Fair Leader Election by Randomized Voting

Siddhartha Brahma[*], Sandeep Macharla[†], Sudebkumar Prasant Pal[‡]
Sudhir kumar Singh[§]

Abstract

Leader election is a fundamental problem in distributed computing where a unique node from a set of nodes declares itself as the leader (a distinguished state). We define a notion of fairness in the leader election problem and show that existing deterministic algorithms based on comparisons of identifiers fail to achieve fairness. We demonstrate how fairness can be achieved through randomization and propose new leader election algorithms based on randomized voting. We separate the task of fair leader election on unidirectional rings into two subtasks: *attrition* and *solitude detection* following [8]. We show that tight interleaving of these two procedures achieves fairness in leader election on asynchronous, anonymous, unidirectional rings using an expected $O(n \log n)$ bits of communication, in expected $O(\log n)$ rounds. (Here, n is the number of nodes in the ring.) This matches the performance of the optimal algorithm of Abrahamson *et al.* [10]. Similar algorithms are presented for electing a fair leader in other models.

Keywords: leader election, randomization, fairness, voting.

1 Introduction

Leader election is a fundamental problem in distributed computing. Its solution forms a building block for many involved distributed computations. The problem is to designate by consensus a unique node from the collection of nodes that forms a distributed system [11, 2]. All nodes in the distributed system should agree upon one among them as the leader. So, leader election causes a unique node, from among a collection of nodes, to enter a distinguished final state [5]. The importance of the leader election problem stems from the fact that many distributed computations, even though they are to be computed by all participating nodes, require a special node. It is fundamental to many complicated distributed algorithms.

The leader election problem can be defined for various models. Usually the ring topology is considered as the standard for the design and analysis of leader election algorithms; the symmetry of the ring topology makes the task of leader election non-trivial. A unidirectional ring is a sequence $P_1, P_2, \cdots, P_n$ of n nodes or processors, where each node sends messages to P_{i+1} and receives messages from P_{i-1}, the subscripts being modulo n. There are several variants of this basic model. Communication between nodes may be synchronous or asynchronous; nodes may have unique identifiers (*asymmetric rings*) or may be

[*] Dept. of Computer Sc. and Engg., IIT Kharagpur
[†] Dept. of Computer Sc. and Engg., IIT Kharagpur, now with Mentor Graphics, Hyderabad, India
[‡] Dept. of Computer Sc. and Engg., IIT Kharagpur (communicating author) email spp@cse.iitkgp.ernet.in
[§] Dept. of Mathematics, IIT Kharagpur

indistinguishable (*symmetric or anonymous rings*); nodes may or may not know the size of the ring at the start of the computation. The complexity of a leader election algorithm is usually measured in terms of the number of messages communicated between the nodes, the total number of bits communicated and the time taken to elect the leader.

There are several deterministic algorithms for leader election in unidirectional rings. Several deterministic algorithms have been developed for finding a leader in an asynchronous, unidirectional, asymmetric ring networks, (see Lelann [11], Peterson [12], Chang and Roberts [2], Hirschberger and Sinclair [7], Burns [9], Dolev [3], Franklin [4]). The problem is simpler for the asymmetric case as compared to that in the symmetric case; the node with the maximum identifier may serve as the leader in the assymetric case. Indeed, all the above algorithms are based on selecting the node with the maximum identifier as the leader and therefore these are also called *e*xtrema-finding algorithms.

Angluin [1], established that deterministic algorithms fail to elect a leader if nodes do not have unique identifiers, even if the number of nodes n is known a priori to all the nodes. This limitation was overcome by the use of randomization by Itai and Rodeh [8]. They were also the first to point out the correspondence between leader election and the two tasks - *attrition* and *solitude detection*. Abrahamson *et al.* [10], gave another randomized algorithm for the same problem with the same message complexity as that of Itai and Rodeh, but with an improved bit complexity by tightly interleaving the solutions of the attrition and solitude detection tasks.

