Optimal Algorithms for Probabilistic Solitude Detection On Anonymous Rings**

by

Karl Abrahamson*
Andrew Adler †
Lisa Higham #
David Kirkpatrick ǂ

Technical Report 90-3
January, 1990

* Computer Science Department
Washington State University
Pullman, WA 99164-1210
U.S.A.

† Department of Mathematics
ǂ Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5
Canada

# Computer Science Department
University of Calgary
Calgary, Alberta, T2N 1N4
Canada

** This research was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Killam Foundation.

Computer Science Department
University of British Columbia
Vancouver, B.C. V6T 1W5
Canada

Abstract

Probabilistic algorithms that err with probability at most $\epsilon \geq 0$ are developed for the Solitude Detection problem on anonymous asynchronous unidirectional rings. Solitude Detection requires that a nonempty set of distinguished processors determine whether or not there is only one distinguished processor. The algorithms transmit an optimal expected number of bits, to within a constant factor. Las Vegas and Monte Carlo algorithms that terminate both distributively and nondistributively are developed. Their bit complexities display a surprisingly rich dependence on the kind of algorithm and on the processors' knowledge of the size of the ring.

1 Introduction

The motivation for this work is to understand the dependence of the communication complexity of a problem in distributed computation on the processors' knowledge of the network over which they communicate. To constrain the question, we choose a simple network topology and a simple problem to study. The network is an asynchronous unidirectional ring. The problem, Solitude Detection, is a subproblem of the well-studied problem of Leader Election, and is one of the simplest distributed problems. A Boolean function closely related to Solitude Detection is among the easiest nontrivial Boolean functions to compute, in terms of expected bit complexity [4, 12]. Given a nonempty set of distinguished processors, called *contenders*, Solitude Detection requires that the processors determine whether or not there is exactly one contender. The relevant knowledge concerns the size of the ring. To isolate that knowledge for study, we assume that processors do not have distinct identifiers; that is, the ring is *anonymous.*

Many algorithms for Leader Election on rings have been proposed. Las Vegas algorithms have been suggested for anonymous rings [1, 2, 13] because, as Angluin observes [6], no deterministic solutions exist for anonymous rings. Suppose that an algorithm has knowledge of some approximation N of the actual ring size n. There is a Las Vegas Leader Election algorithm for rings of size n where $N/2 < n \leq N$ with expected bit (and message) complexity $O(n \log n)$ [2]. This algorithm works by interleaving solutions to the two subproblems Attrition and Solitude Detection. (Attrition is the problem of reducing the original number of contenders to exactly one contender). Furthermore, even with exact knowledge of ring size, $\Omega(n \log n)$ expected messages (and hence also bits) are required to elect a leader [3, 8, 9, 10, 12] on anonymous rings. This lower bound extends to the Attrition subproblem of leader election. However, Solitude Detection can be solved in $O(n)$ messages for $N/2 < n \leq N$, although $\Omega(n \log n)$ bits are still required for rings in this range [2]. Surprisingly, when ring size is known exactly, the expected bit complexity of Solitude Detection drops to $\Theta(n\sqrt{\log n})$ [4]. These results highlight the insensitivity of Attrition and the sensitivity of Solitude Detection to knowledge of ring size.

This paper and a companion paper extend the investigation of the expected bit complexity of Solitude Detection. Both Las Vegas algorithms, which never err, and Monte-Carlo algorithms, which err with low probability, are considered here. The processors' knowledge of ring size varies over the entire range of possibilities from no knowledge to exact knowledge. We also consider two types of termination. The usual notion of termination is *distributive termination*, in which a processor must, upon termination, reach an irrevocable conclusion. *Nondistributive termination*, a weaker notion of termination introduced by Itai and Rodeh [13], permits processors to reach tentative conclusions, which can be revoked upon receipt of further communication. Thus, a processor can base its termination on a condition — the cessation of message traffic — that it cannot directly detect.

This paper establishes upper bounds. The companion paper [5] establishes matching lower bounds. Together, they show that the expected bit complexity of Solitude Detection displays a surprisingly rich dependence on knowledge of ring size and type of algorithm employed. Tables summarizing the results of this paper and the companion paper can be found in the concluding section.

The remainder of this paper is organized as follows. We begin by considering the case where the ring size n is not known exactly. Section 2 presents a simple Solitude Detection algorithm upon which more sophisticated algorithms are built in subsequent sections. Section 3 develops some tools for constructing the more efficient algorithms. The tools provide estimates of the lengths of gaps between consecutive contenders. Section 4 describes improved Solitude Detection algorithms for approximately known ring size. Section 5 develops algorithms for the case where n is known exactly. These take advantage of divisibility properties of n to gain efficiency over prior algorithms. Finally, Section 6 ties together the results of this and its companion paper.

Throughout, "log" denotes $\log_2$ and "ln" denotes $\log_e$.

2 A Basic Approach

Messages travel counter-clockwise around the ring. Say that a processor receives messages from the left, and sends them to the right.

A straightforward idea is used by all of our randomized Solitude Detection algorithms. Each contender makes t independent coin tosses, and sends them, one at a time, to the next contender to the right, waiting to receive a coin toss before sending the next one. Noncontenders participate only by forwarding coin tosses. If there is only one contender, then it will receive the same sequence of coin tosses as it sends.

As soon as the bit received differs from that most recently sent, a contender sends an *alarm.* Alarms are forwarded by noncontenders and by any contender that has not already sent one. Transmission of an alarm causes a contender to abort the rest of the algorithm, and to conclude that it is not alone. A contender tentatively concludes (perhaps erroneously) that it is alone if it has sent and received t coin tosses without sending or receiving an alarm. Notice that a solitary contender always reaches the correct conclusion that it is alone.

The basic procedure is summarized in Figure 1. Each iteration of the loop is called a *round.*

Lemma 2.1 *Let the actual number of contenders be $c > 1$. When the basic procedure is run using t coin tosses,*

(a) the probability that all contenders conclude that they are alone is $2^{-t(c-1)}$,

(b) the probability that some contender concludes that it is alone is at most $c2^{-t\min(t,c)/2}$,

```
alone := true;
for $i$ := 1 to $t$
   $v$ := coin toss;
   send($v$);
   receive($w$);
   if $w \neq v$ then
      alone := false;
      send(alarm);
      stop
   end if
end for
```

Figure 1: The basic procedure for a contender

(c) the total expected number of bits transmitted is $O(n)$.

