Department of Computer Science Technical Reports

Department of Computer Science

2004

Randomized Leader Election

Murali Krishna Ramanathan

Ronaldo A. Ferreira

Suresh Jagannathan
Purdue University, suresh@cs.purdue.edu

Ananth Y. Grama
Purdue University, ayg@cs.purdue.edu

Report Number:
04-028

RANDOMIZED LEADER ELECTION

Murali Krishna Ramanathan
Ronaldo A. Ferreira
Suresh Jagannathan
Ananth Y. Grama
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-028
October 2004

Randomized Leader Election

Murali Krishna Ramanathan
Ronaldo A. Ferreira
Suresh Jagannathan
Ananth Grama

{rmk, rf, suresh, ayg}@cs.purdue.edu
Department of Computer Sciences
Purdue University, IN 47907

Abstract

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 nodes), has round complexity logarithmic in the number of nodes 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. In the first phase, the algorithm reduces the number of contending nodes and in the second phase, it resolves a winner. We provide algorithms for both synchronous and asynchronous models and prove that message and round complexities hold under both models. We also address the impact of failures (and departures) on correctness, and overheads associated with our algorithm.

1 Introduction

The problem of leader election in distributed systems is an important and well-studied one. A leader election algorithm is formally characterized as follows [31, 17]: "*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, there is exactly one process in the state* **leader** *and all other processes are in the state* **lost**". Variants of this problem require all processes in the ensemble to know the identity of the leader [11]. 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 [30].

The central challenge of efficient, scalable, and robust leader election algorithms is to simultaneously minimize the number of messages and execution time in the presence of node and link failures. With these objectives, a number of leader election protocols have been proposed [17, 23, 8, 6, 4, 11, 28]. The emergence of novel distributed paradigms and underlying platforms such as unstructured and structured peer-to-peer (P2P) systems [29, 26, 27, 10] for resource sharing pose interesting new variants of this problem. For example, in unstructured P2P systems, there is no global knowledge at any single point in the network. Even simple problems such as determining the number of nodes in the system lead to significant challenges. Node arrivals and departures are frequent – contributing to the difficulties associated with gathering system state accurately.

In a classic result on distributed consensus [7], Fischer *et al.* show that consensus is impossible in the presence of unreliable processes. This motivates the design of a variety of probabilistic leader election schemes [11, 28]. In [11], 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 that is optimal in terms of message complexity ($O(n)$ for a distributed system with n nodes), has round complexity logarithmic in n, and is correct with high probability (w.h.p.[1]). This is in contrast to known algorithms (Section 7) that are either suboptimal in terms of number of messages [19, 18, 16], require a larger number of rounds [17], or are restrictive in terms of underlying topology[11]. It is important to note that while many of the traditional leader election algorithms provide absolute guarantees for the election of a unique leader, our algorithm guarantees leader election with high probability. In this sense, our algorithm is targeted towards the current generation of P2P and large-scale distributed systems [29, 26, 27, 10], in which link and node failures necessitate the use of probabilistic approaches.

1.1 Technical Contributions

We summarize the main contributions of this paper 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 nodes 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 nodes participating in the leader election algorithm.

3. An asynchronous version of the algorithm along with bounds on failure probability of nodes, so that the algorithm can be effectively realized for a range of distributed applications.

[1] Throughout this paper, w.h.p. (with high probability) denotes probability $1 - \frac{1}{n^{\Omega(1)}}$.

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. Failures are an integral part of distributed systems. The impact of failures on the performance of our protocols is addressed in Section 5. Related work is presented in Section 7. Conclusions and avenues for future research are outlined in Section 8.

2 System Definitions

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

- **Process:**[2]: 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.

- **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 changes 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 communication 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 any negative responses from the mediators through the entire execution of the protocol.

[2] We use 'process' and 'node' interchangeably in the rest of the paper.

3 A Synchronous Leader Election Protocol

In this section, we present a randomized synchronous algorithm for leader election, which is scalable with respect to system size and offers high probabilistic guarantees for correctness. 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 $\log n - 1$ rounds for a system of n processes. In round i of this phase, each contender casts φ_i balls into n bins (we derive precise expressions for φ_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. 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, nodes 2, 4, 5, and 6 proceed, and in the second round, nodes 5 and 6 proceed.

In a realization of this balls-and-bins abstraction, a process can be a contender as well as a mediator at the same time. Casting a ball into a randomly chosen bin corresponds to a message from a contender to a randomly chosen 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 a contender sends messages (the number of balls to cast) to in a round j, φ_j is calculated (described in Section 3.1) independently by every contender based on the system size (total number of processes) and the round number. This number, if carefully selected, can reduce the number of contenders by half, on an average, after every round. It follows from this that the number of rounds is logarithmic in the number of contenders. However, if the number of contenders is not known, then each contender executes the algorithm assuming that every process is a contender and the number of rounds is logarithmic in the total number of processes.

The second phase of the protocol consists of a single round. Each remaining contender generates a random number in the range $[0..n^4]$ [17][3] and sends this number to a set of $\sqrt{n \ln n}$ mediators picked uniformly at random. A contender wins this phase 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 who generates the larger random number. A crucial difference between the two phases is that in the first phase a contender wins a round only if it sends a message to an exclusive set of mediators, while in the second phase, a winner is decided based on the intersection of the selected sets. The reason for this difference is that in the first phase, the number of contenders are reduced just enough so that a few of them can 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] This range is selected for uniqueness.

