1 Introduction

Given a network of processors with distinct identifiers, the maximum finding problem is to find that processor with the maximum identifier. We present a new, efficient, and simple deterministic algorithm for the maximum finding problem on asynchronous unidirectional rings. Our algorithm sends fewer messages than all previously published algorithms for this model while its description and analysis are substantially simpler than that required of the most efficient previously published solution.

Maximum finding on asynchronous rings has been very well studied. The first deterministic algorithm for rings with distinct identifiers was presented in 1977 by LeLann [25]. It used $O(n^2)$ messages even on average for a unidirectional ring. In 1979, Chang and Roberts [14] improved the average case to $O(n \log n)$ messages but the worst case remained $O(n^2)$. Hirschberg and Sinclair [21], in 1980, solved the problem in $O(n \log n)$ messages for bidirectional rings and conjectured that this was not possible in the unidirectional case. In 1982, Peterson [28] and Dolev *et al.*[15] independently disproved this conjecture by finding a solution for the unidirectional version of the problem that used at most $2n \log n + n$ messages. A series of improvements followed. Dolev *et al.* brought the constant factor down to 1.5 with a combination of two techniques that required an extensive and complicated analysis. Then Peterson decreased it to less than 1.441, in a simple algorithm with an elegant analysis. Finally Dolev *et al.* applied their techniques to Peterson's algorithm to achieve a maximum finding algorithm with message complexity $1.356n \log n$.

Probabilistic solutions for asynchronous unidirectional and bidirectional rings with and without identifiers have also been proposed [22, 1, 20]. Lower bounds and impossibility results have been explored in a variety of papers [7, 13, 27, 16, 10, 2, 3, 12, 11]. These show the message complexity to be $\Omega(n \log n)$ for rings of size n on increasingly more powerful models and for increasingly more constrained versions of the problem. Taken together, the lower bound results imply that even in the average case, with or without identifiers and with or without randomization, maximum finding on a unidirectional or bidirectional ring is $\Omega(n \log n)$ even if the ring size, n, is known by the algorithm.

Several closely related problems also imply results for asynchronous maximum finding on rings [9, 26] or extend results to other networks [19, 6, 4, 5, 8, 17, 18, 23, 24].

Because of asynchrony, timing information cannot be exploited to advantage and hence all the algorithms for maximum finding on this model are *message-driven.* It is assumed only that all

messages are eventually delivered and, for each link, the order of delivery matches the order of transmission. It is often helpful to view a distributed algorithm by focusing on the messages as they travel around the network rather than on the processors as they send and receive messages. We adopt this perspective in this paper, so our descriptions of algorithms have the following general form. Initially each processor creates a *(message) envelope* and forwards it to its neighbour. When an envelope arrives at a processor, information from the envelope may be used to update processor information, and then either the contents of the envelope are updated, and it is forwarded, or the envelope is destroyed.

Section 2 develops our new algorithm from this perspective, and provides the analysis. We identify three ideas that, when appropriately combined, suffice for our algorithm. Two of these ideas are inspired by techniques used in previous algorithms for maximum finding. In Section 3, the "envelope perspective" is used to reinterpret these existing algorithms. This helps to illuminate the similarities and differences between them and our new algorithm. Some related ongoing research and open problems are mentioned in Section 4.

2 The new maximum finding algorithm

In this section we apply three observations to develop an efficient and simple deterministic maximum finding algorithm for asynchronous unidirectional rings. We begin with a basic algorithm and then consider two enhancements.

2.1 Leader election

Although the essential equivalence of leader election and maximum finding is folklore, the relationship has not been exploited for deterministic maximum finding on asynchronous unidirectional rings. Once elected, a leader can initiate one message envelope that travels the ring and keeps track of the maximum identifier. A second envelope announces the result. Thus, maximum finding reduces to leader election with at most $2n$ additional messages.

The advantage of electing a leader rather than finding the maximum directly arises because, for leader election, it suffices to isolate *any* one identifier. Preserving local maxima (as in previous algorithms) can be replaced by any process that eliminates identifiers while ensuring that not all

are eliminated. We start with a basic algorithm for leader election called LE.BASIC, and leave the conversion to maximum finding to the reader.

Initially each processor creates an *envelope* containing a *label* set to its own identifier, and a *round* set to 1, and forwards the envelope to its neighbour. (The label of an envelope remains unchanged as long as the envelope survives. The round will be incremented during the course of the algorithm.) Throughout the algorithm, each processor stores the label and the round of the last envelope it sent. When an envelope with an odd (respectively, even) round number arrives at a processor, it is destroyed only if it has the same round number as the processor and a larger (respectively, smaller) label. If the round numbers and the labels both match, then the algorithm terminates and the receiving processor is the leader. If the round numbers match and the envelope is not destroyed, then its round number is incremented and the updated envelope is forwarded. Finally, if the round numbers do not match then the envelope is forwarded unchanged.