Proof: (a) In order for all contenders to conclude that they are alone, every contender must make the same i^{th} coin toss, for $i = 1, \ldots, t$. The probability of that occurring is $2^{-t(c-1)}$.

(b) A contender is *fooled for k flips* if its first k outputs match its first k inputs. Let Q_0, $Q_1, \ldots, Q_{c-1}$ denote the contenders in sequence around the ring.

If any one contender, say Q_j, is fooled for t flips (and so concludes that it is alone), then it must be the case that contender Q_{j-k} is fooled for $t-k$ flips, for $1 \leq k \leq t-1$, since otherwise Q_j would receive an alarm before its t^{th} flip. (Subscripts are implicitly reduced modulo c.) If $c \geq t$ then all of these flips are independent, and there are at least $t^2/2$ of them. On the other hand, if $c < t$ then at least $ct/2$ of these flips are independent. In either case $\Pr(Q_j$ is fooled for t flips$) < 2^{-t\min(t,c)/2}$, so the probability that *some* processor concludes that it is alone is at most $c2^{-t\min(t,c)/2}$.

(c) Since the coin tosses of adjacent contenders are independent, and a contender stops as soon as it receives a coin toss different from what it last sent, the expected number of bits transmitted by each contender is $O(1)$. ∎

It is not possible to detect solitude with distributive termination and probability of error bounded away from zero if processes know nothing about the ring size, as is now shown. Let

R be a ring of size n with exactly one contender. Suppose the contender concludes that it is alone after running some distributively terminating algorithm. Let h be the communication history of that contender, consisting of all of its input and output messages, interleaved as (input$_1$, output$_1$, input$_2$, ..., input$_k$, output$_k$). It is possible, by splicing together enough copies of R, to form a ring R' with many contenders such that the probability that more than one contender has a history of which h is an initial segment is arbitrarily close to one. Therefore, without any knowledge about ring size, it is impossible for a contender to halt and declare its solitude with any degree of certainty. Nondistributively terminating algorithms for this problem do exist, however, as shown in Theorem 2.2.

The basic procedure has some features that are shared by all of our algorithms. Say that a Solitude Detection algorithm has *one-sided error* if it cannot erroneously conclude that there are two or more contenders, when there is just one contender. Say that the algorithm is *one-sided linear* if the algorithm has expected bit complexity $O(n)$ when there are two or more contenders. An algorithm is *one-sided* if it has one-sided error and is one-sided linear.

Throughout, $0 < \epsilon < 1$ represents a parameter to a Monte Carlo algorithm. Such an algorithm is a correct Monte Carlo algorithm if its error probability is at most ϵ.

Theorem 2.2 *There is a nondistributively terminating one-sided Monte Carlo Solitude Detection algorithm with worst case bit complexity* $O\left(n\log(\frac{1}{\epsilon})\right)$ *for rings of any size, where* n *is the actual size of the ring.*

Proof: Run the basic procedure with $t = \left\lceil\log(\frac{1}{\epsilon})\right\rceil$, followed by

```
if an alarm arrives and alone = true then
   alone := false;
   send(alarm)
end if
```

This algorithm terminates nondistributively. When there is one contender, the algorithm answers correctly. By Lemma 2.1(a), the probability that no processor sends an alarm when there are at least two contenders is at most $2^{-t} \leq \epsilon$. If any processor sends an alarm then all processors eventually receive one and conclude that they are not alone. Lemma 2.1(c) provides the complexity bound when there are two or more contenders. ∎

Distributive termination is possible when the algorithm is given an upper bound C on the number of contenders.

Theorem 2.3 *There is a distributively terminating one-sided Monte Carlo Solitude Detection algorithm for all rings with at most C contenders with worst case bit complexity* $\mathrm{O}\left(n\sqrt{\log C} + n\log(\frac{1}{\epsilon})\right)$.

Proof: We can assume that $C \geq 2$. Use the basic procedure with $t = \left\lceil\sqrt{2\log C} + \log(\frac{1}{\epsilon})\right\rceil$. Let c be the actual number of contenders. Then $c2^{-t\min(t,c)/2} < \epsilon$. By Lemma 2.1(b), with probability at least $1-\epsilon$, all of the processors reach the correct conclusion after at most t coin tosses have been exchanged. So the algorithm has the desired properties. ∎

Theorem 2.3 at once gives an $\mathrm{O}\left(n\sqrt{\log N} + n\log(\frac{1}{\epsilon})\right)$ upper bound for the worst case bit complexity of Monte Carlo Solitude Detection when an upper bound N on the ring size is known by all processors. This upper bound can be sharpened to $\mathrm{O}\left(n\sqrt{\log(\frac{N}{n})} + n\log(\frac{1}{\epsilon})\right)$ in the expected case by using some ideas to be developed in the next section.

There are two natural ways to employ the basic procedure within other algorithms. One way is to precede another algorithm with the basic procedure, with the objective of lowering the expected cost when there are two or more contenders. The same objective can often be achieved by interleaving the basic procedure with another algorithm. The first approach can be applied when the number of coin tosses needed is known in advance, and the second works when the number of bits transmitted between successive coin toss exchanges is constant. We will use these ideas in subsequent algorithms.

3 Gap estimation

When the algorithm has even a crude upper bound on the ring size the possibility emerges of verifying that there is just one contender by measuring the lengths of gaps between successive contenders, and comparing them. Thus, more sophisticated Solitude Detection algorithms are based on efficient algorithms for counting, either exactly or approximately, the lengths of gaps between consecutive contenders on the ring. The simplest gap counting algorithm is a deterministic one. Each contender starts a counter, which is incremented and forwarded by each noncontender until it reaches the next contender. This costs $O(n \log n)$ bits in the worst case. More efficient algorithms are based on methods for obtaining estimates of the gap lengths.

Two gap estimation algorithms follow. The first is designed to obtain a very crude estimate at very low cost. The second obtains a good estimate at higher cost.

The first gap counting algorithm, algorithm $G1$, is similar in spirit to Greenberg and Ladner's estimation algorithm [11], although our algorithm tries estimates in *decreasing* order, which is crucial in its use. Assume that an upper bound N on n is known. Let $Q_0, \ldots, Q_{c-1}$ be the contenders in counter-clockwise order around the ring. Let g_j be the number of processors in the interval $[Q_{j-1}, Q_j)$. Let f be any increasing function on the nonnegative reals satisfying $f(t+1) \geq 2^{t+1} f(t)$ for all $t \geq 0$, and let $f^{-1}(x) = \sup\{t : f(t) \leq x\}$ be its functional inverse. So f grows quite rapidly. For our purposes, $f(t) = 2^{t^2}$ will suffice.

