**Distributed Computing manuscript No.**
(will be inserted by the editor)

# Randomized Leader Election

**Murali Krishna Ramanathan**
**Ronaldo A. Ferreira**
**Suresh Jagannathan**
**Ananth Grama**
**Wojciech Szpankowski**

**Department of Computer Sciences**
**Purdue University, IN 47907**

**{rmk, rf, suresh, ayg, spa}@cs.purdue.edu**

**Summary.** We present an efficient randomized algorithm for leader election in large-scale distributed systems. The proposed algorithm is optimal in message complexity ($O(n)$ for a set of $n$ total processes), has round complexity logarithmic in the number of processes in the system, and provides high probabilistic guarantees on the election of a unique leader. The algorithm relies on a *balls and bins* abstraction and works in two phases. The main novelty of the work is in the first phase where the number of contending processes is reduced in a controlled manner. Probabilistic quorums are used to determine a winner in the second phase. We discuss, in detail, the synchronous version of the algorithm, provide extensions to an asynchronous version and examine the impact of failures.

## 1 Introduction

The problem of leader election in distributed systems is an important and well-studied one. This problem is at the core of a number of applications – it forms the basis for replication (or duplicate elimination) in unreliable systems, establishing group communication primitives by facilitating and maintaining group memberships, and load balancing and job-scheduling in master-slave environments [31].

A leader election algorithm is formally characterized as follows [19,32]: *given a distributed ensemble of processes with each process executing the same local algorithm, the algorithm is decentralized, i.e., a computation can be initiated by an arbitrary non-empty subset of processes, and the algorithm reaches a terminal configuration in each computation, and in each reachable terminal configuration, in the subset of processes that initiated the computation, there is exactly one process in the state* **leader** *and all other processes in the subset are in the state* **lost**.

The central challenge of efficient, scalable, and robust leader election algorithms is to simultaneously minimize the number of messages and overall execution time. With these objectives, a number of leader election protocols have been proposed [4,8,9,12,28]. The emergence of novel distributed paradigms and underlying platforms such as peer-to-peer systems [11,26,27,29] for resource sharing pose interesting new variants of this problem *viz.,* scalability, reduced restriction on correctness, etc. This motivates the design of a variety of probabilistic leader election schemes [12,28]. In [12], Gupta *et al.* use a multicast approach to elect a leader in a group with high constant probability. In [28], Schooler *et al.* propose two variants of the leader election algorithm and analyze it in the context of a multicast with respect to several metrics, including delay and message overhead.

In this paper, we present a randomized leader election algorithm (primarily non fault-tolerant) that is optimal in terms of message complexity ($O(n)$ for a distributed system with $n$ processes), has round complexity logarithmic in $n$, and is correct with high probability (w.h.p.). In other words, as $n \to \infty$, the probability of one leader getting elected tends to one. Throughout this paper, w.h.p. [1] (with high probability) denotes probability $1 - \frac{1}{\log^{\Omega(1)} n}$. This is in contrast to known algorithms (Section 8) that involve a large number of messages [18,20,21] or require a larger number of rounds [4]. In this sense, our algorithm is targeted towards large-scale distributed systems [11,26,27,29], in which scalability is an important issue. However, it is important to

[1] Our definition of the term "with high probability" is weaker than the customary one, which typically requires a high probability event to occur with probability at least $1 - \frac{1}{n^\alpha}$, for some $\alpha > 0$, where $n$ is a "system size" parameter. Under both definitions the probability of the event converges to 1 as the system size grows to infinity, but under our use of the term this convergence is exponentially slower than under the customary use.

| Algorithm | Messages | Rounds/Time | Correctness |
|---|---|---|---|
| Probabilistic Quorum [21] | $O(n\sqrt{n})$ | 1 round | with high probability |
| P.C.L.E [12] | $O(K^4 * n)$ per round | unspecified | depends on system parameters |
| LE in multihop broadcast environment [4] | $O(n \log n)$ | $O(n)$ time | guaranteed |
| LE in a synchronous ring [9] | $\Omega(n \log n)$ | Bounded rounds (t) | guaranteed |
| Highly available LE Service [8] | $O(n-1)$ datagrams per round | unspecified | dependent on connectivity |
| LE for multicast groups [28] | dependent on failures | dependent on failures | guaranteed (in the absence of failures) |
| Randomized Leader Election | $O(n)$ | $O(\log n)$ rounds | with high probability |

**Table 1.** Comparison of election algorithms with respect to message complexity, round complexity and correctness.

note that while many traditional leader election algorithms provide absolute guarantees for the election of a unique leader, our algorithm guarantees leader election with high probability. We provide a comparison of leader election algorithms for different metrics in Table 1.

### *1.1 Technical Contributions*

The main contributions of the paper are as follows:

1. A randomized leader election algorithm that is optimal in the number of messages $O(n)$, has round complexity logarithmic in the number of processes in the system $O(\log n)$, and elects a unique leader w.h.p.
2. An approach in which the lack of global information is intelligently leveraged to prune the number of processes participating in the leader election algorithm.
3. An asynchronous version of the first phase of the algorithm and a partially synchronous [5] version of the second phase so that the algorithm can be effectively realized for general distributed applications.
4. A rigorous analysis to prove the correctness and the complexity of the algorithm.

The rest of the paper is organized as follows: Section 2 formalizes definitions and terminology used in this paper, Section 3 presents the synchronous version of the protocol along with proofs relating to message complexity and probabilistic bounds on election of a unique leader, and Section 4 presents an asynchronous version of the protocol with analytical performance bounds. The impact of failures is addressed in Section 5. Issues relating to current distributed systems are discussed in Section 6. We provide a proof of concept of our approach using simulations in Section 7. Related work is presented in Section 8. Conclusions and avenues for future research are outlined in Section 9.

## 2 System Definitions and Model

We start by formalizing definitions and terminology used in the rest of the paper.

- **Process:** A *process* is an individual entity in a distributed system that can participate in the leader election protocol. It can communicate with any other process by sending and receiving messages. It is capable of generating random numbers independent of other processes in the system. In a synchronous system, all processes share a common clock (or, equivalently, their local clocks are synchronized), and message transfers are assumed to take unit time. In an asynchronous system, a process maintains a local clock, which is not necessarily synchronized with other processes. Furthermore, message transfers may take arbitrary time. All processes are assumed to follow the specified algorithm.
- **Contender:** The leader election algorithm presented in this paper is fashioned as a game played by participating processes. The set of processes playing this game decreases as the algorithm proceeds. This set of processes, from which a winner is selected, is referred to as the set of *contenders*.
- **Mediator:** Winners at intermediate steps in the game are decided from among the set of contenders by processes referred to as *mediators*. Specifically, a mediator is a process that receives a message from a contender and arbitrates whether the contender participates in subsequent steps of the protocol.
- **Round:** A *round* is composed of a two-way exchange between a single process and a set of mediators. At the end of a round, if the process is still a contender, a new round (communication with a new set of mediators) is initiated.
- **Winner:** A *winner* is a process that has not received negative responses from any mediator throughout the entire execution of the protocol.

In this paper, we primarily discuss leader election algorithm in a distributed system without process or communication failures. We initially provide the methodology and proof for the correctness and complexity for the synchronous version of the algorithm. The complexity measures considered are:

1. **Message Complexity:** The total number of messages exchanged by all processes (contenders and mediators) in the system across all rounds for one complete execution of the leader election algorithm.
2. **Round Complexity:** The total number of rounds taken by the process elected as 'leader'.

We provide an asynchronous version of the algorithm. We assume partial asynchrony based on the models developed by Dolev *et al* [5], where we assume a maximum time bound for message delay and processor speed i.e., within a specific time ($\tau$) known to all processes, a message sent by any process `A` to any process `B` is received and processed by `B` in at most $\tau$ time units. Subsequently, we consider the implications of fail-stop process failures, in which a failed process neither sends nor receives messages, on our algorithm. We assume that each process fails with some probability $\gamma$.

## 3 A Synchronous Leader Election Protocol

In this section, we present a randomized synchronous algorithm for leader election that is scalable with respect to system size and offers high probabilistic guarantees for correctness. By high probabilistic guarantee, we mean that the probability of two processes getting elected as leader tends to zero with increase in the system size. We initially present the synchronous version of our algorithm for ease of understanding. In Section 4, we extend our algorithm to the asynchronous setting, which is more realistic for large-scale systems.

We first present the algorithm informally using a balls-and-bins abstraction and subsequently formalize it in the context of distributed systems. The algorithm is played as a tournament in two phases. The first phase consists of multiple rounds. In round $i$ of this phase, each contender casts $\sigma_i$ balls into $n$ bins. (We derive precise expressions for $\sigma_i$ later in this section.) A contender is said to 'win' a bin if its ball is the only one that lands in the bin. If a contender wins all the bins that its balls land in, it is considered a winner in this round and proceeds to the next round. An example of this process is illustrated in Figure 1 for $n = 8$. The first phase consists of $\log_2 8 - 1 = 2$ rounds. In the first round, processes 2,4,5 and 6 proceed, and in the second round, processes 5 and 6 proceed.