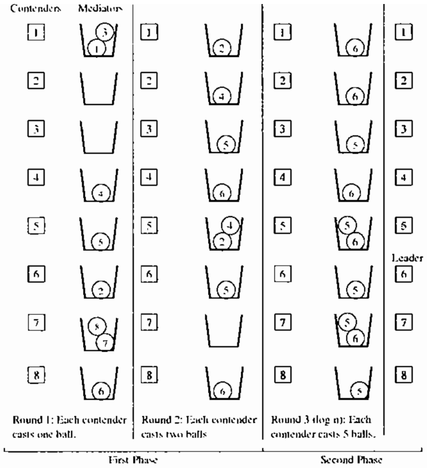
Contenders
Mediators
Leader
Round 1: Each contender casts one ball.
Round 2: Each contender casts two balls
Round 3 (log n): Each contender casts 5 balls.
First Phase
Second Phase

Figure 1: Illustration of the synchronous protocol – contenders are illustrated by squares, mediators by buckets, and messages from contenders to mediators by labeled balls (the label indicating the source of the ball). In all rounds, contenders that are no longer in the running are shaded dark. In the first round, each contender casts one ball and contenders 2, 4, 5, and 6 proceed since their balls uniquely occupy their respective buckets. 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 since it has a higher node id.

Based on the application, the contenders that did not get elected as the leader may retry after a specific period of time. Furthermore, the elected leader could subsequently inform its mediators that it is no longer the leader. We consider these issues to be specific to the application context and their solutions can be built on top of the basic protocol.

3.1 The Synchronous Leader Election Protocol

We consider a distributed system of n processes represented by the set $\Gamma_n = \{a_i | 1 \leq i \leq n\}$, for processes $a_1, a_2, \ldots, a_n$. The set of contenders in round j of the protocol is represented by $\Phi_j = \{\rho_i | 1 \leq i \leq |\Phi_j|\}$, where $\Phi_j \subseteq \Gamma_n$, $\Phi_{j+1} \subseteq \Phi_j$ and $\rho_1, \rho_2, ..., \rho_{|\Phi_j|}$ are the contenders. We associate a unique integer π_i with every contender ρ_i. The protocol concludes by declaring a unique member in Φ_w to be the final winner, where w corresponds to the final round in the protocol, and $\{\forall j : j < w, |\Phi_j| > 1, |\Phi_w| = 1\}$. We show that $w < \log(n)$. Various system parameters are summarized in Table 1.

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

Notation	Description
Γ_n	Distributed system with n processes.
a_i	A process in the distributed system.
ρ_i	A contender in a round.
Φ_j	Set of contenders in round j.
π_i	Random number generated by contender ρ_i.
w	Final round of the protocol.
φ_j	Number of mediators in round j.
Ψ_{ij}	Set of mediators for contender ρ_i in round j.
κ_{ij}	Set of messages received by mediator a_i in round j.
C_n	Maximum number of contenders in round w.

Table 1: System Parameters

- Set

$$\varphi_j = \begin{cases} \sqrt{\frac{n \ln 2}{|\Phi_j|-1}}, & \text{if } j < w \\ \sqrt{n} \ln n, & \text{otherwise} \end{cases}$$

  where φ_j is the number of mediators in round j, $|\Phi_j|$ is the number of contenders and n is the total number of processes in the system. The values of φ_j are selected in this manner to guarantee bounds on number of messages and rounds. Details of this construction are provided in Section 3.2.

- Let Ψ_{ij} be the set of φ_j processes selected uniformly at random from Γ_n. Send the ordered pair (ρ_i, π_i) to all processes in Ψ_{ij}.

- Proceed to the next round $j+1$ if and only if positive responses are received from all the processes in Ψ_{ij}.

Each process a_i in Γ_n performs the following steps in round j:

- Receive messages from ρ_k's in Φ_j and populate the set κ_{ij} with ordered pairs (ρ_k, π_k).

- **First Phase:** $j < w$

  - If $|\kappa_{ij}| = 1$, then send a positive response to ρ_k in κ_{ij} and proceed to round $j + 1$.
  - Otherwise, send a decline message to every ρ_k present in κ_{ij} and proceed to the round $j + 1$.

- **Second Phase:** $j = w$

  - For every (ρ_k, π_k) in κ_{ij}, find the largest π_k and send a positive response to the corresponding ρ_k. Send a decline message to the rest of the ρ_k's.

3.2 Analysis

We now precisely quantify the overhead and the probability of electing a unique leader using our algorithm.