In all the deterministic algorithms mentioned above, given the initial configuration of the nodes, every time the same node is elected as the leader (eg. the person with the maximum identifier). This is somewhat *unfair*, as all the nodes do not get a fair chance to become the leader. We define a notion of fairness, in which all nodes have an equal probability of becoming the leader. With the above notion of fairness, we develop randomized algorithms for asynchronous ring networks that are *fair*. The procedure has an implicit round structure and nodes are elected to progress to the next round by *randomized voting*. Leader election is achieved by the tight interleaving of the solitude detection and attrition procedures.

The remaining paper is organized as follows. In section (2) we define the notion of fairness in leader election and present our algorithm for fair leader election in asynchronous, anonymous, unidirectional ring networks. In section (3) we extend the above algorithm for variants of the above network model. Finally in section (4), we conclude with a few remarks about future research directions.

2 Fair leader election on rings

We define a leader election protocol to be *fair* if all the nodes have equal probability of getting elected as the leader. If there are c contenders initially, each of them must have probability $1/c$ of being elected leader. Our eader election algorithm proceeds in stages. Suppose the ring is in a certain configuration CF_0 at a particular stage. Let $CF_1, CF_2, \cdots, CF_T$ be the configurations that can result from CF_0 in the next stage. Suppose the probability of occurence of each CF_i , $1 \leq i \leq T$ is $1/T$. If this holds for any intermediate configuration CF_0 of the leader election algorithm, then it is easy to see that the overall algorithm will be fair. In what follows, we achieve fairness by exploiting this fact. Any leader election algorithm should finally accomplish the following two tasks (i) eliminate all but one node from contending and, (ii) confirm that only one node is contending. As mentioned above, there is a strong relationship between leader election and the two tasks of attrition and solitude detection. The two procedures are defined as follows [5] :

Attrition Procedure

A procedure solves the *attrition problem* when initiated by a set of nodes if it never makes all candidates noncontenders and, it takes all but exactly one of these candidates into a permanent state of noncontention with probabbility 1.

Solitude Detection Procedure

A procedure solves the *solitude detection* problem when initiated by a set of nodes if it terminates with probability 1, giving a 'yes' answer upon termination when there is exactly one initiator and a 'no' answer when there are more than one initiators. In the latter case, all initiators are set to a 'no' state.

Typically, the attrition procedure does not terminate. It goes into an infinite loop. Solitude detection is required to break this infinite loop. We approach the problem of attrition using a process of *voting*. Our attrition procedure has an implicit round structure. As we want to ensure fairness in the leader election algorithm, the idea of a node randomly casting a vote in favour of another node is natural. Thus, instead of eliminating nodes from contention, we elect nodes for contending in the next round. This is done by a randomized voting protocol which also ensures fairness in the election process. In this section we assume an anonymous (symmetric), asynchronous and unidirectional ring network model. Each node has an infinite FIFO buffer where incoming messages are stored and processed one by one. In the next three subsections, we explain the attrition procedure, the solitude detection procedure and the overall leader election algorithm.

2.1 The Attrition Procedure

Our attrition procedure is based on the idea of one node sending its vote to another node based on a random number generated by itself. The attrition procedure has an implicit round structure. The contender nodes are called *active* and the rest are called *p*assive. Let the number of nodes in the ring be n and let the number of active nodes in the beginning be c ($c \leq n$). In each round, an active node generates a random number r and sends its vote to a node which is r hops away. Clearly, r is the number of active nodes between the sender and the receiver, including the receiver. The intermediate passive nodes simply relay votes. The nodes that receive a non-empty set of votes in the current round, survive to become contenders in the next round. Note that the part of the leader elction algorithm that dictates which node will ultimately become the leader is the attrition procedure. Since, voting is based on random numbers and passive nodes only relay votes, all the active nodes have an equal chance of going to the next round. So, all the configurations that are possible in the next round are equally likely in a particular round of the attrition procedure, thereby ensuring fairness in the attrition procedure. This also ensures fairness in leader election. By setting the range of values that r can take, the attrition procedure can be varied. We consider the following two cases :

(i) **Procedure I**

r lies in the interval $[0, A)$, where A is the number of active nodes in the beginning of a particular round.

(i) **Procedure II**

r lies in the interval $[0, K)$, where K is a positive integer constant.

