The Bit Complexity of Randomized Leader Election on a Ring

Karl Abrahamson *
Andrew Adler +
Rachel Gelbart *
Lisa Higham *
David Kirkpatrick *

Technical Report 86-3
February 1986

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

Abstract

The inherent bit complexity of leader election on asynchronous unidirectional rings of processors is examined under various assumptions about global knowledge of the ring. If processors have unique identities with a maximum of m bits, then the expected number of communication bits sufficient to elect a leader with probability 1, on a ring of (unknown) size n is $O(nm)$. If the ring size is known to within a multiple of 2, then the expected number of communication bits sufficient to elect a leader with probability 1 is $O(n \log n)$.

These upper bounds are complemented by lower bounds on the communication complexity of a related problem called solitude verification that reduces to leader election in $O(n)$ bits. If processors have unique identities chosen from a sufficiently large universe of size s, then the average, over all choices of identities, of the communication complexity of verifying solitude is $\Omega(n \log s)$ bits. When the ring size is known only approximately, then $\Omega(n \log n)$ bits are required for solitude verification. The lower bounds address the complexity of certifying solitude. This is modelled by the best case behaviour of non-deterministic solitude verification algorithms.

The Bit Complexity of Randomized Leader Election on a Ring

Karl Abrahamson *, Andrew Adler +, Rachel Gelbart *, Lisa Higham *, David Kirkpatrick *

*** Department of Computer Science / + Department of Mathematics**

University of British Columbia, Vancouver, B.C., Canada

1. Introduction

Studies in the complexity of distributed computation typically ask two questions. i) What is the complexity, under some chosen measure(s), of a chosen problem or family of problems, on distributed networks formalized in a chosen model? (The model might include both the network topology and assumptions about the processors and their interprocessor communication.) ii) How is this complexity affected by changes in the model? This paper addresses these questions for the problem of electing a leader using randomized distributed algorithms running on asynchronous unidirectional rings of processors, where the measure of complexity is the expected number of bits transmitted.

Leader election results in a unique processor, from among a specified subset of the processors, entering a distinguished final state. This problem is one of a small number of problems which are fundamental in that their solutions form the building blocks of many more involved distributed computations. Earlier work in the study of distributed computation has established the importance of the ring topology as a test-bed for the design and analysis of distributed algorithms. It is the chosen model here because it is a simple topology which exhibits many important attributes of distributed computations.

A unidirectional ring can be viewed as a sequence $P_0,\ldots,P_{n-1}$ of processors where each processor P_i sends messages to P_{i+1} and receives messages from P_{i-1} (subscripts are implicitly

reduced modulo n). A number of variants of this basic model are distinguished by supplementary properties. Communication between processors is either synchronous or asynchronous. Processors may be indistinguishable or may have distinct identifiers. Processors may or may not know the size of the ring at the start of computation. The complexity, measured by the number of communication messages, of leader election on models with various combinations of these properties has been well studied.

On an asynchronous unidirectional ring of processors with distinct identifiers, a leader can be elected by a deterministic algorithm, which operates using pairwise comparisons of processor identifiers, using $O(n \log n)$ messages each of $O(m)$ bits [DKR,Pe], where m is the number of bits in the largest processor identifier. If interprocessor communication is synchronous and identifiers are drawn from some known countable universe, then $O(n)$ messages suffice to elect a leader in the worst case [FL]. On the other hand, in the asynchronous case, if the universe of identifiers is unbounded, any deterministic leader election algorithm must exchange $\Omega(n \log n)$ messages (of arbitrary length) in the worst case [B,PKR] or average case [PKR], even if bidirectional communication is possible. Even if the ring size n is known to all processors and messages are transmitted synchronously, algorithms which are restricted to operate either on the basis of comparisons of processor identifiers or within a bounded number of rounds, must transmit $\Omega(n \log n)$ messages in the worst case [FL].

If processors are not endowed with distinct identifiers then, as was first observed by Angluin [A], deterministic algorithms are unable to elect leaders, even if n is known to all processors. Itai and Rodeh [IR] propose the use of randomized algorithms to skirt this limitation. They present a randomized algorithm that elects a leader in an asynchronous ring of known size n using $O(n \log n)$ expected messages of $O(\log n)$ bits each. The lower bound results of [Pa] show that even if processors have distinct identifiers drawn from some sufficiently large universe, the expected number of messages (of arbitrary length) communicated by a randomized leader election algorithm is $\Omega(n \log n)$. However, $O(n)$ expected messages suffice for randomized leader election on a synchronous ring without identifiers [IR], provided the ring size n is known to all processors.

In this paper, it is shown that with respect to bit complexity, the algorithms cited above for asynchronous rings are not optimal. The relationship between leader election and two subproblems, called attrition and solitude verification, is explored in section 2. Efficient reductions are established which motivate the development of procedures for these two subproblems (section 3) and lower bounds for solitude verification (section 4).

It follows from the results of section 3 that when each processor in a unidirectional ring of n processors, has a distinct m-bit identifier, it is possible to elect a leader by a randomized algorithm using $O(mn)$ expected bits of communication. In addition, when the processors are indistinguishable but each knows the ring size n to within a factor of 2, then it is possible to elect a leader by a randomized algorithm using $O(n \log n)$ expected bits of communication.

These upper bounds are complemented by the lower bounds of section 4. It follows from the results of that section that when processors have unique identifiers drawn from a sufficiently large universe of size s then any algorithm must transmit $\Omega(n \log s)$ bits to elect a leader, even in the best case. If processors are indistinguishable but each knows the ring size only to within some interval of size Δ, then any algorithm must transmit $\Omega(n \log \Delta)$ bits to elect a leader, even in the best case.

The lower bounds are proved for successful computations of algorithms in a very general model. Algorithms may be non-deterministic and non-uniform and may deadlock. Moreover computations need only terminate in the weak non-distributive sense. An algorithm terminates distributively if whenever processors enter a decision state the decision is irrevocable, that is it will not change in light of subsequent messages received. The weaker non-distributive termination refers to the situation in which the cessation of message traffic is not necessarily detectable by individual processors and all decisions are predicated on this undetectable condition.

The upper and lower bounds are tight to within constant factors, except when the ring size is known to lie within some relatively small interval. This gap is reminiscent of earlier results concerning knowledge of ring size [FL]. The special case when the ring size is known exactly is the subject of a companion paper [AAHK1]. Another direction for investigation is uncertain

leader election, that is probabilistic leader election that terminates correctly with probability greater than $1 - \epsilon$ for some fixed $\epsilon > 0$. A second companion paper, [AAHK2], considers this problem on rings of indistinguishable processors. Pachl, [Pa], studies the problem when processors have distinct identifiers. Some of these results together with an overview of the results of the present paper are briefly described in section 5.

2. Leader Election, Attrition and Solitude Verification

This section sets out a general framework for the study of leader election on rings. Two fundamental problems are introduced and their relationship to leader election is established. This relationship motivates the algorithms and lower bound results of the next two sections.