Theorem 1 *In the first phase of the protocol, the number of contenders in round $j+1$ is approximately half the number of contenders in round j.*

Proof: Let X_{ij} be an indicator random variable corresponding to process ρ_i in round j obtaining a positive response from every process in Ψ_{ij}.

$$X_{ij} = \begin{cases} 1, & \text{if } \forall s : s \in \Psi_{ij},\ s \text{ sent a positive response} \\ 0, & \text{otherwise} \end{cases}$$

The expectation of X_{ij} is given by

$$\begin{aligned} E[X_{ij}] &= Pr(X_{ij} = 1) = \left(1 - \frac{\varphi_j}{n}\right)^{\varphi_j(|\Phi_j|-1)} \\ &\approx e^{-\frac{\varphi_j^2(|\Phi_j|-1)}{n}} \end{aligned} \tag{1}$$

Let Y_{j+1} be a random variable that represents the number of processes proceeding to round $j+1$. We have $Y_{j+1} = \sum_{i=1}^{|\Phi_j|} X_{ij}$. The expectation of Y_{j+1} is given by

$$E[Y_{j+1}] = E[\sum_{i=1}^{|\Phi_j|} X_{ij}] = \sum_{i=1}^{|\Phi_j|} E[X_{ij}] \approx |\Phi_j| e^{-\frac{\varphi_j^2(|\Phi_j|-1)}{n}} \tag{2}$$

Since $\varphi_j = \sqrt{\frac{n \ln 2}{|\Phi_j|-1|}}$, we have

$$E[Y_{j+1}] \approx \frac{|\Phi_j|}{2} \tag{3}$$

□

Theorem 2 *The probability of having at least one contender at the end of the first phase is $1-\epsilon$, i.e., $Pr(Y_w > 0) \geq 1 - \epsilon$, where $\epsilon \rightarrow 0$ as $n \rightarrow \infty$.*

Proof: By using the Second Moment Method, (based on Chebyshev's Inequality), for a non-negative integer-valued random variable Y_w, we have

$$Pr(Y_w = 0) \leq \frac{Var(Y_w)}{(E[Y_w])^2} = \frac{E[Y_w^2]}{(E[Y_w])^2} - 1 \tag{4}$$

Let $k = |\Phi_{w-1}|$,

$$\begin{aligned} E[Y_w^2] &= \sum_{i=1}^{k} E[Y_{iw}^2] + \sum_{i \neq j} E[Y_{iw} Y_{jw}] \\ &= E[Y_w] + k(k-1)E[Y_{1w}Y_{2w}] \\ &= E[Y_w] + k(k-1)(1 - \frac{\varphi_{w-1}}{n})^{\varphi_{w-1}(2k-3)} \\ &= E[Y_w] + k(k-1)e^{-\frac{\varphi_{w-1}^2}{n}(2k-3)} \end{aligned}$$

Without loss of generality, assume that $C_n < |\Phi_1|$. At the start of the final round of the first phase, in the worst case, there are at least $C_n + 1$ contenders. By substituting for $|\Phi_{w-1}| = C_n + 1$, $\varphi_{w-1} = \sqrt{\frac{n \ln 2}{|\Phi_{w-1}|-1}}$, $E[Y_w] = \frac{|\Phi_{w-1}|}{2}$ in (4), we have

$$Pr(Y_w = 0) \leq \frac{2}{C_n + 1} + \frac{4C_n}{(C_n+1)2^{\frac{2C_n - 1}{C_n+1}}} - 1.$$

Therefore, $Pr(Y_w > 0) \geq 1 - \epsilon$, where

$$\epsilon = \frac{2}{C_n + 1} + \frac{4C_n}{(C_n+1)2^{\frac{2C_n - 1}{C_n+1}}} - 1.$$

For $C_n = \Theta(\sqrt{\log n})$, $\epsilon \to 0$ as $n \to \infty$. □

Also, if we fix the number of rounds such that the expected number of contenders in the last round of the first phase of the protocol is $\frac{2 \log n}{\delta^2}$, where $0 < \delta < 1$ and δ is a constant, then by Chernoff bound [21], we have $P(Y_w \leq (1-\delta)E[Y_w]) \leq \frac{1}{n}$. For a typical value of $\delta = \frac{1}{2}$, we have $P(Y_w \leq 4 \log n) \leq \frac{1}{n}$. Informally, the number of contenders entering the second phase of the protocol Y_w will be close to the expected value w.h.p. Correspondingly, the number of messages in the second phase would be $\frac{2 \log n}{\delta^2}\sqrt{n \ln n}$, which still maintains the message complexity. In the rest of the paper, we make arguments in terms of $C_n = \Theta(\sqrt{\log n})$. Note that these arguments still hold for $C_n = 4 \log n$.

Theorem 3 *There is exactly one contender remaining at the end of the protocol w.h.p.*

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

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

Since the sets intersect w.h.p, only the process ρ_i with the highest value of π_i receives positive responses from all processes in Ψ_{iw}. The rest of the contending processes have at least one decline message. Hence, a unique final winner of the protocol is chosen w.h.p. □

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

Proof: From (3), it is clear that in every round of the first phase, the number of contenders reduces by at least half. Since there is only one round in the second phase, the number of rounds in the protocol is $O(\log |\Phi_1|)$. In the worst case, when $|\Phi_1| = n$, the number of rounds of the protocol is $O(\log n)$. □

Theorem 5 *When $|\Phi_1| = n$. the total number of messages in the system is $O(n)$.*

Proof: The total number of messages in the system in the first phase is $2\left(\sum_{j=1}^{w-1} |\Phi_j|\sqrt{\frac{n \ln 2}{|\Phi_j|-1}}\right)$. When $|\Phi_1| = n$, the number of contenders in round j is approximately $\frac{n}{2^{j-1}}$, which results in $2n(\ln 2)\left(\sum_{j=1}^{w-1} \frac{1}{2^{\frac{j-1}{2}}}\right)$ messages. Hence, 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.