Gap estimation proceeds in rounds, starting with round 0. In round t, each passive processor performs a random experiment with success probability $\min(f(t)/N, 1)$. Each contender Q_{j-1} sends a message across the gap to its right, informing Q_j whether there were any successes in the gap. If there were no successes, then Q_j proceeds to round $t+1$. If there was a success in the gap to its left in round t, then Q_j sets its estimate $\hat{g}_j$ of g_j to $N/f(t+1)$, sends a message signaling the end of the gap estimation phase, and waits to receive such a message. Such messages end gap estimation for any processor receiving them, and are forwarded until they reach a processor that initiated one. Not all contenders Q_j will produce an estimate $\hat{g}_j$, but at least one must do so. Any contender that does not produce an estimate can be sure that it is not alone. It sends an alarm, which only serves to decrease the communication complexity.

To keep the complexity low when there are two or more contenders, the basic algorithm is interleaved with the steps just outlined. Algorithm $G1$ is summarized in Figure 2.

```
Contender $Q_j$:
    $t := 0$;
    repeat
        run the basic procedure for one round;
        send "any successes?" to the right;
        receive $m$ from the left;
        if $m =$ "end estimation" then abort and conclude not alone;
        $t := t + 1$
    until $m =$ "success!";
    $\hat{g}_j := N/f(t)$;
    send "end estimation" to the right;
    receive messages until "end estimation" arrives.

Noncontender:
    t := 0;
    loop
        run the basic algorithm for one round;
        receive message $m$ from left;
        r := Bernoulli-trial(success probability = $\min(f(t)/N, 1)$);
        if $m \neq$ "end estimation" and $r =$ success
            then send "success!" to right
            else send $m$ to the right;
        t := t + 1
    end loop.
```

Figure 2: Algorithm $G1$.

Lemma 3.1 *Algorithm $G1$ satisfies the following.*

(a) $0 < \mathrm{E}(\hat{g}_j) \le g_j$.

(b) If $c = 1$ *then* $\mathrm{E}\Big(f^{-1}(N/\hat{g}_0)\Big) = \mathrm{O}\Big(f^{-1}(N/n)\Big)$.

(c) The expected bit complexity of algorithm $G1$ is $\mathrm{O}\Big(nf^{-1}(N/n)\Big)$ *when* $c = 1$ *and* $\mathrm{O}(n)$ *when* $c > 1$.

Proof: (a) If g_j independent random experiments are performed, each with probability of success $f(t)/N$, then the probability of at least one success is at most $g_j\, f(t)/N$. So $\Pr(\hat{g}_j = N\,/f(t+1)\,) \le g_j\, f(t)/N$. Therefore

$$\mathrm{E}(\hat{g}_j) \le \sum_{t=0}^{\infty}\left(\frac{g_j\, f(t)}{N}\right)\left(\frac{N}{f(t+1)}\right) \le g_j.$$

(b) When there is a single contender, $T = f^{-1}(N/\hat{g}_0)$ rounds are used for gap estimation. We bound $\mathrm{E}(T)$, the expected stopping time. Choose k such that $N\,/f(k) < n \le N\,/f(k-1)$. The stopping times less than $k+3$ contribute less than $k+2$ to the expectation. It remains to show that the later stopping times make only a small contribution. But $\Pr(T = k+2+j) \le \Pr(\text{no successes at round } k+1+j\,)$. This is zero if $f(k+j) \ge N$, and otherwise it is $(1 - f(k+j)/N)^n < e^{-nf(k+j)/N}$. But $n > N\,/f(k)$, so $\Pr(T = k+2+j) < e^{-f(k+j)/f(k)}$. So from $k+3$ to ∞, the contribution to $\mathrm{E}(T)$ is at most $\sum_{j=1}^{\infty}(k+2+j)e^{-f(k+j)/f(k)}$, which is $\mathrm{O}(1)$.

(c) This follows easily from the fact that $\mathrm{E}(T) = \mathrm{O}\Big(f^{-1}(N/n)\Big)$, and from properties of the basic algorithm. ∎

Algorithm $G1$ gives a very crude estimate of a gap, but spends few bits to do so. We also require an algorithm that generally spends more bits, but achieves a more accurate estimate. We continue to assume that an upper bound N on n is known to the processors, and in addition assume that the gap being estimated has length at least $N/2k$ for some given k. Algorithm $G2$ has each contender start a counter, initially zero. That contender and each following noncontender tosses a biased coin, and increments the counter with probability

```
Contender $Q_j$:
    run the basic algorithm for $\lceil \log\log\lambda \rceil$ rounds;
    $r$ := trial();
    send r to the right;
    receive $b_j$ from the left;
    $\hat{g}_j = N b_j / \lambda$.
Noncontender:
    run the basic algorithm for $\lceil \log\log\lambda \rceil$ rounds;
    receive count $b$;
    $r$ := trial();
    send $b + r$ to the right.
Trial():
    $r$ := Bernoulli-trial(success probability = $\lambda/N$);
    return (if $r$ = success then 1 else 0).
```

Figure 3: Algorithm $G2$.

$p = \lambda/N$, where λ is chosen according to the desired accuracy. When a contender Q_j obtains a count b_j from its left, it sets its gap estimate $\hat{g}_j$ of the actual gap g_j to b_j/p. To keep the bit complexity low when there are two or more contenders, algorithm $G2$ starts with $\log\log\lambda$ coin toss exchanges. Figure 3 shows algorithm $G2$.

Lemma 3.2 *Let $c_1 > 1$ and $c_2 < 1$, and suppose $g_j \geq N/2k$. Then the gap estimate $\hat{g}_j$ produced by Q_j during algorithm $G2$ satisfies the following.*

(a) $\Pr(\hat{g}_j > c_1 g_j) < e^{-\lambda(1-\sqrt{c_1})^2/k}$.

(b) $\Pr(\hat{g}_j < c_2 g_j) < e^{-\lambda(1-\sqrt{c_2})^2/k}$.

(c) The expected bit complexity of algorithm $G2$ is $O(n\log\lambda)$ when there is only one contender and $O(n)$ when there are two or more contenders.