The implicit round structure in the attrition procedure is enforced as follows. Since the network is asynchronous, there must be a way for the nodes to detect the end of a round. Let $\beta_0, \beta_1, \cdots \beta_{n-1}$ be the n nodes in the ring. Let $\alpha_0, \alpha_1, \cdots \alpha_{t-1}$ be the t contenders ($t \geq 2$) in an arbitrary intermediate round j. Every node α_i has a set of potential voters. α_i can detect the end of a round if it has the knowledge that all its potential voters have already, either voted for it or for somebody else. Every node α_i maintains two variables z_{i_1} and z_{i_2} which are initialized to zero in the beginning of each round. z_{i_1} keeps track of the number of votes obtained in a particular round and z_{i_2} maintains the number potential voters for α_i that have already voted in that particular round. Every active node generates a random number and sends a message $< x, y >$ to its neighbor, where x is the random number generated and $y = A$ or $y = K$

for **Procedure I** and **II** respectively. An active node checks whether $x = 0$. If x is zero, then it means the vote has gone to that node. Then, if $y > 1$ the message $< 0, y-1 >$ is forwarded and z_{i_1} and z_{i_2} are incremented, otherwise the message is no longer forwarded. If x is non-zero, then z_{i_2} is incremented and the message $< x-1, y-1 >$ is forwarded. It is easy to see that a node can be sure that a round has ended when z_{i_2} is A or K, for **Procedure I** and **II** respectively. Thus, in each round a contender gives its vote and waits until z_{i_2} becomes A or K, whence it checks the value of z_{i_1}. If it is nonzero it remains active in the next round otherwise it becomes passive. If we assume that there is no loss of messages in the network, then the above attrition procedure works correctly in the sense that a node that should have been active in a particular round is surely active and a node that should have become passive in a particular round is never active, as shown in the following lemma.

Lemma 2.1 *If there is no loss of messages in the ring, every round in the above attrition procedure is completed correctly.*

Proof Let us assume that the attrition procedure has worked for p rounds correctly, where $p \geq 0$. Let us consider the FIFO buffer corresponding to the active node α_{i+1} for some i. Any message in this buffer comes from the previous active node α_i. Since α_i has completed round p successfully, it will send all the messages of round $p+1$ to α_{i+1} before going into the next round. This implies that in the buffer of α_{i+1} there will be no messages of any other round in between the messages of round p. Since, the node we are considering is arbitrary, this is true for all nodes. It is easy to see that the attrition procedure may not work correctly only when the messages from two rounds overlap in the buffer. Since this does not happen in round $p+1$, it will also be completed correctly. By induction on the number of rounds, all the rounds are completed correctly and the lemma is proved. ∎

2.1.1 Procedure I

In this case, in the beginning of each round every node is required to know the number of active contenders. The task of discovering the number of active nodes and informing all the active nodes about the number is left to the solitude detection procedure which is interleaved with the attrition procedure in the leader election algorithm and is explained later.

Correctness Proof

The correctness of the attrition procedure in the sense that all the rounds are completed correctly has been proved in **Lemma** 2.1. Also, since the votes are sent only to active nodes, in each round there will be at least one node which receives a vote, and hence will remain a contender. Therefore, at all intermediate stages there is at least one contender. Also, the probability that a node is eliminated in round j is η_1 where

$$\eta_1 = \left(1 - \frac{1}{t}\right)^t \tag{1}$$

Since $1/e > \eta_1 \geq 1/4$ for $t \geq 2$, in each round there at least a non-zero constant probability of a contender getting eliminated and hence the number of contenders decreases to 1 with probability 1.