Leader election requires that a single processor be chosen from among some non-empty subset of processors called *candidates*. Initially each candidate is a *contender*. Intuitively, a leader election algorithm must i) eliminate all but one contender by converting some of the contenders to *non-contenders*, and ii) confirm that only one contender remains. This separation was pointed out and exploited earlier by Itai and Rodeh [IR]. These subtasks are called attrition and solitude verification respectively. More formally, a procedure solves the *attrition problem* if, when initiated by every candidate, it eventually takes all but exactly one of these candidates into a permanent state of non-contention. Typically an attrition procedure does not terminate but rather enters an infinite loop in which the remaining contender continues to send messages to itself. An algorithm solves the *solitude verification problem* if, when initiated by a set of processors, it eventually terminates with an initiator in state "yes" if and only if it was the sole initiator. The stronger *solitude detection problem* requires, in addition, that all initiators be left in state "no" if there was more than one initiator.

Both attrition and solitude detection can be reduced to leader election with $O(n)$ bits of communication on rings of size n. For the attrition reduction, a leader is elected from among the attrition contenders. For the solitude verification reduction, the processors wishing to detect

their solitude first use $O(n)$ bits to alert the whole ring to contend for leadership.[1] Once a leader is elected, it is easy to see how to solve attrition (no additional communication) or to detect solitude (an additional $O(n)$ bits of communication). Hence, non-linear lower bounds on the complexity of either attrition or solitude verification translate to lower bounds on the complexity of leader election. Conversely, attrition and solitude verification can be interleaved to solve leader election by annotating attrition messages with solitude verification messages. Whenever a contender enters a state of non-contention, it forwards a solitude verification restart message to alert remaining contenders that they were not previously alone. When attrition has reduced the set of contenders to one, solitude verification will proceed unintercepted, eventually verifying that only one contender, the "leader," remains. If the solitude verification algorithm terminates distributively, so does the resulting leader election algorithm.

The efficiency of the leader election algorithm described above depends not only on the efficiency of the attrition and solitude verification procedures from which it is constructed, but also on the cost of interleaving. If solitude verification is *conservative* in the sense that every message is bounded in length by some fixed constant number of bits, then annotated attrition messages are at worst a constant factor longer than unannotated messages. So the cost of premature attempts to verify solitude is dominated by the cost of attrition. The only remaining cost attributable to interleaving involves the transmission of restart messages. In general this cost can also be subsumed by the cost of attrition. This is shown in the next section for the attrition procedure used in this paper. That section describes a randomized attrition procedure, two different conservative solitude verification algorithms, (each exploiting different possible properties of the ring of processors), and an interleaving strategy that combines the procedures to yield efficient leader election algorithms.

[1] The reduction outlined here is from solitude verification to a version of leader election in which all processors are candidates for leadership. In fact, this can be generalized to a reduction to an arbitrary set of candidates as is shown in section 4.2.

3. Procedures for Attrition and Solitude Verification

The previous section argues that leader election can be efficiently reduced to attrition and solitude verification. This section describes and analyses efficient procedures for these two subproblems. The attrition procedure is randomized but completely general in that it makes no assumptions about the host ring. Two solitude verification algorithms are described which exploit different assumptions about the ring, specifically the existence of distinct identifiers or at least partial knowledge of the ring size. Both solitude verification algorithms are deterministic and terminate distributively.

3.1. The Attrition Procedure

3.1.1. Informal description

The attrition procedure is initiated by all candidates for leadership. The number of candidates is denoted by c. The candidates are the initial contenders; all other processors are non-contenders. The procedure uses coin tosses to eliminate some contenders while ensuring that it is not possible for all contenders to be eliminated. A non-contender never converts to a contender, but behaves entirely passively — simply forwarding any messages received. Contenders create messages which are propagated to the next contender. Since contenders (respectively non-contenders) have active (respectively passive) roles in the algorithm, they are referred to as *active (passive)* processors in the following description.

Like the randomized attrition procedure of [IR], the procedure here can be thought of as proceeding in *phases*. However, these phases are implicit only. They are not enforced by counters, but will be justified in the subsequent analysis of the procedure.

At the beginning of each phase, each active processor tosses an unbiased coin yielding h or t, sends the outcome to its successor, and waits to receive a message from its predecessor. Suppose an active processor, P, sends and receives the same coin toss. It is possible that this is true for every active processor, so no decision can be taken by P. P continues alternately to send

and receive coin tosses, remaining in the same phase, until P receives a message different from what it most recently sent. Suppose P eventually sends t and receives h. Then some other active processor, Q, must have sent h and received t. If only one of P and Q becomes passive, the possibility of losing all active processors is avoided. The convention used is that sending t and receiving h changes an active processor to passive in the next phase, while the opposite results in a processor remaining active in the next phase. Once any active processor has decided its state (active or passive) for the next phase, it sends a * message to signal the end of its phase. Any undecided processor, Q, that receives a * message while waiting for a coin flip becomes passive in the next phase, since it is assured that there will remain at least one active processor. Q forwards the * message to announce that it has ended its phase. * messages continue to be forwarded until received by a decided processor (which has already propagated a * message).

These ideas are formalized in the transition diagram of figure 1. The notation "x/y" is read

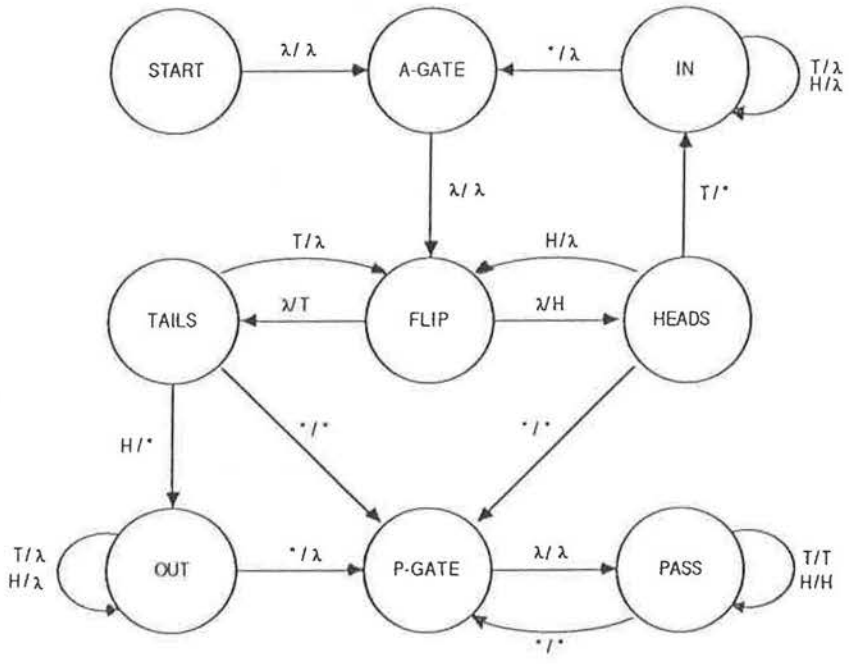

Figure 1. The Attrition Procedure

"receive x, then send y". λ is used where no reception or transmission is to occur. Each processor begins at START. The transition leaving state FLIP is chosen by an unbiased coin toss.

3.1.2. Formal description and correctness

This section establishes the following properties of the attrition algorithm of figure 1:

1. There is always at least one active processor. (A processor is active if it is in any state other than P-GATE or PASS).
2. The attrition procedure cannot deadlock.
3. The number of active processors never increases and eventually decreases to one.

Some definitions are introduced to facilitate the proofs. Let each processor P_x maintain an internal *phase counter*, ρ_x. ρ_x is initialized to 0 and incremented each time P_x enters a gate state. When $\rho_x = k$, P_x is in *phase* k. The following variables are defined relative to an arbitrary computation of the attrition process:

$s_{x,j}^{(k)} =$ the j^{th} message sent by P_x while $\rho_x = k$.

