$

$$-1 \leq f \leq 1, \quad \text{if } U_{j+1} = 0. \quad (3)$$

We prove (2). Then (3) can be proved similarly. Assuming $U_{j+1} = 1$, we have

$$g = (\Pr(U_n = 1 \mid U_1, \dots, U_j, U_{j+1} = 1) - \Pr(U_n = 1 \mid U_1, \dots, U_j, U_{j+1} = 0)) \Pr(U_{j+1} = 0).$$

For $0 \leq k \leq j$, we set k variables among U_1, \dots, U_j to be 1 and the rest $j - k$ variables to be 0.

First, we show that

$$-\frac{1}{n - j - 1} \Pr(U_{j+1} = 0) \leq g \leq 0, \quad \text{for any } k. \quad (4)$$

Without loss of generality, we may assume that the first k variables set to 1 and the last $j - k$ variables are 0. Now we compute the probability

$$\Pr(U_n = 1 \mid U_1 = U_k = 1, U_{k+1} = \dots = U_j = 0, U_{j+1} = 1)$$

under the condition that U , the number of items accessed to processor i is $k + k'$ for $k' > 0$:

$$\begin{aligned} & \Pr(U_n = 1 \mid U_1 = \dots = U_k = 1, U_{k+1} = \dots = U_j = 0, U_{j+1} = 1, U = k + k') \\ &= \frac{\binom{n-j-2}{k'-2}}{\binom{n-j-1}{k'-1}} = \frac{k' - 1}{n - j - 1}. \end{aligned} \quad (5)$$

The above probability is obtained from the following observation. Since $k + 1$ items are set to 1, the number of cases in which $k' - 1$ items among U_{j+2}, \dots, U_n have been accessed by processor i is $\binom{n-j-1}{k'-1}$. Out of these, the number of cases in which U_n is accessed by the processor i is $\binom{n-j-2}{k'-2}$. Similarly, we can obtain that

$$\begin{aligned} & \Pr(U_n = 1 \mid U_1 = \dots = U_k = 1, U_{k+1} = \dots = U_j = 0, U_{j+1} = 0, U = k + k') \\ &= \frac{\binom{n-j-2}{k'-1}}{\binom{n-j-1}{k'}} = \frac{k'}{n - j - 1}. \end{aligned} \quad (6)$$

Therefore, for any $k' > 0$, the difference between (5) and (6) is $-\frac{1}{n-j-1}$. Note that if $k' \leq 0$ then the above probability becomes $0 - 0 = 0$. Thus we conclude that

$$\begin{aligned} -\frac{1}{n-j-1} &\leq \Pr(U_n = 1 \mid U_1 = \dots = U_k = 1, U_{k+1} = \dots = U_j = 0, U_{j+1} = 1) \\ &\quad - \Pr(U_n = 1 \mid U_1 = \dots = U_k = 1, U_{k+1} = \dots = U_j = 0, U_{j+1} = 0) \leq 0, \end{aligned}$$

this completes the proof for (4). From Equation (1), we have

$$-1 \leq 1 - \Pr(U_{j+1} = 1 \mid U_1, \dots, U_j) - \Pr(U_{j+1} = 0) \leq f \leq 1 - \Pr(U_{j+1} = 1 \mid U_1, \dots, U_j) \leq 1. \quad \blacksquare$$

By the above lemma, since $|Z_j - Z_{j-1}| \leq 1$ for $j \in I_1$, by Azuma's inequality, for any $\lambda > 0$,

$$\Pr(|Z_n - Z_0| \geq \lambda) \leq 2e^{-0.5\lambda^2/n}. \quad (7)$$

We want to show that wvhp, $Z_n = X_i \geq d_\epsilon n = (1 - \epsilon)d_1 n = (1 - \epsilon)E[X_i] = (1 - \epsilon)Z_0$. Letting $\lambda = \epsilon d_1 n$ in Equation 7, we obtain that $X_i \geq d_\epsilon n$ with probability at least $1 - 2e^{-0.5\epsilon^2 d_1^2 n}$, for any $\epsilon > 0$. Similarly we can show that $Y_j \geq d_\epsilon n$ holds wvhp. The bound for X_i is with respect to a particular processor i , and Y_j is respect to a particular item j . At the end of the first round, using the union bound it follows that with probability at least $1 - 2ne^{-0.5\epsilon^2 d_1^2 n}$, every processor has at most $(1 - d_\epsilon)n$ items left to be accessed, and at most $(1 - d_\epsilon)n$ processors have not accessed every item.