Proof: Part (c) is evident, since each gap counter has an expected value less than λ. Consider a sequence of T independent Bernoulli trials with success probability p, and let X be a random variable denoting the number of successes. Okamoto [14] shows that, for any

$c > 0$,

$$\Pr\left(\sqrt{X/T} - \sqrt{p} > c\right) \quad < \quad e^{-2Tc^2} \tag{3.1}$$

$$\Pr\left(\sqrt{X/T} - \sqrt{p} < -c\right) \quad < \quad e^{-2Tc^2} \tag{3.2}$$

The similarity of (3.1) and (3.2) allows us to prove only assertion (a), with (b) following almost identical reasoning. Let $T = g_j$, $p = \lambda/N$ and $X = p\hat{g}_j$. Then

$$\begin{aligned}
\Pr(\hat{g}_j > c_1 g_j) &= \Pr(X > c_1 T p) \\
&= \Pr\left(\sqrt{X/T} - \sqrt{p} > \sqrt{c_1 p} - \sqrt{p}\right) \\
&< e^{-2Tp(1-\sqrt{c_1})^2} && \text{by (3.1)} \\
&\leq e^{-\lambda(1-\sqrt{c_1})^2/k} && \text{since } T \geq N/2k.
\end{aligned}$$

■

4 Improved Algorithms

Recall that, when nothing is known about the ring size n, no distributively terminating Monte Carlo Solitude Detection algorithm exists. Assume that an upper bound N on n is known. Then, as is shown below, not only does a distributively terminating Solitude Detection algorithm exist, but its bit complexity is remarkably insensitive to the accuracy of N as an estimate of n. The algorithm is derived from the following idea. Each contender Q_j forms an estimate $\hat{g}_j$ of the size of the gap separating Q_j from the nearest contender to the left. Then Q_j uses $N/\hat{g}_j$ as an estimated upper bound on the number of contenders. This hypothetical upper bound is then used to select the parameter for the basic procedure in much the same way as the true upper bound was used in the algorithm of Theorem 2.3. (Here, each contender makes the bold assumption that all gaps are equal. It is remarkable that the algorithm performs well when the gaps are not at all equal, and the actual number of contenders exceeds the estimated upper bound.)

Theorem 4.1 *If an upper bound N on ring size is known by all processors, there is a distributively terminating one-sided Monte Carlo Solitude Detection algorithm with expected bit complexity* $\mathrm{O}\left(n\sqrt{\log(\frac{N}{n})} + n\log(\frac{1}{\epsilon})\right)$.

Proof: The algorithm starts with each contender executing algorithm $G1$, using function $f(t) = 2^{t^2}$. If contender Q_j obtains gap estimate $\hat{g}_j = a_j$, it executes the basic algorithm for $t = \left\lceil\log(\frac{2}{\epsilon}) + K_j\right\rceil$ rounds, where $K_j = \sqrt{2\log(N/a_j)}$.

Suppose there is a single contender. By Lemma 3.1(c), $\mathrm{O}\left(n\sqrt{\log(\frac{N}{n})}\right)$ bits are communicated on the average in the gap estimation phase. By Lemma 3.1(b), $\mathrm{E}(K_0) = \mathrm{O}\left(\sqrt{\log(\frac{N}{n})}\right)$. So the expected bit complexity is $\mathrm{O}\left(n\sqrt{\log(\frac{N}{n})} + n\log(\frac{1}{\epsilon})\right)$. When there are two or more contenders, each processor sends $\mathrm{O}(1)$ expected bits in the gap estimation phase and $\mathrm{O}(1)$ expected bits in later phases.

It remains to show that the algorithm erroneously asserts solitude with probability at most ϵ. This part of the analysis is a more elaborate version of the analysis in Lemma 2.1(b). The only feature of the gap estimates that is needed here is that $\mathrm{E}(\hat{g}_j) \leq g_j$.

Say that contender Q_j is *fooled* if Q_j reaches an incorrect decision. Let c be the actual number of contenders. For $j = 0, \ldots, c-1$ let F_j be the probability that contender Q_j is fooled, given that algorithm $G1$ has produced particular estimates $\hat{g}_j = a_j$.

Claim 4.2 *If $K_j \geq c$ then $F_j \leq (\epsilon/4)^{c-1}$.*

Proof: Since $K_j \geq c$, in order for Q_j to be fooled $K_j + \log(2/\epsilon)$ times, every contender must be fooled at least $1 + \log(2/\epsilon)$ times. Hence $F_j \leq 2^{-(c-1)\log(4/\epsilon)} = (\epsilon/4)^{c-1}$. ■

Claim 4.3 *If $K_j < c$ then $F_j \leq \epsilon\, a_j\,/2N$.*

Proof: In order for Q_j to be fooled, the K_j distinct processors $Q_j, Q_{j-1}, \ldots, Q_{j-K_j+1}$ must be fooled a total of $\sum_{k=1}^{K_j}(\log(2/\epsilon) + k) \geq \log(2/\epsilon) + K_j^{\,2}/2$ times. Hence $F_j \leq 2^{-(\log(2/\epsilon)+K_j^{\,2}/2)} = \epsilon a_j/2N$. ■

If Q_j does not produce a gap estimate, then Q_j is not fooled. Suppose Q_j produces an estimate. The probability that Q_j is fooled is at most $(\epsilon/4)^{c-1} + (\epsilon/2N)\mathrm{E}(\hat{g}_j \mid \hat{g}_j < c)$. But the conditional expectation of $\hat{g}_j$ is clearly at most $\mathrm{E}(\hat{g}_j)$, which by Lemma 3.1(a) is at most g_j. So the probability that Q_j is fooled is at most $(\epsilon/4)^{c-1} + \epsilon g_j/2N$. Hence the probability that some contender is fooled is at most $c(\epsilon/4)^{c-1} + (\epsilon/2N)\sum_j g_j$. Since $c(\epsilon/4)^{c-1} \leq \epsilon/2$ and $\sum_j g_j = n \leq N$, the probability that some contender is fooled is at most ϵ. ■

The preceding algorithms have positive error probability when there are two or more contenders. No algorithm can distinguish with certainty between a ring of size n containing one contender and a ring of size $2n$ containing two contenders directly opposite each other, as observed by Itai and Rodeh [6, 13]. Consequently, if all processors know at best that $\lfloor N/2 \rfloor \leq n \leq N$, there is no Las Vegas Solitude Detection algorithm. But if the processors know that $\lfloor N/2 \rfloor + 1 \leq n \leq N$, there is a deterministic Solitude Detection algorithm with worst case bit complexity $\mathrm{O}(n \log n)$ [2]. The algorithm simply has each processor execute deterministic gap counting, and then send its gap count to the next contender (possibly itself). A contender that receives two gap counts g_1 and g_2, representing the lengths of the two gaps to its left, is alone if and only if $g_1 = g_2 > N/2$.