At the end of the first phase, the number of remaining contenders is no more than C_n. Since each contender sends at most $\sqrt{n \ln n}$ messages in the second phase, the maximum number of messages exchanged in the second phase is $2c \ln n\sqrt{n}$, where $C_n = c\sqrt{\ln n}$ and c is a constant. Hence, the total number of messages in the protocol is $O(n)$. □ When $|\Phi_1| = O(1)$, the total number of messages in the system is $O(\sqrt{n \ln n})$.

4 An Asynchronous Leader Election Protocol

The design of the asynchronous protocol is largely motivated by its synchronous counterpart. However, asynchrony poses significant challenges in designing the protocol, since message and round complexities of the synchronous protocol must be preserved. We present a suitably modified asynchronous protocol and prove that it is correct and that it follows the bounds presented in Section 3.

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 synchronous protocol, all messages from the contenders to a specific mediator are received at the same time. The mediator, after receiving the messages, can select one of the contenders to send a positive response. It need not be concerned about any future messages that might be received from a new contender in the second phase. In the asynchronous version, messages from contenders need not be received at the same time. 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 can be provided as a parameter to the algorithm). As before, the asynchronous protocol works in two phases.

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 2: Messages from contender to mediator

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

Table 3: Messages from mediator to contender

4.1 The First Phase

In the first phase of the asynchronous protocol, a contender ρ_i sends requests, along with a round number j, to the mediators in the set Ψ_{ij}. The mediators in the set Ψ_{ij} are picked uniformly at random as in the synchronous protocol. The number of mediators in Ψ_{ij} is $\varphi_j = \sqrt{\alpha^j}$, where $\alpha > 1$ is a constant (typically α is set to 2). We later show that this value keeps the message and round complexities the same as the synchronous protocol. A contender ρ_i proceeds to round $j+1$ if it receives positive responses from all the mediators in its set Ψ_{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 ρ_i in round j, if entry j in V at M is zero, M sends a positive response to ρ_i and sets the entry j to ρ_i (signifying that the winner of the jth round at M is ρ_i). Otherwise a negative response is sent to ρ_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 ρ_k. This is because M has knowledge that ρ_k is in an advanced round (ahead of ρ_i) and hence is more likely to be elected the leader when compared to ρ_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

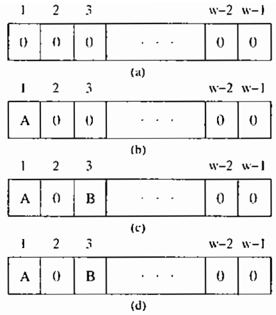

Figure 2: (a) Vector V before receiving any requests. (b) V after a request from A for round number 1. (c) V after a request from B for round number 3. (d) V after a request from D for round number 3

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.

4.2 Second Phase

Let τ 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 ρ_i sends a request to all its mediators Ψ_{iw} (Ψ_{iw} represents the set of mediators for the contender ρ_i in round w, the last round of the protocol, see table 1). Some mediators, in the best case, may receive and process the message within δ time ($\delta << \tau$) and respond immediately. In the worst case, the mediator might have processed the message at τ time units after it was sent and the response might take another τ units. Therefore, ρ_i waits for at least 2τ units of time to receive responses from Ψ_{iw}. In the actual protocol, though, the ρ_i waits for a maximum of 5τ time units. We explain in Theorem 9 the reason for using 5τ. If ρ_i receives a negative response from one of Ψ_{iw} within this period, it sends a decline to the rest of Ψ_{iw}. Otherwise, upon receiving all the positive responses from Ψ_{iw} (a contender will receive a response from every mediator within 5τ units after sending a request), it sends a message to Ψ_{iw} hinting that it could be a potential winner. ρ_i waits for 2τ units to receive a response from Ψ_{iw} after sending the potential winner message. If it does not

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

Table 4: Contender Transitions

receive a negative response from any of Ψ_{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 Ψ_{iw}.

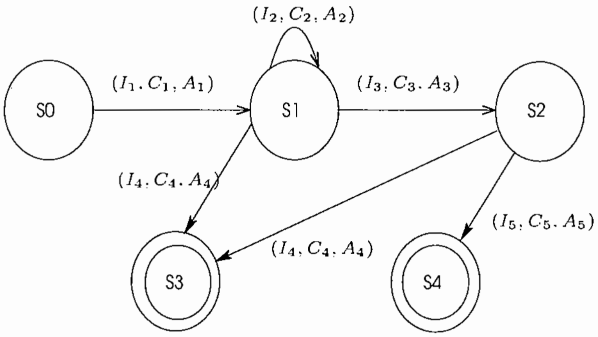

Figure 3: State diagram of a contender.

A detailed state transition for the contender is shown in Figure 3. The subscripts in the figure correspond to rows in the table 4. Table 4 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 4. It is read as, on an *input* of ACK, given the *condition* is *counter* $= 1$, the *action* is sending POTW message to Ψ_{iw}. A contender ρ_i that enters the second phase of the protocol is initially in state $S0$. After it sends the request to Ψ_{iw} in the second phase, it goes to state $S1$. If it receives a negative response, it goes to final state $S3$, where the ρ_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 2τ time units, it goes to the final state $S4$, where the contender is the final winner.

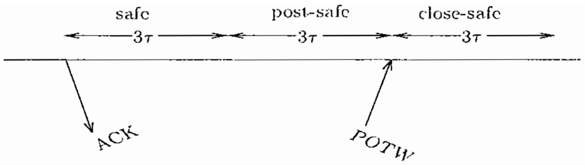

Figure 4: States of a mediator.