Let $K = \lfloor \frac{2}{3} \log_{(1-d_\epsilon)^{-1}} n \rfloor$. In general, for $1 < k \leq K$, we analyze the k -th round as the follows. At the end of the $(k - 1)$ -th round, at most $(1 - d_\epsilon)^{k-1} n$ items have not been accessed by processor

i . Let I_k be the set of $(1 - d_\epsilon)^{k-1}n$ items (in the worst case). Furthermore, wvhp there are at most $(1 - d_\epsilon)^{k-1}n$ processors that have not accessed item j for each $j \in I_k$. The probability that processor i accesses j at some step in the k -th round is at least

$$\frac{1}{(1 - d_\epsilon)^{k-1}n} \left(1 - \frac{1}{n(1 - d_\epsilon)^{k-1}}\right)^{(1-d_\epsilon)^{k-1}n} \geq \frac{1}{(1 - d_\epsilon)^{k-1}en}.$$

Let X_{ij}^k be 1 if processor i accesses item j during the k -th round and 0 otherwise, and X_i^k is the number of items accessed by processor i during the k -th round. Thus,

$$\Pr(X_{ij}^k = 0) \leq \left(1 - \frac{1}{(1 - d_\epsilon)^{k-1}en}\right)^{c_k n} \leq e^{-\frac{c_k}{e(1-d_\epsilon)^{k-1}}}.$$

Choose $c_k = c_1(1 - d_\epsilon)^{k-1}$ so that $\Pr(X_{ij}^k = 0) \leq e^{-c_1/e}$. Therefore,

$$\mathbf{E}[X_i^k] \geq d_1(1 - d_\epsilon)^{k-1}n.$$

Again using a Doob martingale and Azuma's inequality with $\lambda = \epsilon d_1(1 - d_\epsilon)^{k-1}n$, we have

$$X_i^k \geq d_\epsilon(1 - d_\epsilon)^{k-1}n$$

with probability at least $1 - 2 \exp(-0.5\epsilon^2 d_1^2(1 - d_\epsilon)^{k-1}n)$. Thus, during the first k rounds processor i has not accessed at most $(1 - d_\epsilon)^{k-1}n - d_\epsilon(1 - d_\epsilon)^{k-1}n = (1 - d_\epsilon)^k n$ items whp. By the union bound, it follows that *every* processor has not accessed at most $(1 - d_\epsilon)^k n$ items with probability at least

$$1 - 2n \sum_{i=0}^{k-1} \exp(-0.5\epsilon^2 d_1^2(1 - d_\epsilon)^i n) \geq 1 - 2nK \exp(-0.5\epsilon^2 d_1^2 n^{1/3}).$$

Similarly, after the k -th round, at most $(1 - d_\epsilon)^k n$ processors have not accessed *every* item. Therefore, the minimum number of items accessed by *every* processor during the first K rounds is at least

$$\sum_{k=1}^K (1 - d_\epsilon)^{k-1} d_\epsilon n = (1 - (1 - d_\epsilon)^K) n$$

with probability at least $1 - 2nK e^{-0.5\epsilon^2 d_1^2 n^{1/3}}$. The total number of steps taken during the first K rounds is

$$c_1 n (1 + (1 - d_\epsilon) + (1 - d_\epsilon)^2 + \dots + (1 - d_\epsilon)^{K-1}) \leq \frac{c_1}{d_\epsilon} n = \frac{c_1}{(1 - \epsilon)(1 - e^{-c_1/e})} n = O(n)$$

by choosing $c_1 > 0$ and $0 < \epsilon < 1$ to be constants (independent of n). After K -th round, every processor has at most $(1 - d_\epsilon)^K n = n^{1/3}$ items left to be accessed and thus at most $n^{2/3}$ additional steps are needed (in the worst case — cf. Theorem 2.1) until no items are left. Thus we conclude that $M' \leq O(n) + O(n^{2/3}) = O(n)$ with probability at least $1 - e^{-\delta n}$ for some $\delta > 0$. ■

3 Load Bounds and Analysis

Let $S(t)$ be the maximum load at step t among *all* processors, and $S = \max\{S(t), t > 0\}$ be the *maximum load* of any processor during the course of Protocol 1. We assume that each processor wants to access all the n items.