Curiously, at very nearly the same degree of knowledge of ring size that error-free Solitude Detection becomes possible, the expected bit complexity of Monte Carlo Solitude Detection drops. The Monte Carlo algorithms employed so far have decreased the error probability by simply repeating an experiment. Such algorithms typically have a $\log(\frac{1}{\epsilon})$ term in their complexities. To lower the complexity, we need more subtle techniques.

Suppose all processors know that $(\frac{1}{2} + \rho)N \leq n \leq N$ for some $\rho > 0$. If there is only one contender, then there is only one gap, and its length must be at least $(\frac{1}{2} + \rho)N$. If there are two or more contenders, then some gap must have length at most $N/2$. The contenders execute a gap estimation algorithm whose accuracy is, with high probability, sufficient to distinguish between $(\frac{1}{2} + \rho)N$ and $N/2$. Using algorithm $G2$ as the gap estimation algorithm, the probability of obtaining a misleading count decreases at least exponentially with increasing λ, while the cost increases only logarithmically with increasing λ. Hence, the expected bit complexity is proportional to $\log\log(\frac{1}{\epsilon})$, where ϵ is the error probability bound. As ρ decreases, more accuracy is needed, so there is also a $\log(\frac{1}{\rho})$ term in the complexity. The same ideas can be employed even when ρ is unknown, and it is only known that $N/2 < n \leq N$.

Theorem 4.4 *There is a distributively terminating one-sided Monte Carlo Solitude Detection algorithm for rings of size $n \in (N/2, N]$ with expected bit complexity $O\left(n \min\left(\log n, \log(\frac{1}{\epsilon}), \log\log(\frac{1}{\epsilon}) + \log(\frac{1}{\rho})\right)\right)$, where $\rho = N/n - 1/2$.*

Proof: Since worst case bit complexity $O(n \log n)$ is achievable by a deterministic algorithm and expected case bit complexity $O\left(n \log(\frac{1}{\epsilon})\right)$ is achievable by Theorem 4.1, it suffices to produce an algorithm with expected bit complexity $O\left(n \log\log(\frac{1}{\epsilon}) + \log(\frac{1}{\rho})\right)$. We begin by producing an algorithm with two-sided error, and then show how to eliminate the error on one side.

The value of ρ does not need to be known by the algorithm. However, we initially presume that ρ is known, and show later how to do without that knowledge.

First each contender obtains a gap count by executing algorithm $G2$ with parameter λ to be chosen below. Let a_1 be the gap estimate obtained by a given contender. Each contender sends its estimate a_1 (in the form of the raw count $a_1\lambda/N$) to the next contender, and receives a_2. It concludes that it is alone if $a_1 = a_2 \geq \frac{1+\rho}{2}N$.

We can assume that ρ is small. Then an appropriate choice of λ is $\lambda = \frac{5}{\rho^2}\ln\frac{1}{\epsilon}$. The expected bit complexity is $O(n \log \lambda) = O\left(n \log\log(\frac{1}{\epsilon}) + n \log(\frac{1}{\rho})\right)$ when there is one contender, and $O(n)$ when there are two or more. Now we bound the error probabilities.

Claim 4.5 *The probability that a sole contender erroneously concludes that it is not alone is at most ϵ.*

Proof: The sole contender is fooled only if its estimate of the gap length is less that $\frac{1+\rho}{2}N$, which, by Lemma 3.2(b), happens with probability less than $e^{-\lambda(1-\sqrt{c_2})^2}$ for $c_2 = (1+\rho)/(1+2\rho)$. But $(1-\sqrt{c_2})^2 \approx \rho^2/4$ for small ρ, so the probability that a sole contender is fooled is less than $e^{-\lambda\rho^2/5} < \epsilon$. ∎

Claim 4.6 *The probability that some contender erroneously concludes that it is alone is at most ϵ.*

Proof: Say that a contender is almost fooled if its estimates a_1 and a_2 satisfy $a_1 + a_2 \geq (1+\rho)N$. A contender erroneously concludes that it is alone only if it is almost fooled, so it suffices to bound the probability that some contender is almost fooled. That probability is maximized when there are just two contenders, since changing any one of $c \geq 3$ contenders to a non-contender can only increase the probability that some contender is almost fooled. Then the total number of processors that incremented a counter is $(a_1 + a_2)\lambda/N$, and $a_1 + a_2$ is the gap estimate that a sole contender would have obtained with the same coin tosses. The worst case occurs when $n = N$. By Lemma 3.2(a), the probability that some contender is almost fooled is $\Pr(\hat{g}_0 \geq (1+\rho)N) < e^{-\lambda(1-\sqrt{c_1})^2}$ for any $c_1 < 1+\rho$. Choosing c_1 arbitrarily close to $1+\rho$ results in $(1-\sqrt{c_1})^2 \approx \rho^2/4$, so, as in Claim 4.5, the error probability is less than ϵ. ■

The algorithm just described has two-sided error. Errors when there actually is a single contender can be eliminated as follows. First run the algorithm as given. If the outcome is "alone", then conclude alone and halt. Otherwise, run the algorithm of Theorem 4.1, and adopt its conclusion. The modified algorithm does not err when there is one contender, and has error probability less than 2ϵ when there are two or more contenders. When $n \geq (\frac{1}{2} + \rho)N$, and there is in fact one contender, the modified algorithm has expected bit complexity $O\left(n \log\log(\frac{1}{\epsilon}) + n\log(\frac{1}{\rho}) + \epsilon n \log(\frac{1}{\epsilon})\right)$. But $\epsilon \log(\frac{1}{\epsilon}) \leq \frac{1}{2}$ for $\epsilon \leq 1/4$.