We define three second-phase states for the mediator (see Figure 4)[4] based on which the mediator responds to a contender. A mediator is in a *safe* state if it sent a positive response to a request in the last 3τ 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 3τ and 6τ 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 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 3τ 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

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

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.

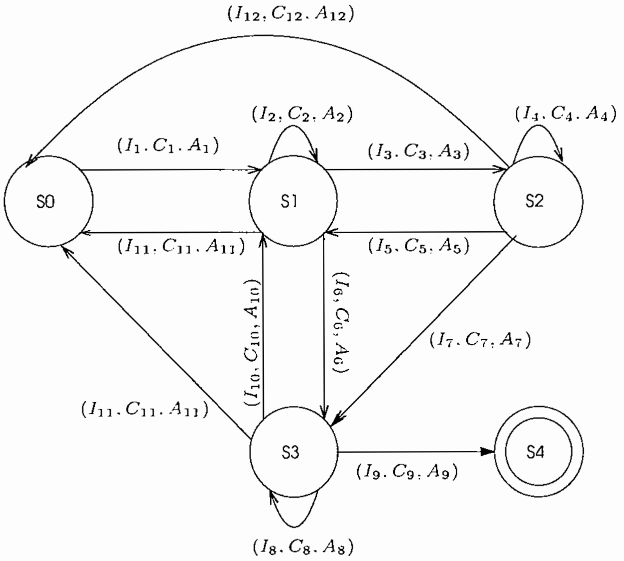

Figure 5: State diagram of a mediator.

A formal description of the mediator actions is presented in the transition diagram in Figure 5, and the corresponding Table 5. The subscripts in the diagram correspond to the rows and the symbols in parentheses correspond to the columns in Table 5. 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 previously, 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 3τ 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

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

Table 5: Mediator Transitions.

Name	Actions
$init(\pi_c, \rho_c, \pi_w, \rho_w, set)$	$\phi_{curr} \leftarrow \pi_c$ $\rho_{curr} \leftarrow \rho_c$ $\pi_{wait} \leftarrow \pi_w$ $\rho_{wait} \leftarrow \rho_w$ if set is not 0 then $timer \leftarrow 3\tau$

$act()$	if $\pi_j < \pi_{curr}$
	$nak(\rho_j)$
	else
	if $\pi_{wait} = \varnothing$
	$init(\pi_{curr}, \rho_{curr}, \pi_j, \rho_j, 0)$
	else
	if $\pi_j < \pi_{wait}$
	$nak(\rho_j)$
	else
	$nak(\rho_{wait})$
	$init(\pi_{curr}, \rho_{curr}, \rho_j, \pi_j, 0)$

$ack(val)$	Send ACK to val
$nak(val)$	Send NAK to val

Table 6: Mediator Actions.

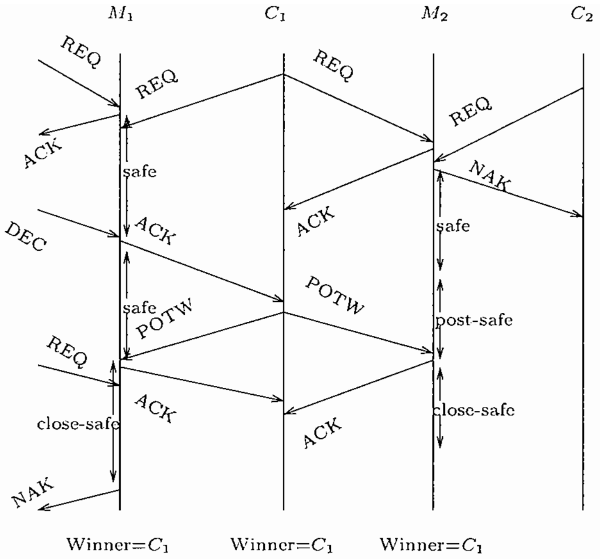

Figure 6: Illustration (example 1) of the second phase of the asynchronous algorithm; C_1 is the final winner. Intervals between message arrivals and departures not drawn to scale.

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 2τ 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.

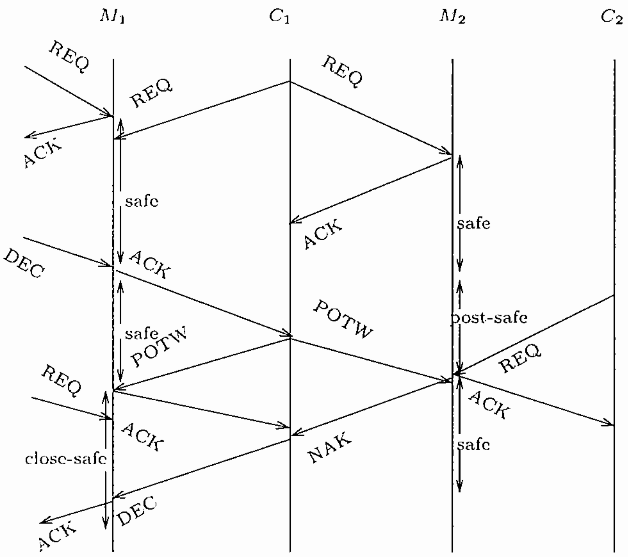

Figure 7: An example (example 2) of the second phase of the asynchronous algorithm; intervals between message arrivals and departures not drawn to scale.