3.1 Upper Bound

For the upper bound we will need the following variant of the Chernoff bound from [32], which works in the case of dependent indicator random variables that are correlated as defined below.

Lemma 3.1 ([32]) *Let $Z_1, Z_2, \dots, Z_s \in \{0, 1\}$ be random variables such that for all l , and for any $S_{l-1} \subseteq \{1, \dots, l-1\}$, $\Pr(Z_l = 1 | \bigwedge_{j \in S_{l-1}} Z_j = 1) \leq \Pr(Z_l = 1)$. Then for any $\delta > 0$, $\Pr(\sum_{l=1}^s Z_l \geq \mu(1 + \delta)) \leq (\frac{e^\delta}{(1+\delta)^{1+\delta}})^\mu$, where $\mu = \sum_{l=1}^s \mathbf{E}[Z_l]$.*

Theorem 3.1

$$S \leq O\left(\frac{\log n}{\log \log n}\right) \text{ whp.}$$

Proof: Let the random variable $X_i(t)$ be the load of processor i at step t . It is easy to see that $E[X_i(t)] = 1$, for any $t > 0$ and for any processor i . This is because, for any step t , $\sum_{i=1}^n X_i(t) = n$, and by symmetry ($X_i(t)$ s are identically distributed), $E[X_1(t)] = E[X_2(t)] = \dots = E[X_n(t)]$.

We first bound $S(t)$, for any $t > 0$. We focus on a particular processor, say processor 1. Fix $t > 0$ and define random variables Z_1, Z_2, \dots, Z_t by

$$Z_j = \begin{cases} 1, & \text{if processor 1 accesses an item at step } j \text{ and the item remains at step } t \\ & \text{i.e., the item remains in processor 1 from step } j \text{ till step } t. \\ 0, & \text{otherwise.} \end{cases}$$

We note that the load of processor 1 at time t , $X_1(t) = \sum_{j=1}^t Z_j$.

First we show that for any $l \leq t$,

$$\Pr(Z_l = 1 | \bigwedge_{j \in S_{l-1}} Z_j = 1) \leq \Pr(Z_l = 1), \tag{8}$$

where S_{l-1} is an arbitrary subset of steps $\{1, 2, \dots, l-1\}$. Let $C = (\bigwedge_{j \in S_{l-1}} Z_j = 1)$. Since,

$$\Pr(Z_l = 1) = \Pr(Z_l = 1 | C)\Pr(C) + \Pr(Z_l = 1 | \bar{C})\Pr(\bar{C}),$$

where \bar{C} is the complement of C , if we prove that

$$\Pr(Z_l = 1 | C) \leq \Pr(Z_l = 1 | \bar{C}) \tag{9}$$

then (8) follows. Let $x_j, j \in S_{l-1}$ be the item that processor 1 accesses at step j . Conditioning on C , any other processor (except processor 1) cannot request item x_j at step l . Otherwise processor

1 would lose x_j which makes $Z_j = 0$. That is, under the condition C , other processors request one of $n - |S_{l-1}|$ items at step l . On the other hand, conditioning on \bar{C} , $Z_j = 0$ for some $j \in S_{l-1}$. Thus the item in step j was lost to some other processor, either in step j or in later steps, including step l . Hence, other processors can request one of (at least) $n - |S_{l-1}| + 1$ items at step l , which implies (9).

Let $\delta = c \frac{\log n}{\log \log n} - 1$, for some constant $c > 0$. Since $X_1(t) = \sum_{j=1}^t Z_j$, we have by Lemma 3.1:

$$\Pr(X_1(t) \geq 1 + \delta) = \Pr(X_1(t) \geq \mathbf{E}[X_1(t)](1 + \delta)) \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right).$$

To bound $S(t) = \max_{1 \leq i \leq n} X_i(t)$, i.e., the maximum load among all processors at step t , we use union bound ([27][Lemma 1.2]):

$$\Pr(S(t) \geq 1 + \delta) \leq n \Pr(X(t) \geq 1 + \delta) \leq \exp(\delta - (1 + \delta) \log(1 + \delta) + \log n) \leq 1/n^3 \quad (10)$$

if the constant c is chosen sufficiently large.