$r_{x,j}^{(k)} =$ the j^{th} message received by P_x while $\rho_x = k$.

$q_{x,j}^{(k)} =$ the state of P_x immediately after P_x sends its j^{th} message of phase k.

$g_{x,j}^{(k)} =$ the state of P_x immediately before P_x receives its j^{th} message of phase k.

If P_x does not receive (send) j messages in phase k, then $r_{x,j}^{(k)}$ and $g_{x,j}^{(k)}$ ($s_{x,j}^{(k)}$ and $q_{x,j}^{(k)}$) are undefined. Note that the state variables are parameterized by messages sent and received, not by transitions made. For example, if $q_{x,j}^{(k)} =$ IN, then $q_{x,j-1}^{(k)} =$ HEADS.

The following lemma establishes that * messages effectively delimit phase boundaries.

Lemma 3.1: $s_{x,j}^{(k)} = r_{x+1,j}^{(k)}$.

Proof: Since messages sent by P_x are exactly those received by P_{x+1}, it suffices to show that P_x and P_{x+1} agree on phase boundaries. Notice from figure 1 that P_{x+1} enters a gate (a phase boundary) precisely when it receives a * message. But P_x, having sent the * message, cannot send any

further message without first passing through a gate. Nor can P_x enter a gate without sending a * message. □

The following lemma is immediate from figure 1. It points out that within a phase, as long as a processor is undecided, its communication alternates between output and input messages.

Lemma 3.2: If P_x is active in phase k and $q_{x,j}^{(k)}$ is defined, then $q_{x,j}^{(k)} = g_{x,j}^{(k)}$.

It follows from lemma 3.1 that each phase can be considered in isolation. Consider an arbitrary phase k with $m > 1$ active processors. Since passive processors merely forward messages, they can be ignored for the following lemmas, and the ring assumed to have only m processors, $P_0,\ldots,P_{m-1}$. In the remainder of this subsection, the superscript, (k), is omitted, and variables are assumed to describe the k^{th} phase.

The next lemma establishes that the processors cannot all become passive.

Lemma 3.3: If $q_{x,j}$ = OUT, then there exists w and i such that $q_{w,i}$ = IN.

Proof: Choose the smallest i such that $s_{v,i} = *$, over all processors P_v. Let P_y be some processor for which i is minimized. Now either $q_{y,i}$ = IN or $q_{y,i}$ = OUT. In the first case nothing remains to be proved. In the second case, it follows that $q_{y,i-1}$ = TAILS, $s_{y,i-1} = t$, and $r_{y,i-1} = h$. Let C be the class of t-messages which were the $(i-1)^{th}$ messages sent by some processor. P_y sends a message of class C, but does not receive one, since $r_{y,i-1} = s_{y-1,i-1} = h$. Since there can be at most n messages in class C, and all are eventually delivered, some processor, P_w, must receive a class C message without sending one. The transition from OUT to OUT cannot correspond to the reception of a class C message, by the minimality of i. The transition from TAILS to FLIP pairs the absorption of one message in C with the production of one on the previous transition. The only remaining transitions compatible with receiving a t-message lead to state IN. Therefore P_w must be in state IN after receiving its i^{th} message, and hence $q_{w,i}$ = IN.

□

The safety properties of the attrition procedure follow from lemma 3.3 and the next lemma. At any point during the execution, let $N(*)$ be the number of * messages awaiting delivery. Let $N(\text{OUT})$ and $N(\text{IN})$ be the number of processors in states OUT and IN respectively.

Lemma 3.4: $N(*) = N(\text{IN}) + N(\text{OUT})$ at every point during the execution.

Proof: The equation holds initially and is preserved by every transition. □

Corollary 3.5: If any processor reaches phase $k+1$, then some processor is active in phase $k+1$.

Proof: In order for any processor to reach a gate, some processor much reach IN or OUT. By lemma 3.3, some processor P_w, reaches IN. As long as P_w remains at IN, there is an undelivered * message, which must move around the ring, eventually reaching P_w and causing P_w to enter the active gate. □

Corollary 3.6: The attrition procedure cannot deadlock.

The next lemma leads to a bound on the number of phases in any computation of the attrition procedure.

Lemma 3.7: Suppose $q_{x,j} = \text{IN}$. Let y be the first number in the list $x-1, x-2, \ldots$ (counting modulo m), such that $q_{y,j} = \text{IN}$. (Such a y must exist since x occurs in the list.) Then there is a $w \in \{y+1, y+2, \ldots, x-1\}$ such that $q_{w,j} \in \{\text{OUT, P-GATE}\}$.

Proof: If $q_{x,j} = \text{IN}$, then $r_{x,j-1} = s_{x-1,j-1} = t$. So $q_{x-1,j-1} = g_{x-1,j-1} = \text{TAILS}$. Suppose $r_{x-1,j-1} \neq t$. After receiving $r_{x-1,j-1}$, P_{x-1} moves from TAILS to either OUT or P-GATE, and $w = x-1$ satisfies the lemma.

Suppose, instead, that $r_{x-1,j-1} = t$. Then $q_{x-2,j-1} = \text{TAILS}$. Again, if $r_{x-2,j-1} \neq t$, then $w = x-2$ satisfied the lemma. Otherwise the search continues through $x-3, x-4, \ldots, y+1$ until a w is found such that $q_{w,j-1} = \text{TAILS}$ and $r_{w,j-1} \neq t$. Such a w must exist since:

$$q_{y,j} = \text{IN} \Rightarrow q_{y,j-1} = \text{HEADS} \Rightarrow s_{y,j-1} = h \Rightarrow r_{y+1,j-1} = h$$

so some first non-t message $r_{w,j-1}$ must be encountered among $r_{x-1,j-1}, r_{x-2,j-1}, \ldots, r_{y+1,j-1}$. That

w satisfies the lemma. □

Corollary 3.8: At least half of the active processors at phase k are passive at phase $k+1$.

Proof: Lemma 3.7 associates with each processor P_x which remains active, a distinct processor P_w which becomes passive. The same P_w is not associated with two different P_{x_1} and P_{x_2} since $q_{w,j}, q_{w,j'} \in$ {OUT,P-GATE} implies $j = j'$. □

Corollary 3.9: There are at most $\lfloor \log c \rfloor$ phases during which more than one processor is active, where c is the number of candidates for leadership.

If there remain more than one active processor in phase k, then with probability one, at least two of them will eventually produce opposite coin flips, thus resulting in a transition to phase $k+1$. This observation, together with corollary 3.9 ensures that the final requirement for correctness is satisfied.

3.1.3. Complexity analysis

Recall that the last phase of the attrition procedure, when only one active processor remains, is an infinite loop broken by the intervention of the solitude verification algorithm. Therefore the complexity of concern for the attrition procedure is the expected number of bits sent, up to but not including the last phase. Corollary 3.9 establishes that there can be at most $\lfloor log\ c \rfloor + 1$ phases. It remains to bound the expected number of messages sent per phase.

The following random variables over computations are needed:

$$M_x^{(k)} = \begin{cases} 0 & \text{if only one processor is active in phase } k\text{, or if phase } k \text{ is not reached} \\ m & \text{if more than one processor is active in phase } k\text{, and processor } P_x \text{ sends } m \text{ messages during phase } k. \end{cases}$$

$M_x^{(k)}$ is defined for both active and passive processors, x, and for all integers $k \geq 1$.