Complexity Analysis

The attrition procedure goes into an infinite loop when only one contender remains. This loop can broken by a suitable solitude verification algorithm. The complexity analysis of the attrition procedure therefore excludes the last solitude detection round. Consider an arbitrary round numbered j. Let X_j be a random variable denoting the number of contenders in round j. So, $X_1 = c$. Let Y_i, $0 \leq i < t$, be a random variable defined as $Y_i = 1$ if α_i is a contender in round $j+1$ and, 0 otherwise. Then, it is easy to see that $E(Y_i) = 1 - \eta_1$, where $t \geq 2$ and η_1 is defined as in (1) above. So, if $t \geq 2$,

$$E(X_{j+1}|X_j = t) = E\left(\sum_{i=0}^{t-1} Y_i\right) = \sum_{i=0}^{t-1} E(Y_i) = (1-\eta_1)t$$

Therefore,

$$E(X_{j+1}) = \sum_{t \geq 1} E(X_{j+1}|X_j = t) Pr(X_j = m)$$

Since $E(X_{j+1}|X_j = 1) = 1$, we have

$$\begin{aligned}
E(X_{j+1}) &= \sum_{t \geq 2} E(X_{j+1}|X_j = t) Pr(X_j = t) + Pr(X_j = 1) = \sum_{t \geq 2} ((1-\eta_1)t \cdot Pr(X_j = t)) + Pr(X_j = 1) \\
&= (1-\eta_1) E(X_j) + \eta_1 Pr(X_j = 1) \leq \frac{3}{4} E(X_j) + \eta_1 Pr(X_j = 1)
\end{aligned}$$

So, after an expected $\log_{4/3}(c)$ rounds of the attrition procedure, the number of contenders will be negligible (constant), and hence will contribute negligibly to the overall complexity. So, in an expected $O(\log n)$ rounds, the number of contenders converges to 1. The message complexity is derived as follows. Let M_j be a random variable representing the number of messages transferred in round j. Let N_i , $(0 \leq i \leq t-1)$, be a random variable denoting the number of messages transferred for the vote of α_i in round j. Let the number of passive nodes between α_i and $\alpha_{i+1 (\text{mod } t)}$ be $n_i - 1$. Then,

$$E(N_i) = \frac{1}{t} \sum_{l=0}^{t-2} \sum_{m=0}^{l} n_{i+l (\text{mod } t)} + n - n_{i+t-1 (\text{mod } t)}$$

Therefore,

$$\begin{aligned}
E(M_j|X_j = t) &= E(\sum_{i=0}^{t-1} N_i) = \sum_{i=0}^{t-1} E(N_i) \\
&= \frac{t-1}{2} \sum_{i=0}^{t-1} n_i + \sum_{i=0}^{t-1} n - n_{i+t-1 (\text{mod } t)} \\
&= \frac{3n(t-1)}{2}
\end{aligned}$$

Therefore,

$$\begin{aligned}
E(M_j) &= \sum_{t \geq 2} E(M_j|X_j = t) Pr(X_j = t) = \frac{3n}{2} \sum_{t \geq 2} (t-1) Pr(X_j = t) \\
&< \frac{3n}{2} \sum_{t \geq 1} t \cdot Pr(X_j = t) = \frac{3n}{2} E(X_j)
\end{aligned}$$

Let M be the random variable denoting the total number of messages required in the attrition procedure. Then,

$$\begin{aligned} E(M) &= E(\sum_{j\geq 1} M_j) = \sum_{j\geq 1} E(M_j) < \frac{3n}{2}\sum_{j\geq 1} E(X_j) \\ &\leq \frac{3n}{2}\left(\sum_{j\geq 1} E(X_1)\cdot\left(\frac{3}{4}\right)^j + d\right) \leq \frac{3n}{2}(4c+d) \end{aligned}$$

where, using the fact that $Pr(X_j = 1) \leq 1$ for all j,

$$d \leq \eta_1 \sum_{j\geq 0}\left(\frac{3}{4}\right)^j < 4\eta_1 < \frac{4}{e} < 2 \qquad (2)$$

Since, each vote can be implemented using $O(\log c)$ bits, this attrition procedure converges in expected $O(\log c)$ rounds using an expected $O(nc\log c)$ bits. The structure of the procedure ensures that it leads to fair leader election. Although the expected number of rounds required is only $O(\log c)$, the expected bit complexity is $O(nc\log c)$, which is far from optimal. Next, we show that the attrition procedure in **Procedure II** has an optimal bit complexity.

2.1.2 Procedure II

There is no need here for the active nodes to know the total number of active nodes in the ring.