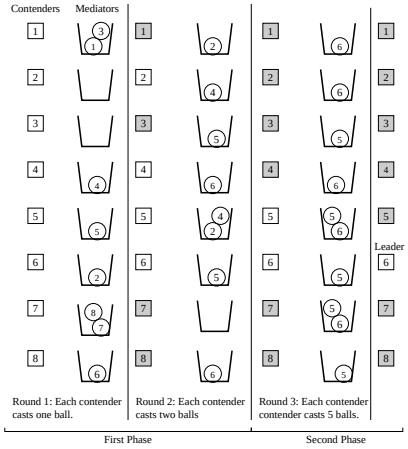


**Fig. 1.** Illustration of the synchronous protocol. Contenders are illustrated by squares, mediators by bins, and messages from contenders to mediators by labeled balls (the labels indicate the source of the ball). In all rounds, contenders that are no longer in the running are shaded. In the first round, each contender casts one ball and contenders 2, 4, 5, and 6 proceed since their balls uniquely occupy their respective bins. In round 2 (last round of the first phase), each contender casts two balls and contenders 5 and 6 proceed. Finally, in round 3 (the only round of the second phase), each contender casts 5 balls and contender 6 is selected the leader.

In a realization of this balls-and-bins abstraction, a process can be a contender as well as a mediator (a process to which one or more contenders sends a message to arbitrate) at the same time. Casting a ball into a randomly chosen bin corresponds to a message from a contender to a randomly chosen process (who thereby becomes a mediator) picked uniformly at random. A contender is a winner if none of the mediators it sends messages to receive messages from any other contenders. The number of mediators to which a contender sends messages in round $i$ is denoted $\sigma_i$. This quantity depends on the number of processes and the current round number. We discuss how to choose the value of $\sigma_i$ in Section 3.1. This number, if carefully selected, can reduce the number of contenders by half, on average, after every round. This results with high probability in a small, nonzero, number of contenders being left after a number of rounds that is logarithmic in the number of processes. The number of contenders is not relevant since each contender executes the algorithm assuming that every process is a contender and the number of rounds is still logarithmic in the total number of processes and the message complexity is preserved.

The second phase of the protocol consists of a single round and is based on probabilistic quorums [21]. Each remaining contender generates a random number in a specific range, such that the random number generated is unique w.h.p among all the random numbers generated by other contenders, and sends this number to a set of mediators picked uniformly at random. A contender wins this phase w.h.p. if it generates the largest random number among all the contenders. The number of mediators in this phase is chosen so that there is a high probability of at least one overlapping mediator between two contenders. This overlapping mediator arbitrates based on who generates the larger random number. A crucial difference between the two phases is as follows: In the first phase a contender wins a round only if it sends a message to an exclusive set of mediators (the set of processes to which no other contender has sent a message for that round). However, in the second phase, the winner is decided based on the random number generated and the intersection of the selected sets of mediators. The reason for this difference is that in the first phase, the number of contenders is reduced just enough so that only a few of them proceed to the next round, maintaining the message complexity. In the second phase, the objective is to have one final winner and the message complexity is maintained due to a smaller set of contenders.

### *3.1 The Synchronous Leader Election Protocol*

We consider a distributed system of $n$ processes represented by the set $\Gamma_n = \{a_i \mid 1 \leq i \leq n\}$, for processes $a_1, a_2, \ldots, a_n$. The set of contenders in round $j$ of the protocol is represented by $\Phi_j = \{\rho_i \mid 1 \leq i \leq |\Phi_j|\}$, where $\Phi_j \subseteq \Gamma_n$, $\Phi_{j+1} \subseteq \Phi_j$ and $\rho_1, \rho_2, ..., \rho_{|\Phi_j|}$ are the contenders. We define $E[X_j]$ to be the expected number of contenders in round $j$, since the exact number of contenders $|\Phi_j|$ cannot be calculated in a distributed setting. We later show in our analysis that $E[X_j] \approx \frac{n}{2^{j-1}}$. A random integer $\pi_i$ is selected in the range $[0..n^4]$ by each contender $\rho_i$. This range guarantees that the chosen values are unique with high probability [19](chapter 4, page 72).

| Notation | Description |
|---|---|
| $\Gamma_n$ | Distributed system with $n$ processes. |
| $a_i$ | A process in the distributed system. |
| $\rho_i$ | A contender in a round. |
| $\Phi_j$ | Set of contenders in round $j$. |
| $\pi_i$ | Random number generated by contender $\rho_i$. |
| $w$ | Final round of the protocol. |
| $\sigma_j$ | Number of processes that will be mediators in round $j$ for a contender. |
| $\Psi_{ij}$ | Set of mediators for contender $\rho_i$ in round $j$. |
| $\kappa_{ij}$ | Set of messages received by mediator $a_i$ in round $j$. |
| $C_n$ | Maximum number of contenders in the final round. |
| $\gamma$ | Failure probability of any process. |

**Table 2.** System Parameters

The protocol concludes by declaring a unique member in $\Phi_w$ to be the final winner, where $w$ corresponds to the final round in the protocol, and $\{\forall j : j < w, |\Phi_j| > 1\}$. We take $w = \log n - \log\log^6 n + 1$, which is a fixed value known by all processes. We show in Section 3.2 that this suffices to satisfy the probabilistic guarantees of our protocol. Various system parameters are summarized in Table 2.

The synchronous algorithm is as follows, with each contending process $\rho_i$ in round $j$ performing the following steps:

- Set
$$\sigma_j = \begin{cases} \sqrt{\frac{n \ln 2}{E[X_j]-1}}, & \text{if } j < w \\ \sqrt{n \ln n}, & \text{otherwise} \end{cases}$$
where $\sigma_j$ is the number of mediators in round $j$, $E[X_j]$ is the expected number of contenders in round $j$ and $n$ is the total number of processes in the system. The value of $\sigma_j$ is selected in this manner to guarantee bounds on number of messages and rounds. A detailed analysis is provided in Section 3.2.
- Let $\Psi_{ij}$ be the set of $\sigma_j$ processes selected uniformly at random from $\Gamma_n$. If $j < w$, send $(\rho_i)$ to all processes in $\Psi_{ij}$. Otherwise, send the ordered pair $(\rho_i, \pi_i)$.
- Proceed to the next round $j + 1$ as a contender if and only if positive responses are received from all the processes in $\Psi_{ij}$. Otherwise, move election result to **lost**.
- If round number is $w + 1$, move election result to **leader**.

Each process $a_i$ in $\Gamma_n$ performs the following steps in round $j$:

- Receive messages from $\rho_k$'s in $\Phi_j$ and populate the set $\kappa_{ij}$ with ordered pairs $(\rho_k, \pi_k)$.
- **First Phase:** $j < w$
  - If $|\kappa_{ij}| = 1$, then send a positive response to $\rho_k$ in $\kappa_{ij}$ and proceed to round $j + 1$.
  - Otherwise, send a decline message to every $\rho_k$ present in $\kappa_{ij}$ and proceed to round $j + 1$.
- **Second Phase:** $j = w$
  - For every $(\rho_k, \pi_k)$ in $\kappa_{ij}$, find the largest[2] $\pi_k$ and send a positive response to the corresponding $\rho_k$. Send a decline message to the rest of the $\rho_k$'s.

[2] The probability that the random numbers generated by any two contenders is equal is extremely low. In the unlikely event that the two equal numbers are the largest among the numbers generated by all contenders, the algorithm fails in that two or no leaders are elected even in the presence of intersection of the mediator sets.

### *3.2 Analysis*

We quantify the overhead and the probability of electing a unique leader using our algorithm. On average, the number of contenders is halved after a single round in the first phase. Let $X_i$ be a random variable representing the number of contenders proceeding to round $i$ from round $i-1$, $E[X_i]$ the expectation of $X_i$, $f_n = \log n - \log\log^6 n$, $1 < i \leq f_n$, $\epsilon$ be $O\left(\frac{1}{\log^2 n}\right)$, $\ell_i = (1-\epsilon)E[X_i]$, $u_i = (1+\epsilon)E[X_i]$, $g(t) = 1 - O(\frac{1}{t})$, and $h(t) = 1 + O(\frac{1}{t})$.