Lemma 3.10: $Pr(M_x^{(k)} \geq m) \leq 2^{2-m}$.

Proof: The lemma is trivial if k is greater than or equal to the number of the last phase. Therefore suppose that there are two or more active processors in phase k. It is sufficient to consider the active processors only, since each passive processor sends the same number of messages as its nearest active predecessor.

The lemma is proved by induction on m. The case $m = 2$ is trivial. If, in phase k, P_x reaches state FLIP s times, then P_x sends $s+1$ messages. But whenever P_x reaches FLIP, there is a probability of at least $\frac{1}{2}$ that it will not return to FLIP in the same phase, since its coin toss is random relative to that of its nearest active predecessor. The induction follows immediately. □

Corollary 3.11: $E(M_x^{(k)}) \leq 3$.

Corollary 3.12: The expected number of bits communicated by the attrition procedure up to the last phase is at most $6n \lfloor \log c \rfloor$.

Proof: The total number of messages sent up to the last phase is given by $\sum_{x=0}^{n-1} \sum_{k \geq 1} M_x^{(k)}$. By corollaries 3.9 and 3.11 this has an expected value of at most $3n \lfloor \log c \rfloor$. Since there are only 3 distinct messages used, each can be encoded in 2 bits yielding an expected bit complexity of at most $6n \lfloor \log c \rfloor$. □

3.1.4. Interleaving properties

The attrition procedure lends itself naturally to being interleaved with a solitude verification algorithm. Attrition messages can simply be annotated with solitude verification messages. The * messages which serve to delimit phase boundaries are interpreted as solitude verification restart signals. The interleaved algorithm proceeds exactly like attrition until the last phase, at which point the absence of * messages allows solitude verification to run to completion. This is summarized by:

Theorem 3.13: Any conservative solitude verification algorithm of complexity $f(n)$ can be combined with the attrition procedure to yield a leader election algorithm of complexity $O(n \log c + f(n))$.

3.2. Solitude Verification Algorithms

In the absence of any information about the ring, solitude verification with certainty is impossible. Therefore solitude verification algorithms must use specific ring information to verify that there is a sole active processor. Two cases are considered here.

1. Each processor P_x has a distinct identifier I_x, consisting of a string such that if $x \neq w$, then I_x is not a prefix of I_w.
2. Each processor knows the size of the ring to within a factor of two.

These assumptions are as weak as possible in the sense that if processors identities can appear at most twice or the size of the ring is not known to within a factor of 2, solitude verification remains impossible. Though solitude verification is all that is required for leader election, the algorithms are, in fact, solitude detection algorithms.

3.2.1. Solitude detection with distinct identifiers

Suppose each processor, P_x, has a distinct identifier, I_x, which is not the prefix of any other identifier in the ring. Processor P_x uses an internal string variable J_x which is initialized to the empty string. Each initiator alternately sends the j^{th} bit of its own identifier and receives the j^{th} bit of its nearest active predecessor, which it appends to J_x. Thus P_x builds up in J_x the identifier of its nearest active predecessor. If I_x contains m bits then after receipt of at most m bits P_x can declare, by comparing J_x and I_x, whether or not it is alone. Because no identifier is a prefix of any other identifier, P_x can never falsely claim solitude.

Theorem 3.14: If processors have distinct m bit identifiers, then conservative and deterministic solitude detection can be achieved with distributive termination using at most $O(mn)$ bits.

3.2.2. Solitude detection when the size of the ring is known approximately

Suppose that distinct identifiers are not available, but each processor knows the size n of the ring. In this case, a non-conservative algorithm for determining solitude has each initiator send a counter which is incremented and forwarded by each passive processor until it reaches an initiator, P_x. By comparing the received counter with n, P_x knows whether or not it is alone. This algorithm can be transformed into a conservative solitude detection algorithm without any increase in bit communication complexity.

Each processor P_x, whether active or passive, maintains a counter c_x, initialized to 0. Let $d_x > 0$ denote the distance from P_x to its nearest active predecessor. The algorithm maintains the invariant:

if P_x has received j bits then $c_x = d_x \ mod \ 2^j$.

Then if P_x reaches a state where $c_x = n$, there must be $n-1$ passive processors preceding P_x, so P_x can conclude that it is the sole initiator. It remains to describe the strategy necessary to maintain the invariant.

Initiators first send 0. Thereafter, all processors alternately send and receive bits. If P_x is passive, then P_x is required to send the j^{th} low order bit of d_x as its j^{th} message. Initiators continue to send 0. Suppose a processor, P_y, has the lowest order j bits of d_y in c_y. A simple inductive argument shows that when P_y receives its $(j+1)^{st}$ message (by assumption the $(j+1)^{st}$ bit of $d_y - 1$), it can compute the first $(j+1)$ bits of d_y and thus can update the value of c_y to satisfy $c_y = d_y \ mod \ 2^j$.

In the previous algorithm, it was assumed that n is known exactly. Suppose instead, that each processor knows a quantity N, such that $N \leq n \leq 2N-1$. Then there can be at most one gap of length N or more between neighbouring active processors. Thus any gap of less than N confirms non-solitude and any processor detecting a gap of N or more can determine solitude by initiating a single checking round. (For the purposes of leader election, it is sufficient for any active processor that detects a gap of N or more to declare itself the leader, since it has confidence that no other processor can do the same). Thus, the algorithm can be used when n is

known to within a factor of less than two.

Theorem 3.15: If each processor knows a value N such that the ring size n satisfies $N \leq n \leq 2N-1$, then conservative and deterministic solitude detection can be achieved using at most $O(n \log n)$ bits. □

3.3. Time Complexity of the Leader Election Algorithm

As is usual for randomized algorithms for asynchronous models, the *time complexity* of an algorithm is the expected number of unit time intervals before the algorithm terminates, under the assumption that messages travel each communication link in unit time, local processing is instantaneous, and the algorithm proceeds with maximum possible synchrony.

The time complexity of [DKR] is $O(n)$. Our leader election algorithm, as described, requires $O(n \log n)$ time steps even for just the final phase, when verification bits are sent and received one at a time by the sole remaining active processor. It therefore might appear that the algorithm achieves an improvement by a factor of $O(\log n)$ in bits, only at the expense of an additional factor of $O(\log n)$ in time over other leader election algorithms. But a slight alteration in the attrition algorithm reduces the time complexity of the final phase from $O(n \log n)$ to almost linear. Suppose that the i^{th} message of a phase of attrition contains a package of $f(i)$ annotated coin flips rather than a single coin flip as previously described. Let $f(1) = 1$ and $f(i) = 2^{f(i-1)}$ for $i > 1$. The probability that a processor sends at least k packages of coin flips in a given phase is no more than $2^{-(f(1)+\quad+f(k-1))}$. Hence the expected number of coin flip bits sent by an arbitrary processor in a given phase is no more than $\sum_{k=1}^{\infty} \frac{f(k)}{2^{f(1)+\quad+f(k-1)}} = \sum_{k=1}^{\infty} \frac{1}{2^{f(1)+\quad+f(k-2)}} = O(1)$. So the expected number of annotated coin flip bits sent by a processor in a phase is also constant. With this packaging, in the final phase of leader election, the remaining active processor will initiate only $O(\log^* n)$ messages which propagate around the ring. $O(n \log^* n)$ time steps are required to confirm solitude when ring size is known without any increase in communication bit complexity. Similar packaging

yields $O(n \log^* m)$ time steps when identifiers of length m are used to confirm solitude.