Finally, we remove the presumption that ρ is known. The algorithm simply tries successively smaller values of ρ and ϵ, starting with $\rho_0 = 1/\log(\frac{1}{\epsilon})$ and $\epsilon_0 = \epsilon/2$, and proceeding with (ρ_1, ϵ_1), (ρ_2, ϵ_2), ... with $\rho_{i+1} = \rho_i^2$ and $\epsilon_{i+1} = \epsilon_i/2$. The algorithm terminates with answer "alone" when a trial with value $\rho = \rho_i$ yields a gap estimate larger than $(\frac{1}{2} + \frac{\rho}{2})N$. Otherwise, it proceeds to the next trial, with $\rho = \rho_{i+1}$. When $\log\log(\frac{1}{\epsilon}) + \log(\frac{1}{\rho}) > \min(\log n, \log(\frac{1}{\epsilon}))$, a different algorithm is started.

When there are two or more contenders, we rely on an interleaved basic algorithm to keep the complexity low. When there is just one contender, the expected number of trials is $O\left(\log\log(\frac{1}{\rho})\right)$, and the total error probability is at most $\sum_i \epsilon_i < \epsilon$. Each trial costs approximately twice as much as the previous trial, and the expected cost is clearly the desired cost. ■

5 Algorithms for Known Ring Size

In this section it is assumed that the algorithm knows the ring size exactly. Solitude Detection algorithms are described for four conditions, depending on whether a Las Vegas or one-sided Monte Carlo algorithm is desired, and whether the algorithm must terminate distributively or not. Since all of the algorithms are similar, they are all presented as a single parameterized algorithm, consisting of four stages. Not all stages are executed in all conditions. Stage 3 is only executed if distributive termination is required. Stage 4 is only executed by Monte Carlo algorithms.

Let $\nu(n)$ denote the smallest positive nondivisor of n. Observe that $\nu(n)$ is a prime power; say, $\nu(n) = p^s$. Let $\vartheta(x) = \ln \prod_{q \leq x} q$, where q ranges over primes. Then $\lim_{x \to \infty} \vartheta(x)/x = 1$ [7]. Clearly $\ln n \geq \vartheta(\nu(n) - 1)$, so $\nu(n) = O(\log n)$. The algorithm has an integer parameter $l > 1$, which is adjusted according to type of algorithm desired. Let t be the smallest integer such that $p^t \geq l$, and let $m = p^{s+t}$. Notice that m does not divide n and $m > l$.

The algorithm is described for a contender. Non-contenders cooperate as described. If a contender receives evidence that it is not alone before the algorithm is finished then it sends one of two kinds of alarm. A loud alarm is sent if the evidence is conclusive. Having sent a loud alarm, a contender aborts the algorithm, and concludes that it is not alone. A soft alarm is sent during a Monte Carlo algorithm when a contender has received strong but inconclusive evidence that it is not alone. After sending a soft alarm, a contender waits to receive a soft alarm, then proceeds directly to stage 4.

Alarms are forwarded by non-contenders. A contender that receives a message of a type different from what it is expecting, including any kind of alarm, aborts what it is doing, and sends a loud alarm. Each contender sends at most one alarm of each kind. A contender that has finished the algorithm without sending or receiving a loud alarm concludes that it is alone.

Stage 0: (This stage makes the algorithm one-sided linear.) Run $\lceil 2 \log m \rceil$ rounds of the basic procedure.

Stage 1: (This stage will generate an alarm if there are c contenders, where $2 \leq c \leq l$.) Send a counter, initially 1, to the right. The counter is incremented mod m^2 by each non-contender, and propagates to the next contender. Receive a count from the left. If the count is not congruent to $n \pmod{m^2}$, send a loud alarm.

Stage 2: (This stage generates an alarm within every sequence of l distinct contenders, provided none of them sent an alarm in stage 1.) Inform the contender to the right whether the distance separating it from yourself is greater than n/l.

(a) For a Las Vegas algorithm, send a counter, initially 1, to the right. Each non-contender increments the counter, until the counter reaches a value greater than n/l. At that point, the message "long" is propagated to the next contender. Receive a message from the left. If the message is not "long," send a loud alarm.

(b) For a Monte Carlo algorithm with error probability at most ϵ, use a truncated version of algorithm $G2$ with increment probability λ/n, where λ is chosen below. When the counter reaches a value greater than $2\lambda/l$, the message "long" is propagated to the next contender. Receive a message from the left. If the message is a counter, rather than "long," let $\hat{g}$ be the estimate, send a soft alarm, wait for a soft alarm to arrive, then go to stage 4.

Stage 3: (This stage is only executed if distributive termination is desired. It serves to flush alarms.) Alternately send and receive l "ok" messages.

Stage 4: (This stage is only executed by processors that sent a soft alarm. It eliminates the possibility of error when there is a single contender.) Let $\hat{g}$ be the estimate computed in stage 2. Execute the basic procedure for t rounds, where t is chosen below. Send a loud alarm as soon as the toss received does not match that just sent.

Lemma 5.1 *Suppose that, in every sequence of $l+1$ consecutive contenders, at least one of the contenders sends an alarm in stage 1 or 2. Then every contender eventually sends an alarm.*

Proof: If the algorithm terminates nondistributively, then all contenders will send an alarm if any do. If the algorithm terminates distributively, then stage 3 will cause an alarm to move through l contenders. ■

Lemma 5.2 *If there are c contenders, where $2 \leq c \leq l$, then some contender sends a loud alarm in stage 1, and all contenders conclude that they are not alone.*

Proof: Suppose no alarm is sent in stage 1. Let $g_1, \ldots, g_c$ be the lengths of the gaps separating the contenders. Then $g_1 + \cdots + g_c = n$. Since no alarms are sent at stage 1, it must be the case that $g_j \equiv n \pmod{m^2}$ for $j = 1, \ldots, c$. Let r be the remainder when n is divided by m^2. Then $cr \equiv g_1 + \cdots + g_c \equiv r \pmod{m^2}$, from which it follows that $m^2 \mid (c-1)r$. But $m \nmid n$, so $m \nmid r$, and, since m is a prime power, $m \mid (c-1)$. Hence, $c > m > l$, contradicting the required condition. The second statement follows from Lemma 5.1. ■

Theorem 5.3 *There is a one-sided linear Las Vegas Solitude Detection algorithm for rings of known size n that*

(a) terminates distributively and transmits $\mathrm{O}\left(n\sqrt{\log n}\right)$ bits in the worst case when there is just one contender, or

(b) terminates nondistributively and transmits $\mathrm{O}(n \log\log n)$ bits in the worst case when there is just one contender.