**Lemma 1.**

1. $\frac{n}{2^{i-1}}[g(\log^2 n)]^{i-1} \leq E[X_i] \leq \frac{n}{2^{i-1}}[h(\log^2 n)]^{i-1}$
2. $P(\ell_i < X_i < u_i) \geq g(\log^6 n)$

*Sketch of proof:* We are given $X_1 = n$ and $\sigma_i = \sqrt{\frac{n \ln 2}{E[X_i]-1}}$. The analysis can be trivially extended to $X_1 < n$. The $X_i$'s are binomially distributed in any round $i$. It can easily be shown that $E[X_{i+1}|X_i = \ell] = \ell(1 - \frac{\sigma_i}{n})^{\sigma_i(\ell-1)}$. We illustrate this concept using a few variables and then generalize for $1 < i \leq f_n$.

$$\begin{aligned} E[X_2] &= \sum_x E[X_2|X_1 = x]P(X_1 = x) \\ &= E[X_2|X_1 = n] = \frac{n}{2}g(n) \end{aligned}$$

We can bound the probabilities of $X_2$ based on the Chernoff bound [30].

$$P(X_2 \geq u_2) \leq e^{-\frac{\epsilon^2 E[X_2]}{3}}$$
$$P(X_2 \leq \ell_2) \leq e^{-\frac{\epsilon^2 E[X_2]}{2}}$$
$$P(\ell_2 \leq X_2 \leq u_2) \geq g(n^2)$$

We calculate the bounds for $X_2$ since the value of $X_1$ is already known. However, for subsequent random variables, we give the bounds for $X_i$ considering the variability of $X_{i-1}$. Let $\chi_i$ represent the condition $\ell_i < x < u_i$, then

$$\begin{aligned} E[X_3] &= \sum_{\chi_2} E[X_3|X_2 = x]P(X_2 = x) \\ &\quad + \sum_{\overline{\chi_2}} E[X_3|X_2 = x]P(X_2 = x) \\ &= \sum_{\chi_2} E[X_3|X_2 = x]g(n^2) + (1 - g(n^2)) \end{aligned}$$

We use the following implication to bound $E[X_3]$. $z < x < y \Rightarrow ze^{-y} < xe^{-x} < ye^{-z}$.

$$\begin{aligned} E[X_3|X_2 = u_2] &\leq u_2\left(1 - \frac{\sigma_2}{n}\right)^{\sigma_2(\ell_2-1)} \\ &= \frac{n}{2^2}[h(\log^2 n)]^2 \end{aligned}$$

Similarly, we can prove that

$$E[X_3|X_2 = \ell_2] \geq \frac{n}{2^2}[g(\log^2 n)]^2$$

From the above results, we get

$$\frac{n}{2^2}[g(\log^2 n)]^2 \leq E[X_3] \leq \frac{n}{2^2}[h(\log^2 n)]^2$$

The above bounds can be extended in a similar fashion for any $i$, $1 < i \leq f_n$.

**Theorem 1.** *At the end of the first phase, there is at least one contender with probability* $1 - O(\frac{1}{\log^6 n})$*, i.e* $P(X_{f_n} = 0) \to 0$ *as* $n \to \infty$.

*Proof:* We use the second moment method, the results from Lemma 1 and the fact that $X_i$'s are binomially distributed.

$$\begin{aligned}\mathrm{Var}[X_i|X_{i-1} = u_{i-1}] &\leq u_i\left[1 - \left(1 - \frac{m_{i-1}}{n}\right)^{m_{i-1}(u_{i-1}-1)}\right] \\ &= \frac{1}{2}E[X_i][h(\log^2 n)]\end{aligned}$$

Similarly, we can prove that

$$\mathrm{Var}[X_i|X_{i-1} = \ell_{i-1}] \geq \frac{1}{2}E[X_i][g(\log^2 n)]$$

From the above results, we get (for $c < 1$)

$$cE[X_i][g(\log^2 n)] \leq \mathrm{Var}[X_i] \leq E[X_i][h(\log^2 n)]$$

From Lemma 1, we have

$$E[X_{f_n}] \geq \frac{n}{2^{f_n-1}}[g(\log^2 n)]^{f_n-1} = \frac{\log^6 n}{2}[g(\log n)]$$

Using the second moment method, we have

$$P(X_{f_n} = 0) \leq \frac{\mathrm{Var}[X_{f_n}]}{E[X_{f_n}]^2} = O\left(\frac{1}{\log^6 n}\right)$$

The theorem follows. □

**Theorem 2.** *At the end of the first phase, the number of contenders remaining is at most* $\frac{1}{2}(1+\epsilon)\log^6 n$ *with probability* $1-O(\frac{1}{\log^6 n})$*, i.e* $P(X_{f_n} > \frac{1}{2}(1+\epsilon)\log^6 n) \to 0$ *as* $n \to \infty$*, for some* $\epsilon < O(\frac{1}{\log^2 n})$.

*Proof:* From Lemma 1, the theorem follows. □

**Theorem 3.** *With high probability, there is exactly one contender remaining at the end of the protocol.*

*Proof:* $\forall u, v : u, v \in \Phi_w, u \neq v$, the probability that any two sets $\Psi_{uw}$ and $\Psi_{vw}$ intersect is

$$Pr(\Psi_{uw} \cap \Psi_{vw}) = 1 - \left(1 - \frac{\sqrt{n \ln n}}{n}\right)^{\sqrt{n \ln n}} \approx 1 - \frac{1}{n}$$

Furthermore, at the end of the first phase, $O(\log^6 n)$ contenders are present w.h.p. To elect a unique leader, the mediator set of the contender with the highest value needs to intersect with the mediator set of all other contenders. The corresponding probability of intersection is $\left(1 - \frac{1}{n}\right)^{O(\log^6 n)} = \left(1 - O\left(\frac{\log^6 n}{n}\right)\right)$. Since the mediator sets intersect w.h.p, only the process $\rho_i$ with the highest value of $\pi_i$ receives positive responses from all processes in $\Psi_{iw}$. The rest of the contending processes each have at least one decline message. However, from Theorem 1, the probability that at least one contender remains at the end of the first phase is 1 - $O(\frac{1}{\log^6 n})$. Therefore, the probability that a unique final winner is chosen at the end of the protocol is given by 1 - $O(\frac{1}{\log^6 n})$. □

**Theorem 4.** *The number of rounds in the protocol is* $O(\log n)$.

*Proof:* The number of rounds in the first phase is $\log n - \log\log^6 n$. There is a unique round in the second phase. The theorem follows. □

**Theorem 5.** *The total number of messages exchanged by all the processes across all rounds is* $O(n)$.

*Proof:* The total number of messages exchanged by all the processes across all rounds of the first phase is $2\left(\sum_{j=1}^{w-1} X_j \sqrt{\frac{n \ln 2}{E[X_j]-1}}\right)$. Based on Lemma 1, in the worst case, the number of messages in the first phase is $O(n)$ as $\sum_{j=1}^{w-1} \frac{1}{2^{\frac{j-1}{2}}}$ converges. Since each contender sends at most $\sqrt{n \ln n}$ messages in the second phase, the theorem follows from Theorem 2. □

An improvement of the algorithm is to send request messages to $\sigma_j - \sum_{l=1}^{j-1} \sigma_k$ mediators in a round $j$ of the protocol with the mediators from previous rounds participating in round $j$. This is in contrast to selecting a new set of $\sigma_j$ processes for every round $j$. This optimization does not change the asymptotic behavior of the algorithm, but reduces the number of messages by a constant factor.

## 4 An Asynchronous Leader Election Protocol

The design of the asynchronous protocol is largely motivated by its synchronous counterpart. However, asynchrony poses significant challenges since we would like to preserve the message and round complexities of the synchronous protocol. We present a suitably modified asynchronous protocol, prove that it is correct, and that it follows the bounds presented in Section 3. While the first phase is totally asynchronous, the second phase, which is based on probabilistic quorums [21], is partially synchronous [5], where we assume a maximum time bound for message delay and processor speed i.e., within a specific time ($\tau$) known to all processes, a message sent by any process `A` to any process `B` is received and processed by `B` in at most $\tau$ time units.

In the first phase of the synchronous protocol, when messages from two distinct contenders are sent to the

same mediator, both contenders receive a negative response. In the asynchronous case, messages from contenders to a mediator for a specific round need not be received at the same time. Therefore, in every round, a mediator responds positively to the first contender request for that round. A negative response is sent to subsequent requests from other contenders for that round. In the second phase of the partially synchronous version [5], messages from contenders need not be received at the same time, as compared to the synchronous version. A contender in the second phase of the protocol wins if it gets positive responses from the required number of mediators within a bounded time (the time bound is provided as a parameter to the algorithm). If a contender receives no response (either positive or negative) from a mediator, such scenario is attributed to the failure of a mediator. An analysis for this is presented in Section 5.

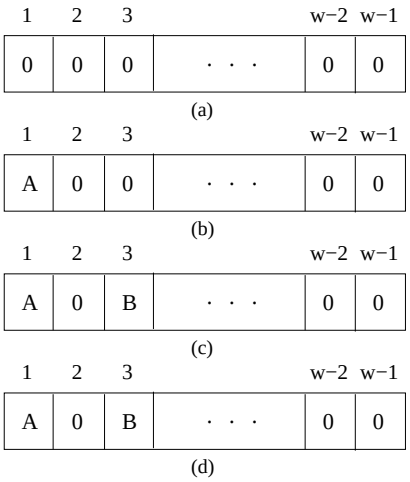


**Fig. 2.** (a) Vector V before receiving any requests. (b) V after a request from A for round number 1. (c) V after a request from B for round number 3. (d) V after a request from D for round number 3

### *4.1 The First Phase*

In the first phase of the asynchronous protocol, a contender $\rho_i$ sends requests, along with a round number $j$, to the mediators in the set $\Psi_{ij}$. The mediators in the set $\Psi_{ij}$ are picked uniformly at random as in the synchronous protocol. The number of mediators in any round of the asynchronous first phase is equal to the corresponding round in the synchronous phase. A contender $\rho_i$ proceeds to round $j+1$ if it receives positive responses from all the mediators in its set $\Psi_{ij}$. We describe below the procedure that the mediator adopts to send a positive response to a contender.