It remains to analyse the expected time for attrition up to the last phase. Let a *round* be one single exchange of coin tosses for all active processors on the ring. It can be shown that there are $O(log\ n)$ rounds expected through the duration of attrition. By analysing the time for each such round separately and summing, a ceiling on the expected time for attrition (up to the last phase) results. The effects of overlapping rounds and packaging only serve to reduce the expected time. But the time required for an isolated round is proportional to the longest gap between active processors in that round. It can be shown that the expected length of the longest gap between active processors in the i^{th} round is no more than 2^i. Thus the expected time for all rounds, even assuming no overlap, is bounded by $k \cdot \sum_{i=1}^{O(log\ n)} 2^i = O(n)$ steps. Combining these results for attrition and solitude verification yields:

Theorem 3.16: The leader election algorithm which results from combining the attrition procedure of section 3.1 and the solitude verification algorithm of section 3.2.1 (section 3.2.2), can be adapted to have time complexity $O(n \log^* m)$ ($O(n \log^* n)$) without any increase in the order of the communication complexity.

4. Lower Bounds on the Complexity of Solitude Verification

This section provides lower bounds on the number of bits of communication required for solitude verification. Two cases, paralleling those considered in section 3.2, are studied. In each case lower bounds are developed which show that the algorithms discussed earlier are essentially optimal.

The lower bounds apply even to non-deterministic distributed algorithms. In order to facilitate the proofs, a formal model of non-deterministic algorithms and their computations is first presented in subsection 4.1. In subsection 4.2, the solitude verification and leader election problems are defined in terms of the formal model. That section also contains an $O(n)$ reduction

from solitude verification to leader election within the model, implying that even the general non-deterministic lower bounds for solitude verification translate to the same lower bounds for leader election. Techniques and tools common to the proofs are grouped together in subsection 4.3. Subsections 4.4, and 4.5 study respectively, the cases where processors have distinct identifiers, and processors know the approximate size of the ring.

4.1. Model of Computation

A *labelled ring* is a unidirectional n-ring of processors $P_0,...,P_{n-1}$ where each processor has associated with it a (not necessarily unique) *label* from ID $\times$ {initiator,non-initiator} where ID denotes some fixed universe of identifiers. Various assumptions about the universe ID reflect the degree to which processors are distinct. The second label field is used to distinguish processors that initiate a computation from those that only participate in response to other initiators.

A *history* is a finite sequence of bits each of which is marked as either an input or an output bit. Each marked bit is a *(communication) event*. A history is *initiating (non-initiating)* if and only if its first communication event is an output (input).

Let H_I be the set of all initiating histories, and H_N be the set of all non-initiating histories. A *(non-deterministic) process* $\mathbf{P}$ is a pair (H,f) where $H \subseteq H_I \bigcup H_N$ and f is a function $f : H \rightarrow \{yes, no\}$. Given $h \in H$, $f(h)$ is the *final state* of h in $\mathbf{P}$. A process $\mathbf{P} = (H,f)$ is *initiating* if H is a subset of H_I and *non-initiating* if H is a subset of H_N. Intuitively, a process provides a choice of histories that describe possible input-output behaviour of a processor.

A *(distributed) algorithm* α is a mapping from labels to processes that assigns initiating processes to initiators and non-initiating processes to non-initiators. In this way it is ensured that indistinguishable processors (processors with identical labels), receive the same process.

Let $\mathbf{P}_i = (H_i, f_i)$ denote the process assigned to processor P_i by distributed algorithm α in a labelled n-ring $R = P_0,P_1,...,P_{n-1}$. The sequence $C = h_0,h_1,...,h_{n-1}$ defines a *computation of* α *on* R if $h_i \in H_i$, for $0 \leq i \leq n-1$.

Given a computation $C = h_0, h_1, \ldots, h_{n-1}$, the communication events that make up history h_i can be partially partitioned, relative to C, in a natural way into *communication intervals*. The 0^{th} communication interval of history h_i is the empty string. If h_i is initiating, then the k^{th}, $k \geq 1$, communication interval in h_i is the largest interval of events following the $(k-1)^{st}$ interval whose sequence of input communication events matches the sequence of output communication events in the $(k-1)^{st}$ communication interval of h_{i-1}. If h_j is non-initiating, then the k^{th}, $k \geq 1$, communication interval of h_j is the largest interval of events following the $(k-1)^{st}$ interval whose sequence of input communication events matches the sequence of output communication events in the k^{th} communication interval of h_{j-1}. If there is at least one initiator, the communication intervals of non-initiators are well defined. Those communication events which are assigned to a communication interval are *realizable*. A history is *realizable* if all of its communication events are realizable. Note that realizable events and realizable histories are defined relative to a computation. $h_i \backslash C$ denotes the initial subsequence of h_i that is realizable relative to C and is referred to as a *truncated history*. $h_i \backslash\backslash C$ denotes the partition of $h_i \backslash C$ into communication intervals and is referred to as a *scheduled history*.

If the full history, h_i, is realizable relative to C then the computation C leaves processor P_i in state $f_i(h_i)$, (the final state of h_i in $\mathbf{P}_i$), otherwise C leaves P_i in state *undecided*.

The computation C is *weakly decisive* if for some i, $0 \leq i \leq n-1$, $h_i \backslash C = h_i$, that is, if some history in the computation is realizable. C is *decisive* if for every i, $0 \leq i \leq n-1$, $h_i \backslash C = h_i$, that is, every history in the computation is realizable.

Let W, (D) be the set of weakly decisive (decisive) computations of an algorithm α on a ring R. Let T be a predicate defined on computations of α. α *solves* the problem corresponding to T on R with *distributive termination* if and only if $\forall w \in W, T(w)$. α solves the problem corresponding to T on R with *non-distributive termination* if and only if $\forall d \in D, T(d)$. These formal definitions of distributive and non-distributive termination capture the corresponding informal notions. Notice that distributive termination of an algorithm α is a stronger requirement than non-distributive termination of α because for distributive termination a condition must be

met by every weakly decisive computation of α, whereas for non-distributive termination, the condition need only hold for fully decisive computations. A decision reached by a processor, P_i, in a computation of a distributively terminating algorithm is final. It cannot be subsequently changed by receipt of messages from processors that have not terminated. In the formal model, this corresponds to the fact that if a non-deterministically chosen history h_i is fully realized, then P_i's state, $f(h_i)$, is final. P_i cannot detect that other histories were not realized. On the other hand, when a decision is reached by a processor, P_i, in a computation of a non-distributively terminating algorithm, the decision is only necessarily final if P_i receives no further messages. In the formal model, this corresponds to all non-deterministically chosen histories being fully realized. Only decisive computations are used in the proofs of this paper. Thus the results are true even for non-distributively terminating algorithms.[2]

The *(communication) complexity* of a computation $C = h_0,\ldots,h_{n-1}$ of algorithm α on labelled n-ring R is defined as $\sum_{i=0}^{n-1} |h_i \backslash C|$ where $|h_i \backslash C|$ is the length of $h_i \backslash C$ (that is the number of realizable communication events in h_i). The complexity of a computation is the number of bits that could be sent and received until the communication either becomes inconsistent or terminates.