To bound $S = \max\{S(t), t > 0\}$ we again use union bound. Since the protocol will terminate in at most n^2 steps (cf. Theorem 2.1), using Equation 10, we have

$$\Pr(S > c \log n / \log \log n) \leq n^2 \Pr(S(t) > c \log n / \log \log n) \leq 1/n.$$

■

3.2 Lower Bound

We show that the load in some processor will be at least $\Omega(\sqrt{\log n})$ whp in *every* step $t \geq \sqrt{\log n}$.

In the proof of the following theorem we will need the powerful FKG inequality (cf. [37]). The FKG inequality can be summarized as follows for our purpose. Let Z_1, Z_2, \dots, Z_l be independent random variables each taking values in $\{0, 1\}$. Define $\vec{Z} := (Z_1, Z_2, \dots, Z_l)$ and all events are assumed to be completely determined by the value of \vec{Z} . Given $\vec{a} := (a_1, a_2, \dots, a_l) \in \{0, 1\}^l$ and $\vec{b} = (b_1, b_2, \dots, b_l) \in \{0, 1\}^l$, let us say that $\vec{a} \preceq \vec{b}$ iff $a_i \leq b_i$, for all i . Define an event A to be *increasing* iff: for all $\vec{a} \in \{0, 1\}^l$ such that if A holds when $\vec{Z} = \vec{a}$, then A also holds when $\vec{Z} = \vec{b}$, for any $\vec{a} \preceq \vec{b}$. Analogously, event A is said to be *decreasing* iff: for all $\vec{a} \in \{0, 1\}^l$ such that if A holds when $\vec{Z} = \vec{a}$, then A also holds when $\vec{Z} = \vec{b}$, for any $\vec{b} \preceq \vec{a}$. The FKG inequality states that if A and B are both increasing or decreasing events then $\Pr(A|B) \geq \Pr(A)$ and if one is increasing and the other is decreasing, then $\Pr(A|B) \leq \Pr(B)$.

Theorem 3.2 *Let $S(t)$ and S be defined as earlier. Then $S \geq \Theta(\sqrt{\log n})$ whp. Furthermore, $S(t) = \Omega(\sqrt{\log n})$ for all $t \geq \sqrt{\log n}$.*

Proof: We focus on processor 1 and show that during the first $n/2$ steps of the protocol, the maximum load of processor 1 is at least $\Theta(\sqrt{\log n})$ whp.

Consider step t , where $1 \leq t \leq n/2 - k$, where $k = o(\log n)$ is a parameter that will be fixed later. The probability that processor 1 accesses a new item in step t is at least $(1 - 2/n)^{n-1}$. This

is because any other processor has a probability of at most $2/n$ of accessing the same item as chosen by processor 1 (the other processors have to choose this item from among at least $n/2$ other items that have not been accessed yet).

In fact, the probability that processor will access a new item in step t and keep it for the next k steps (i.e., will not lose the item to some other processor) is at least

$$(1 - 2/n)^{(n-1)k} \geq (1 - 2/n)^{nk} \geq (e^{-2}(1 - 4/n))^k \geq e^{-2k}(1 - O(k/n)).$$

Processor 1 will have a load of size k if in some interval of k steps it accesses a new item in each of the k steps and keeps it in the interval. This probability is at least

$$e^{-2k} \times e^{-2(k-1)} \times \dots \times e^{-2} \times (1 - O(k/n))^k = e^{-(k)(k+1)}(1 - O(k^2/n)). \quad (11)$$

Partitioning the interval into $\lfloor n/2k \rfloor$ sets, each set consisting of k consecutive steps starting from step 1, we upper bound the probability that processor 1 *does not* have a load of size k in every one of these sets:

$$\left(1 - e^{-(k)(k+1)(1-O(k^2/n))}\right)^{\lfloor n/2k \rfloor} \leq e^{-e^{-(k)(k+1)(1-O(k^2/n))} \lfloor n/2k \rfloor} = o(1/n) \quad (12)$$

if $k = c\sqrt{\log n}$ for some sufficiently small constant $c > 0$.

Thus, $X_1(t) = \Omega(\sqrt{\log n})$ whp in at least one step. Hence $S \geq \Omega(\sqrt{\log n})$ whp.