Correctness Proof

Since the votes are sent only to active nodes, there will be at least one node which receives a vote in each round, thereby ensuring a contender. Also, the probability that a node is eliminated from contention in a particular round is η_2 where

$$\eta_2 = \left(1 - \frac{q+1}{K}\right)^r \left(1 - \frac{q}{K}\right)^{t-r}$$

where $K = t\cdot q + r$. Since,there is at least a non-zero constant probability of a contender getting eliminated in each round, the number of contenders decreases to 1 with probability 1.

Complexity Analysis

Let X_j and Y_i be a random variable defined as in Case(i). It is easy to see that $E(Y_i) = 1 - \eta_2$ where $t \geq 2$ and η_2 is as defined above. So, if $t \geq 2$, $E(X_{j+1}|X_j = t) = E\left(\sum_{i=1}^{t} Y_i\right) = \sum_{i=1}^{t} E(Y_i) = (1-\eta_2)t$. Therefore,

$$E(X_{j+1}) = \sum_{t\geq 1} E(X_{j+1}|X_j = t)Pr(X_j = m)$$

Since $E(X_{j+1}|X_j = 1) = 1$, we have

$$\begin{aligned} E(X_{j+1}) &= \sum_{t\geq 2} E(X_{j+1}|X_j = t)Pr(X_j = m) + Pr(X_j = 1) \\ &= \sum_{t\geq 2} ((1-\eta)t\cdot Pr(X_j = t)) + Pr(X_j = 1) \\ &= (1-\eta_2)E(X_j) + \eta_2 Pr(X_j = 1) \end{aligned}$$

So, after an expected $\log_{1/1-\eta_2}(c)$ rounds of the attrition procedure, the number of contenders will be negligible (constant), and will contribute negligibly to the overall complexity. The bit complexity for each round of attrition is $O(nK \log K)$. Therefore, we have the following lemma.

Lemma 2.2 Procedure II *reduces the number of contenders from c to a negligible number (a constant) in expected* $O(\log_{1/1-\eta_2})(c)$ *rounds, with a bit complexity of* $O((K \log K)n \log c)$.

It is interesting to note that the analysis of the attrition procedures could have been done in a different, albeit difficult way. Let $P(r, s)$ be the probability of s contenders remaining active after a round that had r contenders. For **Procedure I**,

$$P(r,s) = \binom{r}{r-s} \sum_{i=0}^{s-1} (-1)^i \binom{s}{i} (s-i)^r \tag{3}$$

and for **Procedure II**,

$$P(r,s) = \binom{r}{r-s} \sum_{i=0}^{s-1} (-1)^i \binom{s}{i} (s-i)^K \tag{4}$$

Using the expression for $P(r, s)$, one can derive all the required bounds in a way similar to the analysis of M. Hofri in [8].

2.2 The Solitude Detection Procedure

A solitude detection procedure determines whether there is a lone active node in a ring. Such a node can be declared leader. A suitable solitude detection procedure for the attrition **Procedure I** is as follows. A particular node initiates the solitude procedure. Without loss of generality, let us assume that α_0 initiates the procedure. It initaializes a counter F to 0. If it is active it increments the counter by 1, otherwise, it does nothing. The counter is forwarded to the the next node. Each active node increments F and forwards it. All passive nodes simply forward the counter F. When the counter returns back to α_0, the value of F is the number of active nodes in the ring. This value of F is broadcast to all the nodes, whence every node updates its knowledge of the number of the active nodes. If $F = 1$, the only active node declares itself as the leader. This procedure requires $O(n \log c)$ bits of communication and not only achieves solitude detection, but also ensures that all the active nodes know the number of active nodes in the ring.