A mediator $M$ maintains a vector $V$ of size equal to the number of rounds ($\log_\alpha n$). All the entries in $V$ are initially set to zero. On receiving a request from a contender $\rho_i$ in round $j$, if entry $j$ in $V$ at $M$ is zero, $M$ sends a positive response to $\rho_i$ and sets the entry $j$ to $\rho_i$ (signifying that the winner of the $j$th round at $M$ is $\rho_i$). Otherwise a negative response is sent to $\rho_i$. The purpose of this step is to reduce the number of contenders that proceed to subsequent rounds. A better solution that reduces the number of contenders in the protocol and does not change the correctness of the protocol would require $M$ to send a negative response to any request with round number smaller than the index of the highest entry in $V$ with a non zero value $\rho_k$. This is because $M$ has knowledge that $\rho_k$ is in an advanced round (ahead of $\rho_i$) and hence is more likely to be elected the leader when compared to $\rho_i$. We use the first approach here, in which a mediator lets a contender proceed even when a higher numbered entry in its vector is already set. This simplifies our analysis considerably. We show that the asymptotic message complexity and number of rounds hold even under this conservative approach.

Figure 2 illustrates an example of the protocol executed at $M$. In (a), $M$ initially sets all the entries in $V$ to zero. $M$ receives a request for round one from contender $A$. Since the corresponding entry is zero, it sets the entry to $A$ and sends a positive response to $A$ (see (b)). Similarly, $M$ receives a request for round three from contender $B$. It sets entry three to $B$ and sends a positive response to $B$ (see (c)). However, when $M$ receives another request for round three from contender $C$, since the entry is already set, a negative response is sent to $C$ (see (d)). A mediator responds to requests for a round based on the corresponding entry in $V$, independent of other entries. The contenders that survive (which do not receive even a single negative response) all the rounds in the first phase proceed to the second phase of the protocol.

Let $A$ and $B$ be two contenders in the same round $i$ and let $C$, $D$ be two common mediators (in general, the probability of such an event is very low). Suppose $C$ receives requests from $A$ and $B$ in that order and $D$ receives the request in the reverse order. Both $A$ and $B$ cannot proceed to future rounds. This is similar to a collision in the synchronous case. However, if $C$ and $D$ both receive the requests from $A$ and $B$ in the same order, then one of the contender proceeds to the next round. We note that race conditions can occur in an asynchronous system. However, our probabilistic analysis takes into account these race conditions [3]. We show in our analysis that the increased number of contenders proceeding to later rounds as compared to the synchronous phase does not affect the message and time complexity.

### *4.2 Details of the second phase of the asynchronous algorithm*

Let $\tau$ represent the network delay and processing time associated with any message in the system. This is a parameter to the algorithm – messages in an asynchronous environment that exceed this time are treated as failures and their handling is addressed in Section 5. In the second phase of the protocol, a contender $\rho_i$ sends a request

[3] To provide an analogy, the randomized quick sort can take $O(n^2)$ in the worst case. However, such events are rare.

| Message | Description |
|---|---|
| REQ | A *request* from a contender to make the mediator recognize it as a potential winner. |
| POTW | The message from contender to notify the mediator that the requisite ($O(\sqrt{n \ln n})$) mediators have recognized it as a potential winner and to recognize it as a final winner. |
| DEC | To notify the mediator that the contender is dropping out as a contender. |

**Table 3.** Messages REQ (REQuest), POTW (POTential Winner), DEC (DECline) from contender to mediator.

| Message | Description |
|---|---|
| ACK | Positive Response for REQ, POTW messages. |
| NAK | Negative Response for REQ, POTW messages. |

**Table 4.** Messages from mediator to contender

to all its mediators $\Psi_{iw}$ ($\Psi_{iw}$ represents the set of mediators for the contender $\rho_i$ in round $w$, the last round of the protocol, see table 2). Some mediators, in the best case, may receive and process the message within $\delta$ time ($\delta << \tau$) and respond immediately. In the worst case, the mediator might have processed the message at $\tau$ time units after it was sent and the response might take another $\tau$ units. Therefore, $\rho_i$ waits for at least $2\tau$ units of time to receive responses from $\Psi_{iw}$. In the actual protocol, though, the $\rho_i$ waits for a maximum of $5\tau$ time units. We explain in Theorem 8 the reason for using $5\tau$. If $\rho_i$ receives a negative response from one of $\Psi_{iw}$ within this period, it sends a decline to the rest of $\Psi_{iw}$. Otherwise, upon receiving all the positive responses from $\Psi_{iw}$ (a contender will receive a response from every mediator within $5\tau$ units after sending a request), it sends a message to $\Psi_{iw}$ hinting that it could be a potential winner. $\rho_i$ waits for $2\tau$ units to receive a response from $\Psi_{iw}$ after sending the potential winner message. If it does not receive a negative response from any of $\Psi_{iw}$, it becomes the final winner of the protocol. Otherwise, if it receives a negative response in the intervening time period, it sends a decline message to the rest of $\Psi_{iw}$.

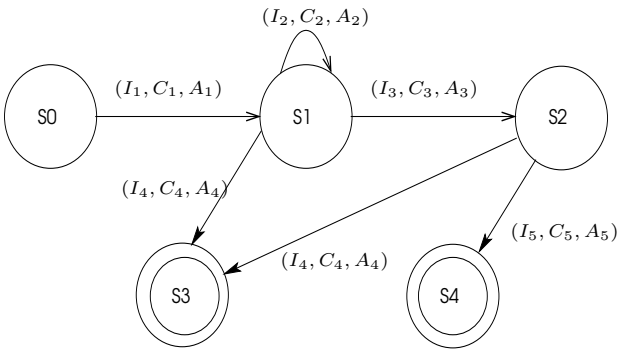


**Fig. 3.** State diagram of a contender.

A detailed state transition for the contender is shown in Figure 3. The subscripts in the figure correspond to rows in the table 5. Table 5 provides the state transition conditions with $I$ representing a received message (input), $C$ representing a condition, and $A$ representing the action taken by the contender. For example, $(I_3, C_3, A_3)$ in Figure 3 is represented by the third row of Table 5. It is read as, on an *input* of ACK, given the *condition* is counter= 1, the *action* is sending POTW message to $\Psi_{iw}$. A contender $\rho_i$ that enters the second phase of the protocol is initially in state $S0$. After it sends the request to $\Psi_{iw}$ in the second phase, it goes to state $S1$. If it receives a negative response, it goes to final state $S3$, where the $\rho_i$ is not the final winner. Otherwise, after receiving all the positive responses, it sends potential winner messages and goes to $S2$ from $S1$. If it does not receive any negative response from a mediator in state $S2$ within $2\tau$ time units, it goes to the final state $S4$, where the contender is the final winner.

| Index | I | C | A |
|---|---|---|---|
| 1 | | | Send $\text{REQ}_{\rho_i}$ message to the mediators in the set $\Psi_{iw}$. counter $\leftarrow \varphi_w$ |
| 2 | ACK | | counter $\leftarrow$ counter $-1$ |
| 3 | ACK | counter = 1 | Send $\text{POTW}_{\rho_i}$ to all mediators in $\Psi_{iw}$. timer $\leftarrow 2\tau$ |
| 4 | NAK | | Send DEC messages to all mediators in $\Psi_{iw}$. |
| 5 | | timer = 0 | Final Winner |

**Table 5.** Contender Transitions

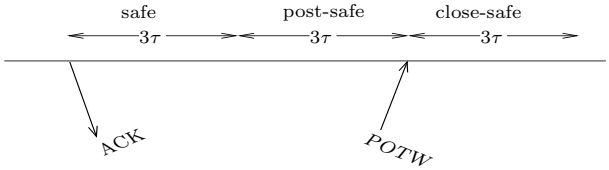


**Fig. 4.** States of a mediator.

A mediator responds to a contender in the second phase based on its second-phase state. We define three second-phase states for the mediator (see Figure 4)[4]. A mediator is in a *safe* state if it sent a positive response to a request in the last $3\tau$ time units. In this state, the mediator is committed to the positive response sent earlier. It responds negatively to a request from another contender with a lower random number value. Otherwise, it delays the response. A mediator is in a *post-safe* state, if a potential winner or decline message is not received from the contender that received the most recent positive response from the mediator, and if the difference between the current time and the time when the positive response was sent is between $3\tau$ and $6\tau$ units. In this state, the mediator can pre-empt (send a negative response to the contender to which it sent a positive response earlier) the earlier contender and send a positive response to some other contender having a higher random number value. A mediator is in a *close-safe* state

[4] The duration of each state is at most $3\tau$. However, it can be lesser than that based on the time of receipt of different messages.

if it received a potential winner message and has not received a decline message from the same contender and the difference between the current time and the time of receipt of potential winner message is not greater than $3\tau$ units. In the close-safe state, the mediator delays responses to any contender requests (similar to safe state). However, it takes different actions based on the messages received in the close-safe state. If a decline message is not received from the contender that sent a potential winner message, it sends a negative response to the delayed contenders and declares the contender that sent the potential winner message to be the final winner. Otherwise, the actions performed at the end of the close-safe state are similar to the actions at the end of the safe state.