We now show that the load of some processor will be at least $\Omega(\sqrt{\log n})$ in every step $t \geq \sqrt{\log n}$. Consider a time step $t \geq \sqrt{\log n}$. As calculated in Equation 11, by considering the interval $[t-k, t]$, processor 1 will have a load of at least k at time t with probability at least $e^{-(k)(k+1)}(1 - O(k^2/n))$. Hence,

$$\Pr(X_1(t) \geq k = \sqrt{\log n}) = \Theta(1/n^\epsilon) \quad (13)$$

for some constant $0 < \epsilon < 1$. Since the the loads of the different processors are not independent we cannot directly conclude the result. We use the FKG inequality to show that the loads of the different processors are (negatively) correlated as stated in the following Lemma.

Lemma 3.2 *$X(t)$ and $Y(t)$ be the loads of processor 1 and 2 at step t . Then for any $h > 0$*

$$\Pr(X(t) \geq h, Y(t) \geq h) \leq \Pr(X(t) \geq h) \cdot \Pr(Y(t) \geq h).$$

Proof: Suppose that processor k ($k = 1, 2, \dots$) has requested an item $z_{k,j}$ at each step $j = 1, 2, \dots, t$. Let $U := (X(t) \geq h)$ and $V := (Y(t) \geq h)$.

Define random variables $Z_{k,j}$, $k = 1, 2, \dots, n$, $j = 1, 2, \dots, t$ to be 1 if processor k accesses $z_{k,j}$ at step j and the item has not been moved to other processors before step t (for $t > j$), and 0 otherwise. To simplify the notation, let $x_j = z_{1,j}$ and $y_j = z_{2,j}$. That is $[x_1, x_2, \dots, x_t]$ and $[y_1, y_2, \dots, y_t]$ are the requested items through step 1 to t by processor 1 and processor 2 respectively. Let $X_j = Z_{1,j}$ and $Y_j = Z_{2,j}$.

For fixed $c_{k,j} \in \{0, 1\}$, $k = 3, 4, \dots, n$, $j = 1, 2, \dots, t$, define the event $W := (Z_{k,j} = c_{k,j}, k = 3, 4, \dots, n, j = 1, 2, \dots, t)$. Under condition W , Y_j is completely determined by $\vec{X} := (X_1, X_2, \dots, X_t)$. That is

$$Y_j = f_j(X_1, X_2, \dots, X_t)$$

for some function $f_j : D \subset \{0, 1\}^s \rightarrow \{0, 1\}$, where D is the set of all possible values of \vec{X} (under the condition W).

Claim: $f_j(X_1, X_2, \dots, X_t)$ is decreasing. That is, if $\vec{b} \preceq \vec{a}$, then $f(\vec{b}) \geq f(\vec{a})$.

It suffices to show that if $f_j(\vec{a}) = 1$ then $f_j(\vec{b}) = 1$ for $\vec{b} \preceq \vec{a} \in D$. Observe that the item y_j has not been moved to other processors since we are assuming that $Y_j = f_j(\vec{a}) = 1$ and hence $y_j \neq x_i$ for all i such that $a_i = 1$. Thus for any $\vec{X} = \vec{b} \preceq \vec{a}$, y_j remains in processor 2, since $y_j \neq x_i$ for such i satisfying $b_i = 1$.

The claim implies that $U = (X_1 + X_2 + \dots + X_t \geq h)$ is an increasing event while $V = (Y_1 + Y_2 + \dots + Y_t \geq h)$ is decreasing in terms of X_1, X_2, \dots, X_t . Thus by FKG inequality, we have $\Pr((U | V) | W) \leq \Pr(U | W)$ and hence $\Pr(U | V) = \sum_W \Pr((U | V) | W) \Pr(W) \leq \sum_W \Pr(U | W) \Pr(W) = \Pr(U)$. ■

Now we are ready to present the lower bound result.

Let $X_i(t)$ be the storage space of processor i at step t . Define event $A_i := (X_i(t) \geq h)$. Let $h = \sqrt{\log n}$.