Proof: Correctness. Loud alarms are sent only when a contender has conclusive evidence that it is not alone. Hence, when there is a single contender, the algorithm answers correctly. Suppose there are $c \geq 2$ contenders. If $c \leq l$ then all contenders conclude that they are not alone, by Lemma 5.2. If $c > l$, then at least one of any sequence of $l + 1$ consecutive gaps must have length less than n/l. So some contender will detect a short gap at stage 2, and will send an alarm. By Lemma 5.1, all contenders send an alarm. Only loud alarms are sent in Las Vegas algorithms, so all contenders conclude that they are not alone.

Complexity. The coin tosses in stage 0 are sufficient to keep the expected bit complexity linear when there are two or more contenders, so consider the case of just one contender. In stages 0 and 1 $O(n \log m) = O\left(n \log \nu(n) + n \log l\right)$ bits are exchanged, and in stage 2 $O(n + (n/l) \log n)$ bits are exchanged. For nondistributive termination, no other stages are executed. Choosing $l = \lceil \log n \rceil$ for $n > 2$ yields total bit complexity $O\left(n \log \nu(n) + n \log \log n\right) = O(n \log \log n)$. For distributive termination, stage 3 costs an additional $O(nl)$ bits. The total bit complexity of $O\left(n \log \nu(n) + (n/l) \log n + nl\right)$ is minimized, to within a constant factor, by choosing $l = \left\lceil \sqrt{\log n} \right\rceil$ for $n > 2$. That yields total bit complexity $O\left(n \sqrt{\log n}\right)$. ■

Theorem 5.4 *There is a one-sided Monte Carlo Solitude Detection algorithm for rings of size n that*

(a) terminates distributively and transmits $O\left(n \min\left(\sqrt{\log n},\ \log \log(\frac{1}{\epsilon}),\ \log \nu(n) + \sqrt{\log \log(\frac{1}{\epsilon})}\right)\right)$ *expected bits when there is just one contender, or*

(b) terminates nondistributively and transmits $O\left(n \min\left(\log \log n,\ \log \log(\frac{1}{\epsilon}),\ \log \nu(n) + \log \log \log(\frac{1}{\epsilon})\right)\right)$ *expected bits when there is just one contender.*

Proof: Correctness. A loud alarm is sent only when a contender receives conclusive evidence that it is not alone, so suppose there are two or more contenders. There are two ways to err: either some contender terminates without having sent any kind of alarm, or every contender sends a loud or soft alarm, and some contender fails to send a loud alarm at stage 4. We show that the probability of each kind of error occurring is at most $\epsilon/2$.

Let p_1 be the probability of the former kind of error, and suppose that such an error occurs. Say that a gap is short if its length is less than n/l, and is long if its length is greater than $2n/l$. By Lemma 5.2, no alarms are sent in stage 1, so there must be more than l contenders. So every sequence of $l+1$ consecutive gaps must include a short gap. By Lemma 5.1, since some contender fails to send an alarm in stage 2, there must be some short gap G that is measured as being long in stage 2(b). The probability q of that occurring is maximized when G has length $n/l - 1$. By Lemma 3.2, $q < e^{-2\lambda(1-\sqrt{2})^2/l} < e^{-\lambda/3l}$. The probability that *some* short gap is measured as long is maximized when there are fewer than $2l$ contenders, since combining two very short gaps to produce a short gap can only increase the probability of many increments in some short gap. So $p_1 < 2le^{-\lambda/3l}$. Choose $\lambda = 3l\log(4l/\epsilon)$. Then $p_1 < \epsilon/2$.

Now consider the second kind of error, that every contender sends some kind of alarm, but some contender fails to send a loud alarm in stage 4. Loud alarms can only help, so suppose all contenders send a soft alarm. Stage 4 is just the algorithm of Theorem 4.1, with the gap estimate already provided. Our estimate $\hat{g}_j$ of the gap length g_j has expected value exactly g_j, assuming stage 4 is reached, so the error analysis of Theorem 4.1 applies. In order to drop the error probability to $\epsilon/2$, we choose $t = \log(4/\epsilon) + \sqrt{2\log(n/\hat{g}_j)}$.

Complexity. Stage 0 guarantees expected linear bits when there are two or more contenders, so consider the case of a single contender. Stage 1 costs $O(n\log m) = O\Big(n\log\nu(n) + n\log l\Big)$ bits. The counter at stage 2 travels a distance that is binomially distributed with a mean of about $2n/l$ before becoming "long", so the expected bit complexity of stage 2 is $O\Big(n + \frac{n}{l}\log\frac{\lambda}{l}\Big) = O\Big(n + \frac{n}{l}\log\log l + \frac{n}{l}\log\log(\frac{1}{\epsilon})\Big)$.

Stage 4 is relatively costly, but fortunately it is rarely executed. The probability that a sole contender reaches stage 4 is less than $e^{-2\lambda(1-\sqrt{c_2})^2}$ for $c_2 > 2/l$, by Lemma 3.2. We will choose $l > 8$, so let $c_2 = 1/4$. Since $\lambda = 3l\log(4l/\epsilon)$, the probability of reaching stage 4 is less than ϵ. Given that a single contender does reach stage 4, its estimate $\hat{g}$ is almost surely very close to the maximum possible value, $2n/l$. So $E(\sqrt{\log(n/\hat{g})}) < \sqrt{\log l}$. The contribution of stage 4 to the expected bit complexity is thus $O\Big(\epsilon n\log(\frac{1}{\epsilon}) + \epsilon n\sqrt{\log l}\Big) = O(n)$ for the choices of l to be made below.

For nondistributive termination, choose $l = \max(9, \lceil\log\log(\frac{1}{\epsilon})\rceil)$. Then the expected bit complexity for stages 1, 2 and 4 is $O\Big(n\log\nu(n) + n\log l + \frac{n}{l}\log\log(\frac{1}{\epsilon})\Big) = O\Big(n\log\nu(n) + n\log\log\log(\frac{1}{\epsilon})\Big)$.

For distributive termination, stage 3 contributes an additional $O(nl)$ bits. Choose $l = \max(9, \sqrt{\log\log(\frac{1}{\epsilon})})$. Then the expected bit complexity is $O\Big(nl + n\log\nu(n) + \frac{n}{l}\log\log(\frac{1}{\epsilon})\Big) = O\Big(n\log\nu(n) + n\sqrt{\log\log(\frac{1}{\epsilon})}\Big)$.

knowledge	distributive termination	nondistributive termination
$N/2 \leq n \leq N$	impossible	impossible
$N/2 < n \leq N$	$\Theta(n \log n)$	$\Theta(n \log n)$
$n = N$	$\Theta(n\sqrt{\log n})$	$\Theta(n \log\log n)$

Table 1: The expected bit complexity of Las Vegas (error-free) Solitude Detection.