On receiving a request from a contender $A$, if the mediator is not in any of the three second-phase states (i.e., it has not received a request from any contender for the second phase), it immediately sends a positive response. If it is in one of the second-phase states, the response is based on random numbers sent by the contender. In a safe/close-safe state, if the incoming request from contender $B$ has the largest random number (compared to other random numbers received by the mediator thus far), the response is delayed. Otherwise, a negative response is sent immediately. In a post-safe state, it sends a positive response to the request from $B$, which was delayed and preempts the earlier contender $A$. However, if a potential winner message from contender $A$ is received before the mediator enters the post-safe state, a negative response is sent to the delayed contender $B$. Also, if a request with a larger random number is received in a post-safe state, a positive response is sent back, and the earlier contender $A$ is preempted.

On receiving a potential winner message for the most recent positive response, the mediator goes to a close-safe state. If no decline messages are received in the duration of a close-safe state, the mediator declares the contender that sent the potential winner to be the final winner.

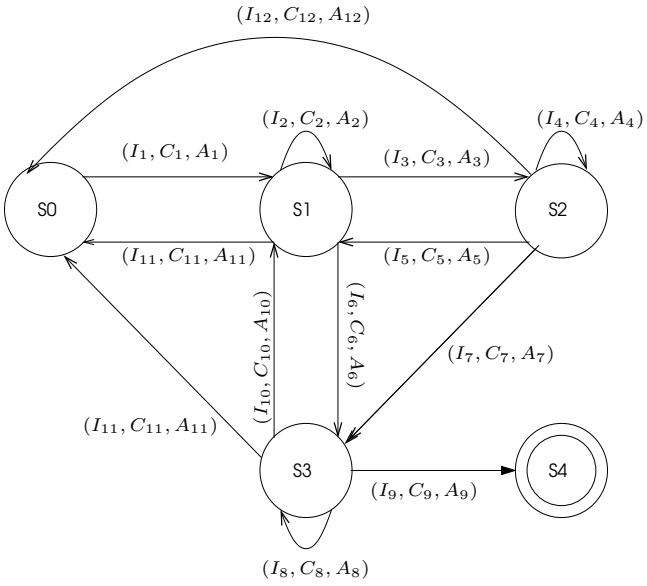


**Fig. 5.** State diagram of a mediator.

| Index | I | C | A |
|---|---|---|---|
| 1 | $\text{REQ}_{\rho_i}$ | | $\text{init}(\pi_i, \rho_i, \varnothing, \varnothing, 1)$ |
| 2 | $\text{REQ}_{\rho_j}$ | | act() |
| | DEC | $\pi_{\text{wait}} \neq \varnothing$ | $\text{ack}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{wait}}, \rho_{\text{wait}}, \varnothing, \varnothing, 1)$ |
| | $\text{POTW}_{\rho_j}$ | $\rho_j \neq \rho_{\text{curr}}$ | $\text{nak}(\rho_j)$ |
| | | timer $= 0$,<br>$\pi_{\text{wait}} \neq \varnothing$ | $\text{nak}(\rho_{\text{curr}})$<br>$\text{ack}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{wait}}, \rho_{\text{wait}}, \varnothing, \varnothing, 1)$ |
| 3 | | timer $= 0$<br>$\pi_{\text{wait}} = \varnothing$ | timer $\leftarrow 3\tau$ |
| 4 | $\text{REQ}_{\rho_j}$ | $\pi_j < \pi_{\text{curr}}$ | $\text{nak}(\rho_j)$ |
| 5 | $\text{REQ}_{\rho_j}$ | $\pi_j > \pi_{\text{curr}}$ | $\text{nak}(\rho_{\text{curr}})$<br>Send ACK to $\rho_j$<br>$\text{init}(\pi_j, \rho_j, \varnothing, \varnothing, 1)$ |
| 6 | $\text{POTW}_{\rho_j}$ | $\rho_{\text{curr}} = \rho_j$ | timer $\leftarrow 3\tau$<br>if $\rho_{\text{wait}} \neq \varnothing$<br>$\text{nak}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{curr}}, \rho_{\text{curr}}, \varnothing, \varnothing, 0)$ |
| 7 | $\text{POTW}_{\rho_j}$ | $\rho_j = \rho_{\text{curr}}$ | timer $\leftarrow 3\tau$<br>if $\rho_{\text{wait}} \neq \varnothing$<br>$\text{nak}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{curr}}, \rho_{\text{curr}}, \varnothing, \varnothing, 0)$ |
| 8 | $\text{REQ}_{\rho_j}$ | | act() |
| 9 | | timer $= 0$ | Final Winner $\leftarrow \rho_{\text{curr}}$ |
| 10 | DEC | $\pi_{\text{wait}} \neq \varnothing$ | $\text{ack}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{wait}}, \rho_{\text{wait}}, \varnothing, \varnothing, 1)$ |
| 11 | DEC | $\pi_{\text{wait}} = \varnothing$ | |
| 12 | DEC | $\rho_j = \rho_{\text{curr}}$ | |

**Table 6.** Mediator Transitions.

| Name | Actions |
|---|---|
| $\text{init}(\pi_c, \rho_c, \pi_w, \rho_w, \text{set})$ | $\phi_{\text{curr}} \leftarrow \pi_c$<br>$\rho_{\text{curr}} \leftarrow \rho_c$<br>$\pi_{\text{wait}} \leftarrow \pi_w$<br>$\rho_{\text{wait}} \leftarrow \rho_w$<br>if set is not 0 then<br>timer $\leftarrow 3\tau$ |
| act() | if $\pi_j < \pi_{\text{curr}}$<br>$\text{nak}(\rho_j)$<br>else<br>if $\pi_{\text{wait}} = \varnothing$<br>$\text{init}(\pi_{\text{curr}}, \rho_{\text{curr}}, \pi_j, \rho_j, 0)$<br>else<br>if $\pi_j < \pi_{\text{wait}}$<br>$\text{nak}(\rho_j)$<br>else<br>$\text{nak}(\rho_{\text{wait}})$<br>$\text{init}(\pi_{\text{curr}}, \rho_{\text{curr}}, \rho_j, \pi_j, 0)$ |
| ack(val) | Send ACK to val |
| nak(val) | Send NAK to val |

**Table 7.** Mediator Actions.

A formal description of the mediator actions is presented in the transition diagram in Figure 5, and the corresponding Table 6. The subscripts in the diagram correspond to the rows and the symbols in parentheses correspond to the columns in Table 6. State $S4$ is the final state where the eventual winner is declared. State $S0$ is the starting idle state, state $S1$ corresponds to the safe state, state $S2$ corresponds to the post-safe state, and state $S3$ corresponds to the close-safe state.

We further elucidate the second phase of the protocol with a few examples. In Figure 6 (example 1), $C_1$ and $C_2$ are two contenders, and $M_1$ and $M_2$ are mediators (in reality, $M_1$ may correspond to same process as $C_1$ or $C_2$, but for the sake of clarity we present it as a different entity). Let the random number generated by $C_1$ be the largest in the system. Contender $C_1$ sends requests to $M_1$ and $M_2$. Since $M_2$ has not received any requests previ-

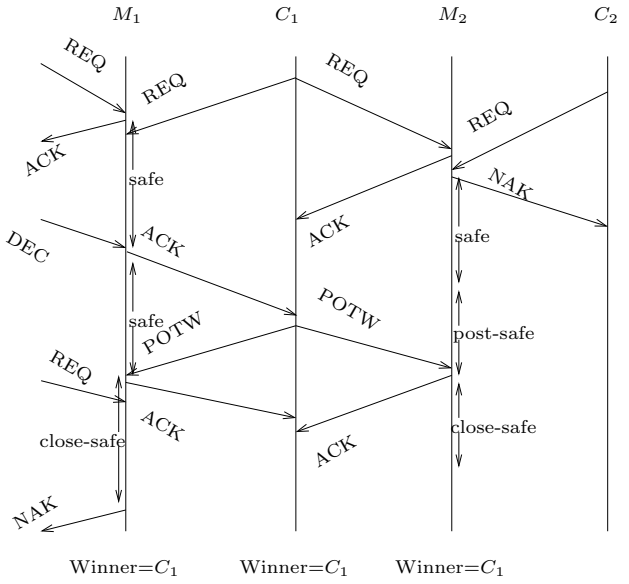


**Fig. 6.** Illustration (example 1) of the second phase of the asynchronous algorithm; $C_1$ is the final winner. Intervals between message arrivals and departures not drawn to scale.