Using the Chung and Erdos' second moment method (cf. [39])

$$\Pr\left(\bigcup_{i=1}^n A_i\right) \geq \frac{(\sum_{i=1}^n \Pr(A_i))^2}{\sum_{i=1}^n \Pr(A_i) + \sum_{i \neq j} \Pr(A_i \cap A_j)},$$

we observe that

$$\Pr(S(t) \geq h) \geq \frac{T_1^2}{T_1 + T_2},$$

where $T_1 = \sum_{i=1}^n \Pr(X_i(t) \geq h)$ and $T_2 = \sum_{i \neq j} \Pr(X_i(t) \geq h, X_j(t) \geq h)$. $T_1 = n \Pr(X_1(t) \geq h)$ and $T_2 \leq n(n-1) \Pr(X_1(t) \geq h)^2$ (by Lemma 3.2). By Equation 13, $\Pr(X_i(t) \geq h) = \Theta(1/n^\epsilon)$ for some constant $0 < \epsilon < 1$. Thus, we have $\Pr(S(t) \geq h) \geq \Theta\left(\frac{1}{1/n^{1-\epsilon}+1}\right) = 1 - \Theta\left(\frac{1}{n^{1-\epsilon}}\right)$. ■

4 Conclusion and Open Problems

We studied simple randomized protocols for conflict-free distributed access. Our results show that randomization with “memory” (i.e., sampling without replacement — Protocol 1) is asymptotically optimal with respect to the best possible time bound and only at most logarithmically larger than the best possible load bound. Below we list some extensions and open problems for future work.

1. There are gaps in the time and load bounds of Protocol 1. We conjecture that a tight concentration result for time can be shown, i.e., Protocol 1 can be shown to run in time $cn \pm o(n)$ with high probability, for some fixed constant c . Is the lower bound for load also $\Theta(\log n / \log \log n)$?
2. Consider the following two-choice paradigm [4] to achieve more even load distribution in our model: in every step, a processor requests two items and accesses the item from that processor having a *larger* number of items (cf. Section 1.2). We ignore the overhead involved in accessing the items and all other assumptions of the model stay the same. What are the load bounds for this protocol?

3. Suppose we allow only $O(1)$ maximum load for each processor. That is, a processor cannot request additional items if its load exceeds a constant. Then what are the time bounds for Protocol 1?
4. Our model assumes exclusive access, i.e., an item can be accessed by only one processor in one time step. We observe that this assumption can be relaxed. For example, one can specify an upper bound (say k) on the number of processors that can simultaneously access an item. Thus, up to k processors can simultaneously access an item in the same time step. We can then study the same randomized protocols under this modified condition. We leave it to the reader to extend the time and load bounds for this generalized setting.
5. Our model makes the simplifying assumption that an item can be accessed in one time step, no matter where it is located in the network. However, to be applicable to practical distributed networks, the model should be extended to take into account the latency in finding and accessing an item. In this context, a natural extension of our work is to study arbitrary networks where *routing delay* and *congestion* in intermediate nodes have to be taken into account. In particular, it will be interesting to study the problem first on special types of network topologies such as hypercube and grid networks.

Acknowledgments. We are very grateful to the referees for their careful reading of the paper and detailed comments which helped greatly in improving the presentation of the paper. We also thank the anonymous referees of ACM PODC 2005, Eli Upfal, and Wojciech Szpankowski for their comments.

References

- [1] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel Randomized Load Balancing, *Proc. of 27th ACM Symposium on Theory of Computing (STOC)*, 1995.
- [2] N. Alon and J. Spencer. *The Probabilistic Method*, Second Edition, John Wiley, 2000.
- [3] D. Angluin and L. Valiant. Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings, *Journal of Computer and System Sciences*, **18**, 1979, 155-193.
- [4] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced Allocations, *SIAM J. on Computing*, **29**, 2000, 180-200.
- [5] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time Complexity of Broadcast in Radio Networks: An Exponential Gap Between Determinism and Randomization, *Journal of Computer and System Sciences*, **45**, 1992, 104-126.
- [6] B. Chlebus. Randomized Communication in Radio Networks, a chapter in *Handbook of Randomized Computing*, P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolin (Eds.), Kluwer Academic Publishers, 2001, vol I, 401-456.