A solitude detection algorithm suitable for use with **Procedure II** is given in [8]. We reproduce it here for the sake of completeness. All the nodes β_i maintain a counter c_i initialized to zero. The variable $d_i > 0$ of node β_i, where $1 \leq i \leq n$, denotes the distance from node β_i from its preceding active node. The algorithm maintains the invariant: node β_i has received j bits then if and only if $c_i = d_i \bmod 2^j$. The actual number of nodes in relay state preceding node β_i is d_i. This is calculated as the bits are received. For an active node to conclude that it is the only node surviving, there must be $n - 1$ nodes in the relay state preceding this active node. For maintaining the invariant, the active nodes first send 0. Thereafter, all nodes alternately receive and send bits. If node β_i is passive, then node β_i will send the j^{th} low order bit of d_i as its j^{th} message. The active nodes continue to send 0. Whenever a node k receives its j^{th} message, it computes the first j bits of d_k and updates the value of c_k. The j^{th} message received by node k contains the j^{th} bit of $d_k - 1$ because it is sent by the preceding neighbor to node k. In this algorithm, each message is of constant number of bits. And $O(n)$ messages are communicated in each step. If there is exactly one active node, the algorithm confirms the survival of this single node in $\log n$ steps. Solitude detection can be done using $O(n \log n)$ messages of constant length. So the bit complexity of this solitude detection procedure is $O(n \log n)$ bits.

2.3 The Leader Election Algorithm

In the previous two subsections we have proposed two attrition procedures and solitude detection procedures that can work with them. The structure of the attrition procedure ensures that the leader election in which they will be used is fair. Since **Procedure I** leads to an algorithm which has high bit complexity that is not optimal, we only describe the leader election algorithm for **Procedure II**. The leader election algorithm is obtained by interleaving the attrition procedure with the solitude detection procedure. There is a trivial way to interleave the two procedures coarsely. In this method the attrition procedure is run until with very high probability the number of contenders has reduced to one. This is followed by a round of solitude detection. In most cases solitude will be confirmed. In the rare case of non-solitude, the attrition procedure will be restarted. These two procedures will be repeated until solitude is confirmed. It can be shown by an analysis similar to that given in [6] that even this naive approach achieves an $O(n \log n)$ expected bit complexity.

The above leader election algorithm has a deficiency that it cannot be adapted to work in a ring with each node having a distinct identifier but no knowledge of the ring size or the number of contenders. This deficiency can be overcome by a tighter interleaving as shown in [5]. The new procedure also has an added advantage of stopping early if the attrition procedure proceeds more quickly than expected. We discuss the main points of the algorithm that has been adapted to work with our attrition procedure.

The second solitude detection procedure as outlined in section (2.2) basically tries to maintain gap information between two adjacent active nodes. In a leader election algorithm in which attrition and solitude detection is tightly interleaved, the gaps keep on changing from one round to another. This problem is eliminated by the use of restart flags in the messages. In each round, each contender sends a message consisisting of three parts - an attrition part (as described in section (2.1)), a solitude detection part (as described in section (2.2)) and a solitude restart flag initialized to FALSE, to the next contender. If a message arrives at a passive node α_i that was active in the previous round, then all nodes following it upto and including the next active node have gap information that is no longer valid. β_i notifies this by setting the restart flag to TRUE so that the succeeding nodes reinitialize their solitude detection variables. Since β_i was active in the previous round it had received information about the gap between itself and the preceeding active node. It can send the first bit of this unchanged gap information as the solitude detection bit.

Each node maintains the following three variables for solitude detection apart from the variables for the attrition procedure.

1. j_i: count of the number of messages received containing correct gap information.

2. c_i: counter variable containing the value d_i mod 2^j.

3. o_i: position of the outgoing solitude detection bit.

All nodes initialize their local variables j_i, c_i and o_i to 0. In each round, an active node generates a random number and sends its vote in the form of a message as mentioned above. Depending on the message an active and a passive node act in the following ways :

Active Node : The attrition part of the message is handled in the way described in section (2.1). If the restart flag is TRUE, it reinitializes its local variables j_i, c_i and o_i to 0 and resets the restart flag to FALSE. Based on the received solitude detection bit, it updates counter variable c_i to maintain c_i=d_i mod 2^j and increments j_i and o_i. Now it forwards a message containing the modified attrition part, the solitude detection bit and restart bit set to FALSE to its neighbor.