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

Theorem 6 *The probability of having at least one contender at the end of the first phase is at least* $1 - \epsilon$, *i.e.,* $Pr(Y_w > 0) \geq 1 - \epsilon$, *where* $\epsilon \longrightarrow 0$ *as* $n \longrightarrow \infty$.

Proof: The proof follows from Theorem 2. Contenders that do not collide at any mediator will always go through to the next round in the asyn-

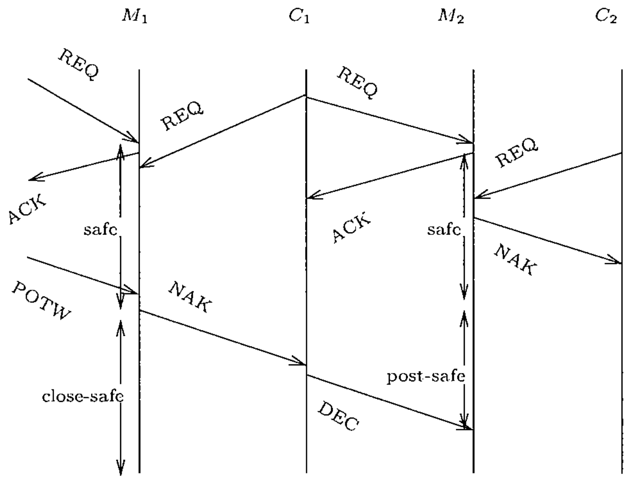

Figure 8: An example of the second phase of the asynchronous protocol (example 3). Intervals between message arrivals and departures not drawn to scale.

chronous case as in the synchronous case. Some contenders who collide with other contenders at a mediator may also proceed to the next round, unlike the synchronous case. Consequently, at the end of the first phase, the number of contenders in the asynchronous case is at least as large as the number of contenders in the synchronous case. From Theorem 2, the proof of this Theorem follows. □

Theorem 7 *The number of rounds in the protocol is $O(\log n)$.*

Proof: Let X_j be a random variable indicating the number of contenders participating in round j. In the asynchronous protocol, the contenders may potentially arrive at distinct points of time. If an ordering on the arrival time of a contender at round j is considered, then every contender ρ_i has a rank r_i in the ordering, where $0 \leq r_i \leq |\Phi_j| - 1$. For a contender ρ_i to proceed to round $j+1$, it should not collide at a mediator with any other contender whose rank is less than r_i. (In the synchronous case, all contenders involved in a collision do not proceed to the next round). The expected number of contenders in round

$j+1$ is given by:

$$
\begin{aligned}
E[X_{j+1}] &= \sum_{i=1}^{|\Phi_j|}(1-\frac{\varphi_j}{n})^{\varphi_j(i-1)} \approx \sum_{i=1}^{|\Phi_j|} e^{-\frac{\varphi_j^2(i-1)}{n}} \\
&= \sum_{t=0}^{|\Phi_j|-1} e^{-\frac{\alpha^i t}{n}} \text{ where } t = i-1 \\
&= 1 + \sum_{t=1}^{|\Phi_j|-1} e^{-\frac{\alpha^j t}{n}} \qquad (5)
\end{aligned}
$$

Since $e^{-\frac{\alpha^j t}{n}}$ is a montonically decreasing function, we can approximate (5) by the following integral:

$$
\begin{aligned}
E[X_{j+1}] &\leq 1 + \int_0^{|\Phi_j|-1} e^{-\frac{\alpha^j t}{n}} dt \\
&\leq \frac{n}{\alpha^j}
\end{aligned}
$$

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

Theorem 8 *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:

$$
N = E[\sum_{j=1}^{w-1} X_j \varphi_j] = \sum_{j=1}^{w-1} E[X_j]\varphi_j < n\alpha \sum_{j=1}^{w-1} \frac{1}{\alpha^{\frac{j}{2}}} \qquad (6)
$$

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

Theorem 9 *It takes a maximum of 7τ 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 τ 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 3τ units (or in the worst case, a negative response after 3τ units, if the earlier contender sent a potential winner). The response from the mediator to the contender can take at most τ units. Therefore, a contender need not wait for more than 5τ 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 τ units and the response from the mediator takes τ units. By accounting for the time taken for all the messages, a contender need not wait for more than 7τ units in the worst case to know whether it is a winner or not. □

Theorem 10 *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 7) 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 2τ time units for communication with the mediators. By Theorem 9, a contender will not take more than 7τ units in the second phase. Hence, a contender will know whether it is a winner or not in $O(\tau \log n)$. □

Theorem 11 *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. By Theorem 3, 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 explain and analyze the impact of failures of mediators in our synchronous protocol. Subsequently, we provide a generalization of this to the asynchronous case. We assume fail stop failures, i.e., a process can stop responding at any point of time. There are two scenarios under which a process can fail: before responding to contender(s) or after responding to contender(s). If the mediator fails after responding to contender(s), it means that the mediator has already arbitrated and does not pose a problem to the correct functioning of the algorithm. Consequently, we address the issue of mediators failing before they have sent any message to the contender(s).

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 such a 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 will affect 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 preserve the message and round complexity bounds. We adopt the second approach and provide bounds on the failure probability of processes such that the characteristics of the protocol are still maintained (as in the case of non-failures).