- [7] B. Chlebus, D. Kowalski. Randomization Helps to Perform Independent Tasks Reliably. *Random Struct. Algorithms*, **24**(1), 2004, 11-41.
- [8] A. Czumaj and W. Rytter. Broadcasting Algorithms in radio networks with unknown topology
- [9] M. Dilman and D. Raz. Efficient Reactive Monitoring, *IEEE Journal on Selected Areas in Communications*, **20**, 2002, 668-676.
- [10] D. Dubhashi and D. Ranjan. Balls and Bins: A Study in Negative Dependence, *Random Structures and Algorithms*, **13**(2), 99-124, 1998.
- [11] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Randomized Broadcast in Networks, *Random Structures & Algorithms*, **1**, 1990, 447-460.
- [12] R. Gallager. A Perspective on Multiaccess Channels, *IEEE Transactions on Information Theory*, **31**(2), 1985, 124-142.
- [13] L. Goldberg, P. Mackenzie, M. Paterson, and A. Srinivasan. Contention resolution with constant expected delay, *JACM*, **47**(6), 2000, 1048-1096.
- [14] L. Goldberg, M. Jerrum, S. Kannan, and M. Paterson. A Bound on the Capacity of Backoff and Acknowledgment-based Protocols, *SIAM J. Comput.*, **33**(2), 2004, 313-331.
- [15] J. Hastad, F. Leighton, and B. Rogoff. Analysis of Backoff Protocols for Multiple Access Channels, *SIAM J. on Computing*, **25**(4), 1996, 740-774.
- [16] N.L. Johnson and S. Kotz. *Urn Models and their Applications*, John Wiley, 1977.
- [17] S. Jagannathan, G. Pandurangan, and S. Srinivasan. Query Protocols for Highly Resilient Peer-to-Peer Networks, in *Proc. of the 19th International Conference on Parallel and Distributed Computing Systems*, 2006.
- [18] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized Rumor Spreading, *IEEE Symposium on Foundations of Computer Science*, 2000.
- [19] D. Kempe and J. Kleinberg. Protocols and Impossibility Results for Gossip-Based Communication Mechanisms, *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
- [20] D. Kempe, J. Kleinberg, and A. Demers. Spatial Gossip and Resource Location Protocols, *Proc. of ACM Symposium on Theory of Computing (STOC)*, 2001.
- [21] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information, *IEEE FOCS*, 2003.
- [22] D. Kempe and F. McSherry. A Decentralized Algorithm for Spectral Analysis, *In Proceedings of STOC*, 2004.

- [23] V. Kolchin, V. Chistiakov, and B. Sevastianov. *Random Allocations*, V.H. Winston, New York, 1978.
- [24] D. Kowalski. On Selection Problem in Radio Networks, *Proc. of ACM PODC*, 2005, 158-166.
- [25] Z. Lotker, B. Patt-Shamir, E. Pavlov, and D. Peleg. Minimum-Weight Spanning Tree Construction in $O(\log \log n)$ Communication Rounds, *SIAM J. Comput.*, **35**(1), 2005, 120-131.
- [26] M. Mitzenmacher, A. Richa, and R. Sitaraman. The Power of Two Random Choices: A survey of the Techniques and Results, in *Handbook of Randomized Computing, Vol. I*, Edited by P. Pardalos, S. Rajasekaran, J.Reif, and J. Rolim, 255-312, Kluwer Press, 2001.
- [27] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, 2005.
- [28] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.
- [29] S. Nikolettseas and P. Spirakis. Randomized Techniques for Modeling Faults and Achieving Robust Computing, a chapter in *Handbook of Randomized Computing*, P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolin (Eds.), Kluwer Academic Publishers, 2001, vol I, 313-399.
- [30] A. Oram (Editor). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly, 2001.
- [31] M. Ozsdu and P. Valduriez. *Principles of Distributed Database Systems*, 2nd Edition, Prentice Hall, 1999.
- [32] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds, *SIAM Journal on Computing*, **26**, 1997, 350-368.
- [33] D. Peleg. *Distributed Computing: A Locality Sensitive Approach*, SIAM, 2000.
- [34] M. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance, *Journal of the ACM*, **36**(2), 1989, 335-348.
- [35] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network in *Proceedings of ACM SIGCOMM*, 2001.
- [36] P. Raghavan and E. Upfal. Stochastic Contention Resolution with Short Delays, *SIAM Journal of Computing*, **28**(2), 1998, 709-719.
- [37] A. Srinivasan. Approximation Algorithms via Randomized Rounding: a Survey. *Lectures on Approximation and Randomized Algorithms (M. Karonski and H. J. Promel, editors)*, Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN, Warszawa, 1999, 9-71.
- [38] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, in *Proceedings of ACM SIGCOMM*, 2001.
- [39] W. Szpankowski. *Average Case Analysis of Algorithms on Sequences*, Wiley, New York, 2001.