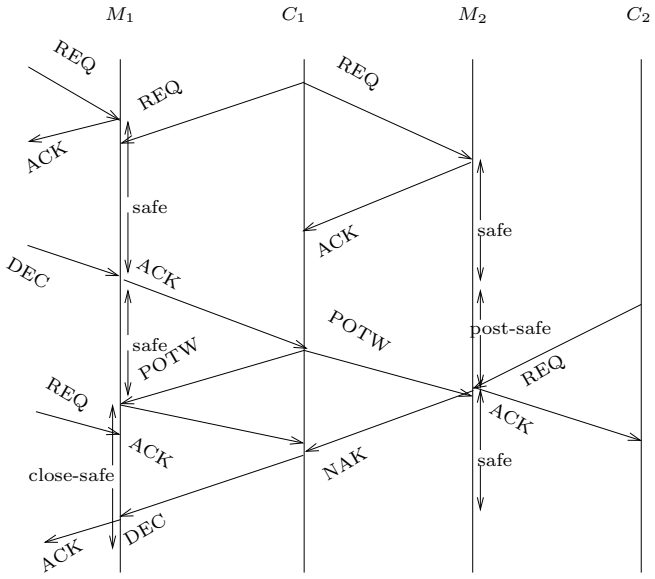


**Fig. 7.** An example (example 2) of the second phase of the asynchronous algorithm; intervals between message arrivals and departures not drawn to scale.

ously, it sends a positive response immediately. However $M_1$ had already sent a positive response to some other contender, say $C_3$, in the system and is in a safe state. The request from $C_1$ must wait for a maximum of $3\tau$ units at $M_1$ since the random number value generated by $C_1$ is the largest. In the mean time, $C_2$ also sends a request to $M_2$. Since $M_2$ is in a safe state and the random number of $C_2$ is smaller than that of $C_1$, a negative response is sent to $C_2$. Meanwhile, $C_3$ sends a decline message to $M_1$, since it might have received a negative response from some other mediator. Mediator $M_1$ sends a positive response to the next waiting contender $C_1$. After having received the requisite positive responses from all the mediators to which it had sent a request (the requisite number of responses is $O(\sqrt{n \ln n})$), $C_1$ sends a potential winner message to the mediators from which it has received positive responses and waits for $2\tau$ units of time. Because it receives a positive response from both $M_1$ and $M_2$, $C_1$ becomes the final winner. Suppose, if some other contender with a larger random number than $C_1$'s random number sends a request to $M_1$ when $M_1$ is in the close-safe state, then that contender receives a negative response from $M_1$ as shown in the Figure 6.

In Figure 7 (example 2), $C_1$ and $C_2$ are contenders, and $M_1$ and $M_2$ are mediators. Let the random number generated by $C_1$ be less than that generated by $C_2$. As shown in the figure, $C_1$ gets positive responses from its mediators but before it could send a potential winner message, $M_2$ is in a post-safe state. When $C_2$, which has a larger random number sends a request, $M_2$ sends a positive response immediately to $C_2$, and a negative response to $C_1$. This has a cascading effect as $C_1$ sends a decline to other mediators (here $M_1$), which had previously sent a positive response. Now, $M_1$ is free to respond positively to the request that has been delayed after the receipt of the decline from $C_1$.

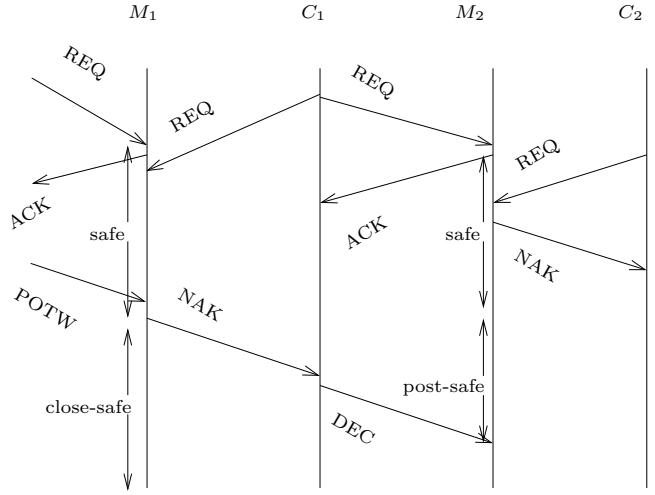


**Fig. 8.** An example of the second phase of the asynchronous protocol (example 3). Intervals between message arrivals and departures not drawn to scale.

In Figure 8(example 3), let the random number generated by $C_1$ be the largest. The response to request from $C_1$ is delayed by $M_1$ since it is in a safe state. $M_1$ receives a potential winner message from the contender (say $C_3$) to which it had sent a positive response earlier before the end of the safe state. Therefore, a negative response is sent to $C_1$. In turn, $C_1$ sends a decline to $M_2$ from which it received a positive response. Note that $M_2$ has already sent a negative response to another contender $C_2$, whose random number value was lower than that generated by $C_1$. A delay in response to $C_2$ would not help because a request from $C_2$ must intersect with a request from $C_3$ at some mediator (the probability of intersection of the set of mediators generated by two different contenders is

high) and $C_2$ will receive a negative response from that mediator also (given that a potential winner message was received by $M_1$ from $C_3$ when it was in a safe state).

### *4.3 Analysis of asynchronous protocol*

We prove that the bounds established for the synchronous protocol still hold for the asynchronous case as well.

**Theorem 6.** *The number of rounds in the protocol is* $O(\log n)$.

*Proof:* In the first phase of the asynchronous protocol, the contenders may potentially arrive at distinct points of time. A contender can proceed to the next round, if it sends requests to mediators, who have not received request from any other contender for the same round. Unlike in the synchronous case, where all contenders involved in a collision do not proceed to the next round, the contender which arrives first at all mediators for a specific round proceeds to the next round (apart from the contenders that sent requests to unique mediators). The expected number of contenders in round $j+1$ can be calculated as:

$$\begin{aligned} E[X_{j+1}|X_j = |\Phi_j|] &\approx \sum_{i=1}^{|\Phi_j|} e^{-\frac{\sigma_j^2(i-1)}{n}} \\ &= \sum_{t=0}^{|\Phi_j|-1} e^{-\frac{\alpha^i t}{n}} \text{ where } t = i-1 \\ &= 1 + \sum_{t=1}^{|\Phi_j|-1} e^{-\frac{\alpha^j t}{n}} \end{aligned}$$

Since $e^{-\frac{\alpha^j t}{n}}$ is a monotonically decreasing function, we can approximate $E[X_{j+1}|X_j = |\Phi_j|]$ by the following integral:

$$E[X_{j+1}|X_j = |\Phi_j|] \leq 1 + \int_0^{|\Phi_j|-1} e^{-\frac{\alpha^j t}{n}} dt \leq \frac{n}{\alpha^j}$$

Since the number of contenders on an average decreases by a factor of $\alpha$ and there is a single round in the second phase, the number of rounds in the protocol is $O(\log n)$. □

**Theorem 7.** *The total number of messages in the protocol sent by all nodes is* $O(n)$.

*Proof:* The expected number of messages, $N$, in the system in the first phase of the protocol is given by:

$$N = E[\sum_{j=1}^{w-1} X_j \sigma_j] = \sum_{j=1}^{w-1} E[X_j]\sigma_j < n\alpha \sum_{j=1}^{w-1} \frac{1}{\alpha^{\frac{j}{2}}}$$

Since $\alpha > 1$, the above summation converges. The number of messages sent in the second phase of the protocol is $O(\log^6 n)\sqrt{n \ln n}$. Therefore, the number of messages sent by all nodes in the system is $O(n)$. □

**Theorem 8.** *It takes a maximum of* $7\tau$ *units for a contender in the second phase of the protocol to know whether it is a winner or not.*

*Proof:* In the worst case, a request from a contender can take $\tau$ units. The mediator that received the request may have entered the safe state at approximately the same time as the request was received. The mediator may respond positively after $3\tau$ units (or in the worst case, a negative response after $3\tau$ units, if the earlier contender sent a potential winner). The response from the mediator to the contender can take at most $\tau$ units. Therefore, a contender need not wait for more than $5\tau$ units of time after it sends a request. Suppose, the contender gets a positive response from all its mediators, it sends a potential winner message, which takes at most $\tau$ units and the response from the mediator takes $\tau$ units. By accounting for the time taken for all the messages, a contender need not wait for more than $7\tau$ units in the worst case to know whether it is a winner or not. □

**Theorem 9.** *A contender will know whether it is a winner or not within* $O(\tau \log n)$ *time.*

*Proof:* For every contender, the time to know whether it is a winner or not is bounded by the total time taken for the first phase and the second phase. There are $O(\log n)$ rounds (by Theorem 6) in the first phase and the contender takes at most $O(\tau \log n)$ because for every round in the first phase, a contender spends $2\tau$ time units for communication with the mediators. By Theorem 8, a contender will not take more than $7\tau$ units in the second phase. Hence, a contender will know whether it is a winner or not in $O(\tau \log n)$. □

**Theorem 10.** *There is exactly one contender remaining at the end of the protocol w.h.p.*

*Proof:* A contender can become the final winner only when it has sent potential winner messages to its mediators. This can happen if and only if all its mediators sent a positive response. There are two states, *viz.* idle and post-safe, in which a mediator can send a positive response to a request. However after sending the potential winner messages, a contender can either win or lose. There are two scenarios of two or more contenders receiving positive responses from all its mediators.

- **Scenario 1:** More than one contender receives positive responses from all its mediators when the mediators were in idle state. This can happen only if the set of mediators chosen by two distinct contenders does not intersect. This can happen with probability $\frac{1}{n}$.
- **Scenario 2:** Since the set of mediators intersect w.h.p., two contenders might have received a positive response from the same mediator if and only if the mediator was in an idle/post-safe state for one contender and a post-safe state for the other contender. A mediator will not send a positive response in a safe or close-safe state. But the most recent positive response is outstanding and replaces the earlier positive response. The potential winner message from the

contender that received a positive response from the mediator (when it was in a idle state) will receive a negative response from the mediator.