Theorem 12 *In the first phase, if ϵ is the failure probability of any process, $|\Phi_j|$ is the number of contenders in round j of the protocol, φ_j is the number of mediators, and n is the total number of processes, then for $\epsilon < \frac{1}{\varphi_j}$ the decrease in the number of contenders proceeding to subsequent rounds is bounded.*

Proof: Using the same definition for X_{ij} as in Theorem 1. the expectation of X_{ij} is given by

$$\begin{aligned} E[X_{ij}] &= Pr(X_{ij} = 1) = \left(1 - \frac{\varphi_j}{n}\right)^{\varphi_j(|\Phi_j|-1)+n\epsilon} \qquad (7) \\ &\approx e^{-\frac{\varphi_j^2(|\Phi_j|-1)}{n} - \varphi_j \epsilon} \end{aligned}$$

The difference between the expectation of X_{ij} when compared to Theorem 1 is the addition of $n\epsilon$ in (7). 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 $\frac{|\Phi_j|}{2e^{\varphi_j \epsilon}}$. When $\varphi_j \epsilon < 1$, we have between $\frac{|\Phi_j|}{2e}$ and $\frac{|\Phi_j|}{2}$ contenders proceeding to round $j+1$ on an average. Therefore, when $\epsilon < \frac{1}{\varphi_j}$, it limits the decrease in the number of contenders proceeding to subsequent rounds. □

To account for the reduction in number of contenders when compared to the non-failure case, we reduce the number of rounds that needs to be performed in the protocol to sustain the value of C_n, which in turn provides us with high probability of having at least one contender at the end of the first phase. When the initial number of contenders is less than C_n, the contenders proceed directly to the second phase in the presence of failures. This is somewhat different from the case in which no failures are present. A similar analysis holds good for the asynchronous case.

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. The additional number of messages sent is given by $(\sqrt{n \ln n})\epsilon$, where ϵ is the failure probability of any process. Given the bounds for ϵ for the first phase, the message complexity of the second phase is maintained. Generalizing this to the asynchronous protocol, the contender gets positive responses from all the $\sqrt{n \ln n}$ mediators to which a potential winner message was sent. If the contender receives positive responses from fewer mediators and does not receive at least a single negative response, it can assume the presence of failures and pick more mediators to account for the failed mediators.

6 Applications and Relevance to Current Distributed Systems

As mentioned before, in distributed systems with failures, probabilistic approaches to problems such as leader election and mutual exclusion are necessary, since deterministic protocols for these problems do not exist. On the flip side, in many applications, it is enough to guarantee probabilistic (as opposed to absolute) mutual exclusion. One such application, that forms our larger research and development goal, is a versioning-based distributed file system called Plethora [5]. 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 nodes 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. 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 [13, 2, 24]. In [13], for example, 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. Mayank *et al.* [2] propose an estimation scheme based on the birthday paradox [21][page 45]. A peer estimates the network size by sending a message on a random walk and using the hop count when the message returns to the peer. It is shown that it takes approximately $\sqrt{n}$ hops for a message to return to its sender. Psaltoulis *et al.* [24] propose a network size estimation algorithm for large and dynamic networks by using sampling techniques.

In our protocol, when the processes need to send messages to random processes in the network, they can perform a random walk to the required number of processes and use them as mediators. In [9], Gkantsidis *et al.* provide a method to perform uniform sampling through random walks in expander graphs. The Metropolis-Hastings algorithm [20, 12, 3] provides a method to hasten the mixing time of a random walk to reach a stationary distribution and can be applied to different graphs. These results provide the needed algorithms for uniformly selecting mediators.

The number of contenders is generally unknown in a large scale distributed system. Each contender can assume a worst case scenario of n other contenders participating and send $\sqrt{2^{j-1}\ln 2}$ messages in the jth round. Even in this worst case scenario, the message and round complexities are still preserved. An improvement of the algorithm presented in Section 3 would be to send request messages to $\varphi_j - \sum_{l=1}^{j-1}\varphi_k$ 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 φ_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.

7 Related Work

In [19], Malkhi *et al.* show, based on the birthday paradox, that in a system with n nodes, any two quorums of size $c\sqrt{n}$, where c is a constant and quorum members are picked uniformly at random, intersect with probability $\frac{1}{c^2}$. If c is set to $\sqrt{\ln n}$, quorum intersections are guaranteed with high probability. A direct extension of the probabilistic quorum algorithm to our problem would result in $O(n\sqrt{n\ln n})$ total 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 nodes 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. Also. we provide a protocol for the asynchronous version of the protocol.

In [18]. Maekawa proposes a $\sqrt{n}$ deterministic algorithm for mutual exclusion in distributed systems. The distributed sites (in our case. processes) 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. If the algorithm is trivially extended to solve the problem when $O(n)$ contenders are present in the system. it results in $O(n\sqrt{n})$ message complexity and a single round. In contrast to Maekawa's algorithm. we do not require the distributed processes to be arranged in a grid and thus do not require the elements of a mediator set to belong to the same row/column. Furthermore. we provide an efficient leader election algorithm with respect to message complexity at the cost of the number of rounds and low probability of error in correctness.

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 sub optimal when compared to the algorithm presented in this paper.

Mullender and Vitanyi propose a "distributed match-making" problem in [22] 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 nodes 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.