For very small ϵ, the Las Vegas algorithm of Theorem 5.3 is more efficient than the one just analyzed. Also, in some cases the $n \log \nu(n)$ term can dominate, and it might be better to run the algorithm of Theorem 4.4, with $\rho = \frac{1}{2}$. By choosing the best algorithm for the situation, the expected bit complexity becomes $\mathrm{O}\left(n \min\left(\log \nu(n) + \log\log\log(\frac{1}{\epsilon}), \log\log n, \log\log(\frac{1}{\epsilon})\right)\right)$ for nondistributive termination, and $\mathrm{O}\left(n \min\left(\log \nu(n) + \sqrt{\log\log(\frac{1}{\epsilon})}, \sqrt{\log n}, \log\log(\frac{1}{\epsilon})\right)\right)$ for distributive termination. ∎

6 Conclusion

This paper has demonstrated several techniques for solving Solitude Detection on asynchronous anonymous unidirectional rings using few expected bits. Given the variety of techniques employed, it would not be at all surprising to find that other techniques yield even better algorithms. But that is not the case. Lower bounds that match (to within a constant factor) the upper bounds proved here are proved in [5]. In fact, the algorithms were in some cases inspired by the lower bound proofs.

Tables 1 and 2 summarize the results of this paper and its companion paper [5]. Sometimes, distributive termination costs more bits than nondistributive termination, as one would expect. But when processors know that ring size n satisfies $N/2 < n \leq N$, the cost of achieving distributive termination is at most a constant factor worse than that of nondistributive termination. Also, although the tables show that more information generally helps, there are exceptions. For example, when nondistributive termination suffices, there is no difference

knowledge	distributive termination	nondistributive termination
none	impossible	$\Theta\Big(n\log(\frac{1}{\epsilon})\Big)$
$a \leq n \leq N$ (*)	$\Theta\Big(n\sqrt{\log(\frac{N}{n})} + n\log(\frac{1}{\epsilon})\Big)$	$\Theta\Big(n\log(\frac{1}{\epsilon})\Big)$
$N/2 < n \leq N$	$\Theta\Big(f(n, n/N - 1/2, \epsilon)\Big)$	$\Theta\Big(f(n, n/N - 1/2, \epsilon)\Big)$
$n = N$	$\Theta\Big(n\min\Big($ $\log\nu(n) + \sqrt{\log\log(\frac{1}{\epsilon})}$, $\sqrt{\log n}$, $\log\log(\frac{1}{\epsilon})\Big)\Big)$	$\Theta\Big(n\min\Big($ $\log\nu(n) + \log\log\log(\frac{1}{\epsilon})$, $\log\log n$, $\log\log(\frac{1}{\epsilon})\Big)\Big)$
$f(n,\rho,\epsilon) = \min\Big(n\log n,\ n\log(\frac{1}{\epsilon}),\ n\log\log(\frac{1}{\epsilon}) + n\log(\frac{1}{\rho})\Big)$		

(*) $a \leq N/2$; lower bounds require $n \leq N/2$.

Table 2: The expected bit complexity of Monte Carlo Solitude Detection with error probability $\epsilon > 0$.

between the cases where just an upper bound on n is known and where nothing at all is known about n.

What the tables do not show is that the upper and lower bounds are in fact stronger than stated, and contrast in strength in several ways. In the case of Monte Carlo algorithms, the lower bounds apply to algorithms with two-sided error, while the upper bounds all achieve one-sided error. Our algorithms never deadlock. But the lower bounds apply even to algorithms that might deadlock or loop forever with arbitrarily high probability, and need never do anything useful (but must not lie) when there are two or more contenders; the given bound applies to the class of computations with a single contender that do terminate with the correct answer. Our nondistributively terminating algorithms actually *distributively reject*: a decision that there are two or more contenders is never revoked on later receipt of a message. Our lower bounds for distributively terminating algorithms apply also to algorithms that *distributively accept*, but can nondistributively reject. Our upper bounds are for anonymous rings. Our lower bounds permit processors to have identities, provided that those identities are *not* guaranteed to be distinct, and the algorithm must be correct (with the required probability) for any sequence of identifiers; the lower bounds apply to the best case over all identifier sequences. Our upper bounds all require only $O(n)$ expected bits when there are two or more contenders; the lower bounds neither constrain nor make assumptions about complexity when there are two or more contenders.

References

[1] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. The bit complexity of probabilistic leader election on a unidirectional ring. In *Distributed Algorithms on Graphs*, pages 1–12. Carleton University Press, 1986. Proc. 1st International Workshop on Distributed Algorithms.

[2] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 18(1):12–29, 1989.

[3] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Probabilistic evaluation of common functions on rings of known size. Technical Report 88-15, University of British Columbia, 1988.

[4] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Randomized function evaluation on a ring. *Distributed Computing*, 3(3):107–117, 1989.

[5] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Tight lower bounds for probabilistic solitude verification on anonymous rings. Technical Report TR 90-4, University of British Columbia, 1990.

[6] 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.

[7] T. M. Apostol. *Introduction to Analytic Number Theory.* Springer-Verlag, New York, 1976.

[8] H. Attiya and M. Snir. Better computing on the anonymous ring. In *Proc. Aegean Workshop on Computing*, pages 329–338, 1988.

[9] H. L. Bodlaender. New lower bound techniques for distributed leader finding and other problems on rings of processors. Technical Report RUU-CS-88-18, Rijksuniversiteit Utrecht, 1988.

[10] P. Duris and Z. Galil. Two lower bounds in asynchronous distributed computation (preliminary version). In *Proc. 28nd Annual Symp. on Foundations of Comput. Sci.*, pages 326–330, 1987.

[11] A. Greenberg and R. Ladner. Estimating the multiplicities of conflicts in multiple access channels. In *Proc. 24nd Annual Symp. on Foundations of Comput. Sci.*, pages 383–392, 1983.

[12] L. Higham. *Randomized Distributed Computing on Rings.* PhD thesis, University of British Columbia, Vancouver, Canada, 1988.

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

[14] M. Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10:29–35, 1958.