Therefore, at any point during the execution of the protocol, by Theorem 3, a mediator will maintain only one contender as a potential winner and when a set of $\sqrt{n \ln n}$ of mediators hold the same contender as a potential winner, the corresponding contender becomes the final winner. □

## 5 Handling Failures

Failures are an integral part of large-scale distributed systems. In this section, we examine the impact of failures of mediators in our synchronous protocol. Contenders can also fail. However, as long as there is at least one contender who has not failed, a leader gets elected. In the event of all contenders failing or the last remaining contender failing, the scenario is analogous to any other election system where in the elected leader fails. This issue needs to be handled within the application framework. We assume fail stop failures, i.e., a process can stop responding at any point of time, based on the failure probability. A process can fail either before or after responding to a contender. If the mediator fails after responding to a contender, it means that the mediator has already arbitrated and does not pose a problem for the correct functioning of the algorithm. Consequently, we address the issue of mediators failing before they have sent any message to contenders.

In the first phase of the protocol, when a request is sent by a contender to $m$ mediators, it expects responses from each of the mediators. When a mediator to which the contender sends a message fails, the contender will not receive any response from the mediator as to whether the contender can proceed to subsequent rounds. We can apply two different approaches in this scenario:

1. A contender can proceed to the next round as long as it does not receive a negative response.
2. A contender can proceed to the next round if and only if it receives positive responses from all the mediators.

If we adopt the first approach, the number of contenders that proceed to subsequent rounds may be greater than the number of contenders proceeding to subsequent rounds when there are no failures. This affects the message and round complexity bounds. In the second approach, the number of contenders proceeding in the face of failures will be fewer than when there are no failures. This can lead to a case where the number of contenders in successive rounds decreases by a large factor resulting in a lower probability of having at least one contender at the end of the first phase of the protocol. This approach does, however, preserve the message and round complexity bounds. We adopt the second approach and identify the operating range of the high probabilistic correctness of our algorithm by providing bounds on the failure probability of processes.

**Theorem 11.** *In the first phase ($1 \leq j < w$), for $\gamma < \frac{1}{\sigma_j}$ the decrease in the number of contenders proceeding to subsequent round ($j+1$) is bounded by a factor of $\frac{1}{e^{\gamma\sigma_j}}$. (see Table 2 for definition of the parameters)*

*Proof:* Let $X_j$ is the number of contenders that proceed to round $j$,

$$E[X_{j+1}|X_j = |\Phi_j|] = |\Phi_j|\Big(1 - \frac{\sigma_j}{n}\Big)^{\sigma_j(|\Phi_j|-1)+n\gamma} \approx |\Phi_j| e^{-\frac{\sigma_j^2(|\Phi_j|-1)}{n} - \sigma_j\gamma} \quad (1)$$

The difference between the expectation of $X_j$ when compared to Lemma 1 is the addition of $n\gamma$ in (1). We do this to account for the failure of mediators and to let a contender proceed to the next round if and only if it has not sent a message to one of the failed processes.

The expected number of contenders proceeding to round $j+1$ is given by $\frac{|\Phi_j|}{2e^{\sigma_j\gamma}}$. When $\sigma_j\gamma < 1$, we have between $\frac{|\Phi_j|}{2e}$ and $\frac{|\Phi_j|}{2}$ contenders proceeding to round $j+1$ on an average. Therefore, when $\gamma < \frac{1}{\sigma_j}$, it limits the decrease in the number of contenders proceeding to subsequent rounds. □

It is necessary that every contender in the second phase intersect with other contenders in the system. In the second phase of the protocol, more messages are sent to negate the presence of failures in the system (i.e.,) every contender in the second phase needs to get a response from $\sqrt{n \ln n}$ other mediators. Given the bounds for $\gamma$ for the first phase, the message complexity of the second phase is maintained. Since, $\gamma$ is the failure probability, the expected number of failed processes is $n\gamma$. We use $n - n\gamma$ instead of $n$ in calculating the number of rounds in the first phase.

**Theorem 12.** *If $\gamma\sigma_{w-1} < 1$, a single leader is elected with probability $1 - O(\frac{e^{\gamma\sigma_{w-1}}}{\log^6 n})$, the message complexity is $O(n)$ and the round complexity is $O(\log n)$.*

*Proof:* We specify the modifications to the earlier proof without failures. Notice that $\sigma_i$ is a monotonically increasing function, we have $\sigma_{w-1} > \sigma_0$. From Theorem 11 and applying the method used in obtaining Lemma 1, we restate Lemma 1 as follows:

1. $\frac{n}{2^{i-1}e^{\gamma\sigma_{i-1}}}[g(\log^2 n)]^{i-1} \leq E[X_i] \leq \frac{n}{2^{i-1}e^{\gamma\sigma_{i-1}}}[h(\log^2 n)]^{i-1}$
2. $P(\ell_i < X_i < u_i) \geq g(\frac{\log^6 n}{e^{\gamma\sigma_{w-1}}})$

The result follows. The round and message complexities can be shown using the proofs from Theorems 4 and 5 respectively.

## 6 Applications and Relevance to Current Distributed Systems

For many applications in large scale distributed systems, probabilistic guarantees are sufficient as a tradeoff for

scalability. One such application, that forms our larger research and development goal, is a versioning-based distributed file system called Plethora [6]. In this and related systems, two contending processes for a common data object may modify the object concurrently, assuming they each have exclusive access. If both of these contenders commit their changes, a branch in the version tree is created and the two committed versions are installed as siblings in the version tree. While these versions may subsequently be reconciled, it is desirable that the number of such branches in the version tree be minimized. In this scenario, failed mutual exclusion (multiple processes gaining access to a mutually exclusive resource) merely results in a branch in the version tree. Minimizing the probability of such an occurrence, while minimizing associated overhead can be achieved using the protocol presented in this paper. Similarly, duplicate elimination of files in distributed stores [7] can be efficiently realized by using the algorithm presented in this section. In this section, we present issues that need to be addressed for a practical realization in realistic networks of the algorithm described in the paper.

One of the parameters of our protocol is the size of the network. An estimate of the number of processes in the system is needed to determine the number of messages that need to be sent in each round of the protocol. A number of researchers have addressed the problem of estimating network size [2,14,24]. For example, in [14], Horowitz *et al.* present an estimation scheme that allows a peer to estimate the size of its network based only on local information with constant overhead by maintaining a logical ring. Our approach does not, however, require the knowledge of the number of contenders.

Random walks are typically used to obtain uniform samples. The Metropolis-Hastings algorithm [3,13,22] provides a method to hasten the mixing time of a random walk to reach a stationary distribution and can be applied to different graphs. Gkantsidis *et al.* [10] show that successive steps of a random walk on an expander graph from a random point is equivalent to uniformly sampling in the entire graph. King and Saia [16] present a non random walk approach to choosing a random peer in a structured network. These results provide the needed algorithms for uniformly selecting mediators. Uniform sampling has many other applications [16].

## 7 Simulation Results

To explore the performance of our approach (RE), we simulated it on a network of 50,000 processes with power law connectivity i.e., the degree distribution of the processes follows a power-law distribution. To generate the network, we first generate the degrees of the processes according to the power-law distribution with power-law parameter $\alpha = 0.8$ and then connect the processes randomly. We assume that $\beta$ of the processes participate as contenders and these contenders are selected uniformly at random. We vary $\beta$ from 1% to 50%. These percentages are chosen to show the number of messages on the system based on the percentage of processes participating as contenders. We provide a comparison with the probabilistic quorum (PQ) approach, which is equivalent to the second phase of the RE approach. The simulations are performed over the asynchronous versions of the respective approaches.

### *7.1 Message Overhead*

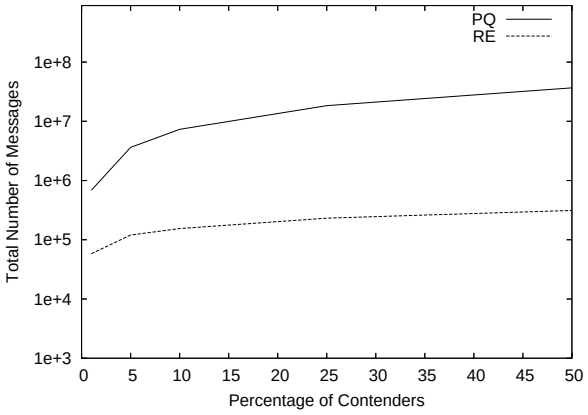


**Fig. 9.** Total number of messages in the system vs Percentage of Contenders.

We first compare the approaches with respect to the total number of messages exchanged in the system. Fig. 9 shows the number of messages for the two leader election approaches with varying $\beta$. The probabilistic quorum approach does not scale well when the percentage of contenders increases in the network. When 50% of the processes are contenders, the number of messages in the PQ approach is more than one order of magnitude greater than that of our approach.

### *7.2 Accuracy of the Protocols*

The final goal of both protocols (RE and PQ) is to find a unique leader in the system. To evaluate how close to this goal the protocols perform, we fix the percentage of contenders to 1% (500 processes) of the system size. In ten thousand different runs with different seeds, the protocols always produced a unique leader.