Gupta *et al.* [11] propose a probabilistic leader election protocol for large groups. The complexity of the protocol is $O(1)$ multicast messages (or $O(n)$ unicast messages. where n is the size of the group size). The success of the algorithm is guaranteed only with high *constant* probability. Our algorithm does not make any assumptions about the degree of the nodes; the success of the algorithm is also guaranteed *w.h.p.* and the message complexity is $O(n)$.

Schooler *et al.* [28] study two variants of the algorithm in the context of a multicast – leader election. where the processes announce the leader immediately (LE-A). and its variant in which the processes postpone the announcement to reduce collisions (LE-S). The authors analyze these variants using different metrics including delay and message overhead, among others. In our paper, we adopt a different approach to leader election instead of using multicast techniques. We also analyze our algorithms with respect to message overhead and the time taken for a leader to get elected in the system.

There has been significant work in developing mutual exclusion algorithms for shared memory models [15, 14]. These could be extended for purposes of leader election. In [15], 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 [14], Young 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.

8 Conclusion

In this paper, we design 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 the best of 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 provide a rigorous analysis of the correctness of the algorithm (in both the cases) and bounds on the number of messages and the number of rounds. We also examine the impact of failures on the performance and correctness of our algorithms.

References

[1] D. Agrawal, O. Egecioglu, and A.E. Abbadi. Billiard quorums on the grid. In *Information Processing Letters, 64*, pages 9–16, 1997.

[2] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical Report, Computer Science Department, Stanford University, 2003.

[3] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing markov chain on a graph. In *SIAM Review (to appear)*.

[4] I. Cidon and O. Mokryn. Propagation and leader election in a multihop broadcast environment. In *Proceedings of 12th International Symposium on Distributed Computing (DISC 1998)*, pages 104–118, Andros, Greece,.

[5] R.A. Ferreira, A. Grama, and S. Jagannathan. Plethora: A Locality Enhancing Peer-to-Peer Network. In *Journal of Parallel and Distributed Computing, (revised version submitted)*, 2004.

[6] C. Fetzer and F. Cristian. A highly available local leader election service. In *IEEE Transactions on Software Engineering, Vol. 25, No. 5*, pages 603–618, Sept 1999.

[7] M.J. Fischer, N.A. Lynch, and M.S.Paterson. Impossibility of distributed consensus with one faulty process. In *Journal of the ACM, Vol. 32, No. 2*, pages 374–382, Apr 1985.

[8] H. Garcia-Molina. Elections in a distributed computing system. In *IEEE Transactions on Computing, Vol. C-31, No. 1*, pages 48–59, 1982.

[9] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of IEEE INFOCOM, 2004*, Hong Kong, March 2004.

[10] Gnutella. http://gnutella.wego.com/.

[11] I. Gupta, R. van Renesse, and P. Birman. A Probabilistically Correct Leader Election Protocol for Large Groups. In *Proceedings of The 14th International Symposium on Distributed Computing (DISC 2000)*, Toledo, Spain, October 2000.

[12] W. Hastings. Monte carlo sampling methods using markov chains and their applications. In *Biometrika, 57*, pages 97–109, 1970.

[13] K. Horowitz and D. Malkhi. Estimating network size from local information. In *The Information Processing Letters journal 88(5)*, pages 237–243, December 2003.

[14] Y. Joung. Asynchronous group mutual exclusion. In *Distributed Computing (13)*, page 189.

[15] E. Kushilevitz and M.O. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the eleventh annual ACM symposium on Principles of distributed computing (PODC 1992)*, pages 275–283, Vancover, Canada, 1992.

[16] S. Lin, Q. Lian, M. Chen, and Z. Zhang. A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems. In *The Third International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, San Diego, CA, February 2004.

[17] Nancy A. Lynch. In *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[18] M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. In *ACM Transactions on Computer Systems*, pages 145–159, May 1985.

[19] D. Malkhi, M. Reiter, and R. Wright. Probabilistic quorum systems. In *Proceedings of the 16th Annual ACM Symposium on the Principles of Distributed Computing (PODC 97)*, pages 267-273, Santa Barbara, CA, August 1997.

[20] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. In *J. Chem. Phys., Vol. 21*, pages 1087–1092, 1953.

[21] R. Motwani and P. Raghavan. In *Randomized Algorithms*. Cambridge University Press, 1995.

[22] S.J. Mullender and P.M.B. Vitanyi. Distributed match-making. In *Algorithmica, 3*, pages 367–391, 1988.

[23] D. Peleg. Time-optimal leader election in general networks. In *Journal of Parallel and Distributed Computing, Vol. 8*, pages 96–99, 1990.

[24] D. Psaltoulis, D. Kostoulas, I. Gupta, K. Birman, and A. Demers. Practical algorithms for size estimation in large and dynamic groups. Submitted for publication and available at http://www.cs.cornell.edu/Info/Projects/Springlass/.

[25] M.O. Rabin. N-process mutual exclusion with bounded waiting by $4 \log_2 N$ valued shared variable. In *JCSS, Vol. 25(1)*, pages 66–75, 1982.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 247–254, San Diego, CA, August 2001.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 247–254, San Diego, CA, August 2001.

[28] E.M. Schooler, R. Manohar, and K.M Chandy. An analysis of leader election for multicast groups. In *Technical Report, ATT Labs-Research, Menlo Park, CA*, Feb 2002.

[29] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, San Diego, CA, August 2001.

[30] A.S. Tanenbaum and M.V. Steen. In *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.

[31] Gerard Tel. In *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.