4.2. The Problems

A weak version of leader election requires that at most one processor be left, at termination, in a distinguished final state. Formally, the *weak leader election* predicate, L, is defined over computations by: $L(C) =$ (there exists at most one h_i in C such that h_i is initiating, and $h_i \backslash C = h_i$ and $f_i(h_i) = yes$). A weak version of solitude verification requires that if any initiator is left in a distinguished final state at termination, then that initiator is the only initiator of the computation. Formally, the *weak solitude verification* predicate, V, is defined over computations by: $V(C) =$ (If there exists h_i in C such that h_i is initiating and $h_i \backslash C = h_i$ and $f_i(h_i) = yes$, then h_i is the only initiating history in C). Thus, using the definitions of the

[2]The distinction between the two types of termination is required for [AAHK1].

previous subsection, an algorithm α solves the weak leader election (weak solitude verification) problem on a ring R, with non-distributive termination if and only if, for every decisive computation C of α on R, $L(C)$ ($V(C)$) is true.

Notice that an algorithm has to meet only a rather weak requirement in order that it be said to solve one of the problems. For example, deadlocking computations are tolerated. This only serves to strengthen the lower bound results that follow. Conclusions are drawn about the number of bits that must be transmitted in any computation of an algorithm which succeeds in verifying solitude, (a *successful* computation), even if the algorithm solves the problem in only this weak sense. A *successful* solitude verification computation has exactly one initiating history, h_i, and h_i is realized and has final state *yes*. A *successful* leader election computation has exactly one of the initiators left in final state *yes*.

Section 2 contains a description of how a randomized, distributively terminating algorithm for electing a leader from among all processors on the ring can be converted, using an additional $O(n)$ bits of communication, to an algorithm for solitude detection (and hence solitude verification). In fact this $O(n)$ reduction holds even for the non-deterministic, non-distributively terminating model, and for the weak versions of the two problems. Initiators, wishing to determine their solitude, alert the ring by propagating a wake-up message. All processors, having been alerted, non-deterministically choose whether or not to be candidates for leadership, and run the weak leader election algorithm. A candidate remaining in contention guesses if and when the leader election algorithm terminates with itself as leader. At this time the elected leader circulates a single constant length message to determine whether one or more than one original initiator was present. A final round announces the result. Because the portion of this algorithm following leader election is deterministic, the reduction converts successful computations of leader election to correct and decisive computations of solitude detection. Unsuccessful leader election can happen if either no processor chose to be a candidate or if the weak leader election algorithm left all processors in a non-leader state. But in both of these cases the corresponding solitude detection computation deadlocks, satisfying weak solitude verification. Finally, if a candidate guesses erroneously that it is the sole remaining contender before weak leader election has

terminated (non-distributively), then eventually the leader election algorithm must correct this error. So the resulting solitude verification computation is also eventually alerted to the error, and correctly achieves non-distributive termination. Thus non-linear lower bounds for computations that verify solitude translate to lower bounds for computations that elect a leader even for this general model. Notice that non-determinism allows this reduction to hold for the general version of leader election when a leader is elected from c candidates rather than from the fixed configuration of all processors on the ring. Thus the lower bounds for solitude verification imply lower bounds for a best case configuration of candidates for leadership.

4.3. Tools

In the lower bound results that follow, two techniques are used to create new rings and (weakly) decisive computations on them from existing ones. Let $C = h_0,h_1,...,h_{n-1}$ be a computation of some algorithm α on the ring $R = P_0,P_1,...,P_{n-1}$. Suppose that $h_i \backslash\backslash C = h_j \backslash\backslash C$ and $h_{i+1},...,h_j$ contains no initiators. Consider the new sequence $C' = h_0,...,h_i,h_{j+1},...,h_{n-1}$ formed by removing $h_{i+1},...,h_j$ from C. Then C' is a computation of α on $R' = P_0',...,P_i',P_{j+1}',...,P_{n-1}'$ where P_k' has the same label as P_k. In addition, if C is a (weakly) decisive computation of α on R then C' is a (weakly) decisive computation of α on R'. This process of forming C' on R' from C on R when $h_i \backslash\backslash C = h_j \backslash\backslash C$ while retaining all initiators is referred to as *collapsing*.

Let $C_1 = h_0,...,h_{n-1}$ and $C_2 = l_0,...,l_{m-1}$ be computations of an algorithm α on the two rings $R_1 = P_0,...,P_{n-1}$ and $R_2 = Q_0,...,Q_{m-1}$. Suppose that $h_i \backslash\backslash C_1 = l_j \backslash\backslash C_2$ for some i and j. Consider the sequence $C' = h_0,...,h_i,l_{j+1},...,l_{m-1},l_0,...,l_j,h_{i+1},...,h_{n-1}$ formed by combining C_1 and C_2. Then C' is a computation of α on $R' = P_0',...,P_i',Q_{j+1}',...,Q_{m-1}',Q_0',...,Q_j',P_{i+1}',...,P_{n-1}'$ where P_k' (Q_k') has the same label as P_k (Q_k). Also, if C_1 or C_2 is a weakly decisive computation of α on R_1 or R_2 respectively, then C' is a weakly decisive computation of α on R'. If both C_1 and C_2 are decisive computations of α on R_1 and R_2 respectively, then C' is a decisive computation of α on R'. This process of forming C' on R' by combining C_1 on R_1 and C_2 on R_2 is referred to as

splicing. The special case of splicing a computation to itself is called *doubling*.

In a computation $C = h_0,h_1,...,h_{n-1}$, it may happen that a scheduled history has only input bits in some communication interval before its last. This corresponds to P_i acting as a temporary *sink* absorbing bits before the end of the computation but not sending any on. As a result, successors of P_i may have some empty communication intervals. However, if there are k initiators, there can be at most $k-1$ sinks. In particular, if a computation has only one initiator, then each communication interval of each scheduled history must contain output bits until computation ceases. Therefore, when there is one initiator, $h_i \backslash\backslash C$ is encoded in $3 \mid h_i \backslash C \mid$ bits by using 2 extra bits per communication event — one to describe the type of event and one to denote communication interval boundaries. $\mid h_i \backslash\backslash C \mid$ denotes the length of this encoding. The fact that $\mid h_i \backslash\backslash C \mid = 3 \mid h_i \backslash C \mid$ when there is one initiator is used frequently, and it is summarized in the following lemma.

Lemma 4.1: Let $C = h_0,h_1,...,h_{n-1}$ be a computation with exactly one initiator. Then each scheduled history of C can be encoded so that its length is 3 times the length of the corresponding truncated history.

The proofs to follow take advantage of the fact that any collection of k distinct binary strings contain at least $\frac{k \log k}{2}$ bits for $k > 3$. The following lemma is also needed.

Lemma 4.2: Let $C = h_0,h_1,...,h_{n-1}$ be a computation with exactly one initiator. If the complexity of C is less than $cm \log m$, with $m \leq n$, then there exist i and j, $0 < j-i < m^{12c}$ such that $h_i \backslash\backslash C = h_j \backslash\backslash C$.