## 8 Related Work

A direct application of the probabilistic quorum algorithm developed by Malkhi *et al.* [21] to the leader election problem would result in $O(n\sqrt{n \ln n})$ number of messages and a single round. In our protocol, we increase the number of rounds to $O(\log n)$, but reduce the number of messages sent by all processes in all rounds to $O(n)$. This reduction is possible because winner at each round

in the first phase is chosen based on the non-intersection of their mediator sets with any other contender's mediator sets. We use the probabilistic quorum algorithm in the last round with a reduced ($O(\sqrt{\log n})$) set of contenders. In this case, the $O(n)$ message complexity is still preserved.

In [20], Maekawa proposes a $\sqrt{n}$ deterministic algorithm for mutual exclusion in distributed systems, where the distributed sites are arranged in a grid. A process that participates in the mutual exclusion algorithm wins if it can get exclusive access to processes on an arbitrarily selected single row and column. A direct application of this result for leader election results in $O(n\sqrt{n})$ messages.

In [1], Agrawal *et al.* improve upon Maekawa's algorithm for mutual exclusion by realizing quorum sets as sites lying on paths similar to trajectories of billiard balls. The size of the quorum generated is $\sqrt{2n}$ when compared to $2\sqrt{n}$ of Maekawa's algorithm. Even though their mutual exclusion algorithm can be extended to solve the leader election problem deterministically, the message complexity of their method is of the same order as that of Maekawa's algorithm ($O(n\sqrt{n})$) and hence suboptimal when compared to the algorithm presented in this paper.

Gupta *et al.* [12] propose a probabilistic leader election protocol for large groups. The complexity of the protocol is $O(K^4 * n)$ messages per election round, where $K$ is the filter value on the number of processes that can participate in the relay phase. The success of the algorithm is guaranteed only with probability depending on various system parameters viz., the view probability (the probability that a random process has another random process in its view), $K$ value, the system size among others. In their paper, it is necessary that the hash function and $K$ value, used to decide whether a process will participate in the relay phase, should be the same (or approximately the same) for all processes in the group. While the hash function can be the same for all processes, agreement on the $K$ value is not straight forward because it decides the number of processes that participate in the relay phase and hence is a function of group size. If the $K$ value is a predetermined constant, then it is not clear on how this method can be adopted for varying group sizes. Also, while simulation results in the paper shows that the number of election rounds is quite less, the number of rounds as a function of the system size is not clearly specified. Furthermore, the probability of success of the protocol is significantly dependent on the filter value, the view probability among other system parameters. However, a significant contribution of their work is in recognizing the usefulness of probabilistic approaches in large scale systems, where correctness can be sacrificed occasionally for scalability. We also design a probabilistic algorithm such that the probability of electing exactly one leader tends to unity as the system size approaches infinity. Furthermore, the message complexity is linear in the system size and no assumptions are made with respect to the knowledge of any constants. While the exact number of messages exchanged is dependent on the number of initial contenders (or group size), the knowledge of the number of contenders is irrelevant for our approach.

In [28], Schooler *et al.* propose two interesting leader-election algorithms for multicast groups. Both algorithms (LE-A and LE-S) use announcements to decide the leader. In the first algorithm (LE-A), a process initially announces itself as the leader to the group and listen for announcements of the other members of the groups. Leaders are selected based on their identifiers, the process with the greatest identifier is selected as leader. On receiving an announcement, a process decides locally if the current leader must be changed based on the process identifier received in the announcement. The second algorithm, (LE-S) includes a suppression phase to avoid the initial announcement. In LE-S, processes wait for a random time before sending their leader announcements. An extension to the algorithm is making the suppression wake-up timer value the actual "id" that serves as the basis of the resolution algorithm. Thus, when processes with smaller timer values announce their leadership, they are deemed leaders and appointed leaders earlier. This (LE-S) approach greatly reduces the number of messages, since it probabilistically suppresses unnecessary leadership announcements. This reduction in the number of messages shares a similarity in its goal with the first phase of our approach, where the number of contenders are reduced in successive rounds. In [28], the value upon which the leadership decision is based is selected from a range that deliberately minimizes collision. The contenders in our approach also generate a random number from a range ($[0..n^4]$) for the sake of uniqueness. Their approach has the potential on the one hand to lead to more messages in the network, yet on the other hand to reduce the likelihood of implosion with large $n$ when coupled with suppression. A valuable aspect of their work is that it considers message loss as a fundamental concern for distributed leader election algorithms.

Mullender and Vitanyi propose a "distributed matchmaking" problem in [23] to study distributed control issues arising in name servers, mutual exclusion, replicated data management, that involve making matches between processes. In a general network, they propose that a connected graph of $n$ processes can be divided into $O(\sqrt{n})$ connected sub graphs of diameter $O(\sqrt{n})$ each. Our algorithm (in the first phase) can be viewed as constructing sub graphs in a step by step fashion instead of a single round which results in a better message complexity.

There has been significant work in developing mutual exclusion algorithms for shared memory models [17,15]. These could be extended for purposes of leader election. In [17], Kushilevitz and Rabin correct the randomized mutual exclusion algorithm presented in [25]. The authors develop a randomized algorithm for mutual exclusion in a shared memory model. In [15], Joung proposes a new problem called *Congenial Talking Philosophers* to model the mutual exclusion problem and provides several criteria to evaluate solutions to the problem. Our leader election algorithm is for a large scale distributed system where it is not feasible for processes to share memory.

## 9 Conclusion

This paper presents the design of an efficient randomized algorithm for leader election in large-scale distributed systems. The algorithm guarantees correctness with high probability and has optimal message complexity $O(n)$. To our knowledge, this is the first result providing high probabilistic guarantees with optimal message complexity for a general topology. We propose variants of the algorithm for synchronous as well as asynchronous environments. We give an analysis for the correctness of the algorithm and bounds on the number of messages and the number of rounds.

**Acknowledgements**

We sincerely thank Prof. Vassos Hadzilacos and the anonymous reviewers for their insightful comments which helped us improve the quality of the paper.

## References

1. D. Agrawal, Ö. Egecioglu, and A.E. Abbadi. Billiard quorums on the grid. *Inf. Process. Lett.*, 64(1):9–16, 1997.
2. M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical Report, CS Department, Stanford University, 2003.
3. S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing markov chain on a graph. *SIAM Review vol 46, no 4*, pages 667–689, Dec 2004.
4. I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In *DISC '98: Proceedings of the 12th International Symposium on Distributed Computing*, pages 104–118, London, UK, 1998. Springer-Verlag.
5. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.
6. R.A. Ferreira, A. Grama, and S. Jagannathan. Plethora: A Locality Enhancing Peer-to-Peer Network. *Journal of Parallel and Distributed Computing (in print)*.
7. R.A. Ferreira, M.K. Ramanathan, A. Grama, and S. Jagannathan. Randomized Protocols for Duplicate Elimination in Peer-to-Peer Storage Systems. In *Proceedings of The Fifth IEEE International Conference on Peer-to-Peer Computing*, Konstanz, Germany, Sep 2005.
8. C. Fetzer and F. Cristian. A highly available local leader election service. *IEEE Trans. Software Eng.*, 25(5):603–618, 1999.
9. G.N. Frederickson and N.A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987.
10. C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM, 2004*, Hong Kong, March 2004.
11. Gnutella. http://gnutella.wego.com/.
12. I. Gupta, R. Renesse, and K.P. Birman. A probabilistically correct leader election protocol for large groups. In *DISC '00: Proceedings of the 14th International Conference on Distributed Computing*, pages 89–103, London, UK, 2000. Springer-Verlag.
13. W. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika, 57*, pages 97–109, 1970.
14. K. Horowitz and D. Malkhi. Estimating network size from local information. *The IPL journal 88(5)*, pages 237–243, December 2003.
15. Y. Joung. Asynchronous group mutual exclusion. *Distributed Computing*, 13(4):189–206, 2000.
16. V. King and J. Saia. Choosing a random peer. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 125–130, New York, NY, USA, 2004.
17. E. Kushilevitz and M.O. Rabin. Randomized mutual exclusion algorithms revisited. In *PODC '92: Proceedings of the eleventh annual ACM symposium on Principles of distributed computing*, pages 275–283, 1992.
18. S. Lin, Q. Lian, M. Chen, and Z. Zhang. A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems. In *IPTPS*, pages 11–21, 2004.
19. N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.
20. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2):145–159, 1985.
21. D. Malkhi, M. Reiter, A. Wool, and R. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2):184–206, November 2001.
22. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys., Vol. 21*, pages 1087–1092, 1953.
23. S.J. Mullender and P.M.B. Vitanyi. Distributed matchmaking. *Algorithmica*, 3:367–391, 1988.
24. D. Psaltoulis, D. Kostoulas, I. Gupta, K. Birman, and A. Demers. Practical algorithms for size estimation in large and dynamic groups. Springlass project at Cornell.
25. M.O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log_2$ N valued shared variable. *Journal of Computer and System Sciences*, 25(1):66–75, 1982.
26. S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001.
27. A.I.T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.
28. E.M. Schooler, R. Manohar, and K.M. Chandy. An analysis of leader election for multicast groups. *Technical Report, ATT Labs-Research, Menlo Park, CA*, Feb 2002.
29. I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.
30. W. Szpankowki. *Average Case Analysis of Algorithms on Sequences*. Wiley, 2001.
31. A.S. Tanenbaum and M.V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.
32. G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.