Passive Node : In each round, a passive node β_i receives messages that it has to forward. If the restart bit in the received message is set, it will reinitialize its local variables j_i, c_i and o_i to 0. Based on the solitude detection bit, it updates its counter variable c_i to maintain c_i=d_i mod 2^j and increments j_i and o_i. If the position variable o_i is one, β_i is a node that was active in the previous round, and as reasoned above it sets the restart flag to TRUE. It uses the first bit of the counter c_i as the new solitude detection bit. With the new values for the solitude detection bit and the restart flag and keeping the attrition part unchanged β_i forwards the message to its neighbor.

In our attrition scheme, in a particular round a node has to handle more than one message. Thus a node updates its solitude detection variables only once in a particular round. The solitude detection parts of all subsequent messages are ignored for that round.

3 Extensions to other models

The algorithm that has been discussed above works for an anonymous, asynchronous ring where the number of nodes n is known to every node. Even when n is known only within a factor of less than 2, i.e. $n \in [N, 2N-1]$ and N is known to every node, the solitude detection procedure for **Procedure II** can be modified to work correctly. There can be at most one gap of length N or more between neighboring contenders and if the gap is less than N, non-solitude is confirmed. If a particular node detects a gap greater than N it can confirm solitude by initiating a single round to check whether the next gap also remains the same. This modified procedure detects solitude when every node has the knowledge of N, where $n \in [N, 2N-1]$.

The leader election algorithm can also be adapted to a situation in which each node has a distinct identifier but no knowledge of the ring size or the number of contenders. This is achieved by a simpler solitude detection procedure. In the new solitude detection procedure, an active node verifies its solitude by confirming that the preceeding active node has the same identifier as itself. For this confirmation, each active node sends its identifier to the succeeding neighbor. If the node is passive, it simply forwards the identifier it has received. An active node checks the received identifier - if it is same as its own identifier, then it declares itself as the leader. Each round of solitude detection requires only $O(n)$ messages and each message can be at most m bits long, where m is the length of the longest identifier in the ring. Thus solitude can be verified using only $O(nm)$ bits. The leader election algorithm can easily be adapted to work with this new solitude detection procedure.

4 Conclusion

In this paper we have proposed a new notion of fairness in the leader election problem. We design algorithms for fair leader election amongst c contending nodes in an n-node unidirectional, asynchronous and anonymous ring ($c \leq n$) using randomized voting, achieveing optimal $O(n \log n)$ expected communication bit complexity in expected $O(\log c)$ rounds. Exploiting the fact that leader election can be achieved by interleaving an attrition procedure with a solitude detection procedure, we develop new algorithms for leader election in rings. We assume that the ring size n is known. We also show how to adapt our algorithms for slightly different unidirectional ring models.

References

[1] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93, 1980.

[2] E. Chang and R Roberts. An improved algorithm for decentralized extrema-finding in circularly configurations of processes. *Comm. ACM*, 22(5):281–283, May 1979.

[3] Michael Rodeh Danny Dolev, Maria M. Klawe. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *J. Algorithms*, 3(3):245–260.

[4] Wm. Randolph Franklin. On an improved algorithm for decentralized extrema finding in circular configurations of processors. *Comm. ACM*, 25(5):336–337.

[5] Lisa Higham. Simple randomized leader election with extensions.

[6] Lisa Higham. *Randomized Distributed Computing on Rings.* Phd thesis, University of British Columbia, 1988.

[7] D.S. Hirschberg and J.B Sinclair. Decentralized extrema-finding in circular configurations of processors. *Comm. ACM*, 23(2):627–628, November 1980.

[8] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. In *22nd Annual Symp. on Foundations of Comput. Sci.*, pages 150–158, 1981.

[9] J.E.Burns. A formal model for message passing systems. Technical Report TR-91, Indiana University, 1980.

[10] Rachel Gelbart Lisa Higham Karl Abrahamson, Andrew Adler and David Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 18(1):12–29, 1989.

[11] G LeLann. Distributed systems-towards a formal approach. In *Inform. Proc. 77*, pages 155–160, Amsterdam, 1977. North-Holland Pub. Co.

[12] Gary L. Peterson. An o(n log n) unidirectional distributed algorithm for the circular extrema problem. *ACM Transactions on Programming Languages and Systems*, 4(4):758–762, October 1982.