Proof: Let $\hat{C} = h_0\backslash\backslash C, h_1\backslash\backslash C, ..., h_{n-1}\backslash\backslash C$, be the sequence, in the standard encoding, of the scheduled histories corresponding to C. Let k be the maximum integer such that every subsequence of $\hat{C}$ containing k scheduled histories has each scheduled history distinct. The encoding of a subsequence of k distinct scheduled histories must contain a total of at least $\frac{k \log k}{2}$ bits. But $\hat{C}$ can be decomposed into $\lceil n/2k \rceil$ disjoint subsequences each with at least k scheduled his-

tories. Since the complexity of C is less than $cm \log m$, at least one of these subsequences must have less than $\frac{cm \log m}{\lceil n/2k \rceil} \leq 2ck \log m$ realizable communication events and hence can be encoded by at most $3 \cdot 2ck \log m$ bits. Hence, $\frac{k \log k}{2} < 3 \cdot 2ck \log m$. Thus $k < m^{12c}$. □

The lower bound arguments both have similar structure. It is assumed that an algorithm α exists that solves solitude verification on a ring R with non-distributive termination and that α has a successful computation C with small communication complexity. It follows, by some combination of collapsing and splicing, that C can be transformed into a decisive computation C' of α on a different ring R', in which more than one initiator terminates in state *yes*. Thus α does not solve solitude verification on R'.

4.4. Distinct Identifiers

The objective of this subsection is to characterize the complexity of algorithms that solve the solitude verification problem on rings of processors with distinct identifiers chosen (otherwise arbitrarily) from a set ID of size s. Let α be any algorithm that solves weak solitude verification on all n-rings with distinct identifiers chosen from ID. There are $\binom{s}{n}$ possible identifier sets for such an n-ring. Suppose that for each of these sets, there is at least one permutation of the identifier set, such that for an n-ring labelled with this permutation and with exactly one initiator, P, α has some successful computation. α is then said to *non-trivially* solve solitude verification for n-rings with identifiers chosen from ID.

Theorem 4.3: Let α be any algorithm that non-trivially solves the weak solitude verification problem on rings of between N and $2N$ processors with distinct identifiers chosen from a universe ID of size $s \geq 2N$. Then the average, over all $\binom{s}{N}$ identifier sets L of size N, of the minimum communication complexity of successful computations of α on any ring R with label set L is $\Omega(N \log (s/2N))$.

Proof: Let L be any subset of ID of size $n = N$ and let R be a ring of n processors, with exactly

one initiator and with distinct identifiers chosen from L. Let $C = h_0,h_1,...,h_{N-1}$ be a computation on R. C is a *cheap computation* if it has complexity less than $\frac{n \log (s/2N)}{3}$. If a cheap C exists, the ring R and the identifier set L are also said to be cheap. If C is cheap, then at least one of the truncated histories $h_0\backslash C,h_1\backslash C,...,h_{n-1}\backslash C$ must have length less than $l = \frac{\log (s/2N)}{3}$. Choose any such truncated history and call it the cheap (truncated) history associated with C and indirectly associated with R and L. Now suppose that α has the property that for at least one-half of the $\binom{s}{n}$ possible choices for L, there exists a cheap successful computation of α and therefore an associated cheap truncated history. Among all the partitions of ID into s/n subsets of size n, at least one such partition must have among its subsets at least $s/2n$ cheap identifier sets. Therefore there exist $s/2n$ disjoint, cheap labellings with corresponding cheap successful computations. But there are fewer than $2^{3l} = s/2N = s/2n$ distinct cheap scheduled histories in total. So some cheap scheduled history must be associated with successful computations of α on two rings with disjoint sets of identifiers. If these computations are spliced at their common history, the result is a decisive computation of α on a ring of size $2N$, whose processors all have distinct identifiers. However, this computation leaves two processors in the final state *yes* contradicting the correctness of α. □

Corollary 4.4: Let α be an algorithm that meets the conditions of theorem 4.3. If the size s of the universe ID is $\Omega(N^{1+\epsilon})$ for some $\epsilon>0$, then the average, over all identifier sets L of size N, of the minimum communication complexity of successful computations of α on any R with label set L is $\Omega(N \log s)$.

4.5. Ring Size Known Approximately

If ring size n is to be used to verify solitude, it must be known to within a factor of two. The objective of this subsection is to characterize the complexity of computations that verify solitude, as a function of processors' uncertainty of ring size within this limit.

Theorem 4.5: Let α be any non-distributively terminating algorithm that solves the solitude

verification problem on the class of all rings of size n, where $n \in [N, N+\Delta]$ for some N and $0 < \Delta < N$. Then any successful computation of α on any ring in the class must have communication complexity $\Omega(N \log \Delta)$.

Proof: Let R be any ring of n processors, $N \leq n \leq N + \Delta$, exactly one of which, say P_0, is an initiator. Let $C = h_0, h_1, \ldots, h_{n-1}$ be any successful computation of α on R confirming the solitude of P_0. (So $f_0(h_0) = yes$.) Let $\epsilon = \frac{\log \Delta}{\log N}$. Suppose that the communication complexity of C is less than $(N \log \Delta - 2N) / 12 = (\epsilon N \log N - 2N) / 12$. By lemma 4.2, there exist i and j such that $0 < j - i < N' = \Delta$ and $h_i \backslash\backslash C = h_j \backslash\backslash C$. Consider the new computation $h_0, \ldots, h_i, h_{j+1}, \ldots, h_{n-1}$ formed by removing $S = h_{i+1}, \ldots, h_j$ and apply repeated collapsing to each subsequence $H_1 = h_0, \ldots, h_i$ and $H_2 = h_{j+1}, \ldots, h_{n-1}$ separately until these subsequences each is composed of distinct scheduled histories. Let the resulting computation be $C' = h_0', \ldots, h_l', h_{l+1}', \ldots, h_{m-1}'$, where $h_0 \backslash\backslash C' = h_0 \backslash\backslash C$, $h_l \backslash\backslash C' = h_i \backslash\backslash C$, and $h_{m-1} \backslash\backslash C' = h_{n-1} \backslash\backslash C$. Since $h_0', \ldots, h_l'$ and $h_{l+1}', \ldots, h_{m-1}'$ are sequences of distinct scheduled histories, their combined communication complexity is at least $\frac{(l+1)\log(l+1)}{6} + \frac{(m-l-1)\log(m-l-1)}{6}$ which is at least $\frac{m}{6}\log(\frac{m}{2})$. Thus their combined length m must not exceed $\frac{N}{2}$, since otherwise, the assumption on the complexity of C is violated. Therefore C' can be doubled to form a new computation, C'', of α on a ring R' of size $2m < N$ processors. But since the subsequence S, which was originally removed, has length $s < \Delta$, there exists k such that $N \leq 2m + ks \leq N + \Delta$. Thus k copies of S can be spliced into C'', after either scheduled history identical to $h_i \backslash\backslash C$, forming a new decisive computation of α on a ring of size $n' \in [N, N + \Delta]$ which contains 2 scheduled histories identical to h_0. Thus 2 processors conclude solitude, contradicting the correctness of α for rings of size $n \in [N, N + \Delta]$. □

It can be shown that theorem 4.5 holds even if the algorithm α is required to work correctly only on the class of rings in which processors have identifiers, and no processor identifier appears more than twice. If all processor identifiers are known to be distinct then the results are different,

as was shown in section 4.4.

5. Conclusions

5.1. Technical Results

The inherent communication complexity, measured in terms of the expected number of bits, of electing a leader in a ring of processors has been identified to within constant factors for two cases. When all processors know the ring size to be within some interval $[N_l, N_u]$ and all processors have distinct identifiers drawn from some set of size $s \geq N_u^{1+\epsilon}$, where $\epsilon>0$, then for all n satisfying $N_l \leq n \leq N_u/2$, the average, over all n-rings, of the expected bit complexity of randomized leader election is $\Theta(n \log s)$. On the other hand, if the ring size is known to be in some interval $[N_l, N_u]$ where $N_l + N_l{}' \leq N_u < 2N_l$, for some $\epsilon<0$, and processor identifiers are not necessarily distinct then, for all n satisfying $N_l \leq n \leq N_u$, the expected bit complexity of randomized leader election is $\Theta(n \log n)$.

The results for leader election stem from bounds on the complexity of two more primitive processes called attrition and solitude verification. The identification of these subproblems and the clarification of their relationship to leader election is one of the important contributions of this paper. Efficient conservative solitude verification algorithms that exploit known properties of a ring can be combined with the randomized attrition procedure described in section 3.1 to provide new efficient leader election algorithms. Solitude verification is of equal interest for its role in the proof of lower bounds for leader election. For all solitude verification computations of concern there is only one initiator, which considerably simplifies the analysis. This is reflected in the strong lower bounds of section 4.

5.2. Related Issues

In addition to the specific technical contributions cited above the results of this paper shed light on a number of important issues in distributed computing. These are summarized under

three general headings below.

5.2.1. Global knowledge of ring

Suppose that all processors know that the ring size n lies in the interval $[N_l, N_u]$. If the processors are indistinguishable then deterministic leader election is impossible [A], even if $N_l = N_u$. Furthermore, if $N_u \geq 2N_l$ then even randomized algorithms cannot elect a leader among indistinguishable processors with certainty. However, if $N_u < 2N_l$, then randomized leader election can be achieved in $O(n \log n)$ expected bits. If, in addition, $N_l + N_l' \leq N_u$, for some $\epsilon > 0$, (i.e. the interval is not too small) then $\Omega(n \log n)$ bits are required to elect a leader among indistinguishable processors.

On the other hand, even if $N_l = 1$ and $N_u = \infty$, if processors have distinct identities chosen from a universe S of size s (which need not be known explicitly) then a leader can be elected with $O(n \log s)$ expected bits. In fact, assuming $N_u \geq 2N_l$ and $n \leq s/2$, $\Omega(n \log(s/n))$ bits are required to elect a leader with distinct identities from S.

5.2.2. Type of algorithm

The leader election algorithms described in this paper are all randomized. In fact, the solitude verification process is deterministic. The algorithms cannot deadlock. They all terminate distributively with probability 1 and elect a leader (or detect solitude) with certainty. Finally, with the exception of those modifications described in section 3.3, the algorithms are all conservative.

In contrast to the above, the lower bounds on solitude verification (and hence leader election) are proved on a non-deterministic model of computation. The model admits algorithms that may deadlock. Furthermore algorithms may communicate non-conservatively, may terminate non-distributively, and may, in the case of solitude verification, tolerate errors when there is only one initiator. The lower bounds state a minimum bit complexity of any computation that provides a certificate of solitude.

The juxtaposition of the algorithms and model of computation highlights a remarkable insensitivity for the problems and complexity measure studied in this paper, to the details of the underlying model of computation. This insensitivity is not preserved when the focus shifts to certain closely related problems [AAHK1,AAHK2].

5.2.3. Type of analysis

The solitude verification algorithms are analysed with respect to the worst case number of bits of communication. The lower bounds refer to the best case number of bits communicated by computations of algorithms that certify solitude.

The bulk of earlier results on leader election are concerned with message complexity. The leader election algorithms of this paper are competitive in this measure while improving upon earlier results by a factor of $\log n$ in the number of bits transmitted. While the obvious implementation of the leader election algorithms of this paper on a synchronous model makes them somewhat unattractive in terms of communication time, implementations exist, as described in section 3.3, which for all practical purposes make the algorithms comparable with earlier algorithms in this measure as well.

5.3. Extensions

The results of the present paper can be extended in two natural directions. First, the case where the ring size n is known exactly — a situation where the upper and lower bounds of this paper do not agree — can be explored in more detail. The solitude verification problem when n is known exactly is examined in [AAHK1]. In this case number theoretic properties of n can be exploited to improve upon the $O(n \log n)$ algorithm contained in this paper. With exact knowledge of ring size, there is a distinction between the complexity of distributively and non-distributively terminating versions of solitude verification. $\Theta(n \sqrt{\log n})$ bits are necessary and sufficient to achieve solitude verification with distributive termination. This becomes $\Theta(n \log \log n)$ bits for non-distributive termination. The upper bounds in this case are achieved

by non-deadlocking, deterministic algorithms, and the lower bounds by the same general models as used in this paper. The algorithms are non-conservative. If conservative solitude verification is required then $\Theta(n \log n)$ bits are necessary and sufficient [AAHK1].

This paper and its first companion paper are concerned with leader election and solitude verification when enough information is available to achieve certainty. When processor information is insufficient to confirm solitude with certainty, it is still possible to solve these problems probabilistically. [AAHK2] examines probabilistic solitude verification, that is, algorithms that are correct with probability at least $1 - \epsilon$. When there is no knowledge of ring size, the communication complexity of solitude verification with non-distributive termination is $\Theta(n \log \frac{1}{\epsilon})$ bits. (Distributive termination with probability $1 - \epsilon$ of correctness is impossible.) When ring size is known to be less than a bound N, then distributive termination can be achieved with complexity $O(n \sqrt{\log \frac{N}{n}} + n \log \frac{1}{\epsilon})$ bits. A matching lower bound is shown for rings of actual size no larger than $\frac{N}{2}$.

6. References

[A] D. Angluin, *Local and Global Properties in Networks of Processors,* Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (1980), pp.82-93.

[AAHK1] K. Abrahamson, A. Adler, L. Higham, D. Kirkpatrick, *Solitude Verification on Rings of Known Size,* in preparation, U. of British Columbia.

[AAHK2] K. Abrahamson, A. Adler, L. Higham, D. Kirkpatrick, *Probabilistic Solitude Verification on Rings,* in preparation, U. of British Columbia.

[B] J. Burns, *A Formal Model for Message Passing Systems,* TR-91, Indiana University, September 1980.

[DKR] D.Dolev, M. Klawe and M. Rodeh, *An O(n log n) Unidirectional Distributed Algorithm for Extrema Finding in a Circle,* J. Algorithms 3,3 (Sept. 1982), pp.245-260.

[FL] G. Fredrickson and N. Lynch, *The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring,* Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing (1984), pp.493-503.

[IR] A. Itai and M Rodeh, *Symmetry Breaking in Distributed Networks,* Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (1981), pp.150-158.

[Pa] J. Pachl, *A Lower Bound for Probabilistic Distributed Algorithms,* Research Report CS-85-25 (August 1985), University of Waterloo. Waterloo, Canada.

[Pe] G. Peterson, *An O(n log n) Unidirectional Algorithm for the Circular Extrema Problem,* Trans. Prog. Lang. Sys. 4,4 (1982), pp.758-762.

[PKR] J. Pachl, E. Korach and D. Rotem, *Lower Bounds for Distributed Maximum-finding Algorithms,* J. ACM 31,4 (Oct. 1984), pp. 905-918.