Figure 1 specifies LE.BASIC from the processors' perspective. The pseudo code assumes the following two tests that are employed when an envelope containing label *id*, and round *rnd* reaches a processor that has recorded a label *fwd_label* and a round number *fwd_round*. The functions odd($\cdot$) and even($\cdot$) return the obvious Boolean values.

Leader-test: $id = fwd\_label$ and $rnd = fwd\_round$

Casualty-test:

($fwd\_round = rnd$ and odd(rnd) and $id > fwd\_label$) or
($fwd\_round = rnd$ and even(rnd) and $id < fwd\_label$).

The protocol for each processor is parameterized by its identifier (*proc-id*). Envelope traffic terminates when one processor passes the Leader-test. This leader should then initiate maximum-finding as previously described.

We note in passing that only the parity of the round number was actually used; in LE.BASIC, a processor with round number i always receives an envelope with round number i or $i+1$. So a single bit in each message could replace the round number. In fact, round information could be entirely eliminated from messages by having processors keep track of them instead. We do not follow this observation further since round numbers (not just their parity) will be required in the improved algorithm, and because we wish to keep the role of the processors simple.

```
Processor(proc-id):
    id ⟵proc-id ; rnd ⟵1 ; fwd_label ⟵ −∞ ; fwd_round ⟵ −1 ;
    repeat
        if not Casualty-test then
            if fwd_round = rnd then
                rnd ⟵rnd+1 ;
            fwd_round ⟵rnd ;
            fwd_label ⟵id ;
            send(id, rnd) ;
        receive(id, rnd) ;
    until Leader-test .
```

Figure 1: Algorithm LE.BASIC

Correctness of LE.BASIC follows immediately after establishing:

safety: the algorithm never deletes all message envelopes,

progress: eventually only one envelope remains, and

correct termination: the algorithm elects a leader exactly when one envelope remains.

Because the ring is unidirectional and messages are processed in first-in-first-out order, the scheduler is powerless to influence the outcome of the computation. We are free, therefore, to adopt any convenient scheduler to establish correctness. We choose a scheduler that processes envelopes in such a way that all undestroyed envelopes have the same round number.

Safety and progress are consequences of the following property of LE.BASIC. In odd numbered rounds, the envelopes that are eliminated are exactly those envelopes except the first in any maximal chain of successive envelopes with descending labels; in even numbered rounds, the envelopes that are eliminated are exactly those envelopes except the first in any maximal chain of successive envelopes with ascending labels. Correct termination holds because, in order to terminate, some processor must receive the envelope it last sent.

Let ϕ denote the golden ratio $\frac{1+\sqrt{5}}{2}$. It can be shown that this algorithm requires at most $n \log_\phi n \leq 1.441 n \log n$ messages. The analysis mimics that of Peterson's algorithm [28] and is omitted here since we will shortly analyse an improved algorithm.

Notice that an envelope can only be eliminated by a processor if the round number of the

envelope and the round number of the last envelope sent by the processor agree. As well, for algorithm LE.BASIC, a processor in round i always receives an envelope in round number i or $i+1$. As a consequence, there is no danger of eliminating all envelopes, even if processors are permitted to arbitrarily promote an envelope to its next round. Our improvement to LE.BASIC employs two restricted applications of this notion of early promotion that turn out to help keep the message complexity low.

2.2 Early promotion by witness

The first technique is used to promote an envelope when it can be verified that the envelope would be promoted by LE.BASIC anyway. Denote an envelope with label a and round i by $< a, i >$. Suppose an envelope $< b, i >$, where i is even, encounters a processor, x, that last sent an envelope $< a, i-1 >$ where $a < b$. We claim that the first processor with round number i, say w, that $< b, i >$ next encounters necessarily will have a label no bigger than a. As a consequence, $< b, i >$ would be promoted to $< b, i+1 >$ at w, so instead it receives *early promotion by witness* at x, and x is a *witness for round* i.

To see the claim, consider the fate of the last envelope, $< a, i-1 >$, forwarded by x. When $< a, i-1 >$ arrives at a processor z with round number $i-1$, either it is promoted to round i because a is less than the label of z, or it is destroyed because a is greater than the label of z. In the first case, z will have recorded a as its last label and i as its round, so z is the claimed processor w. In the second case, some chain of envelopes with decreasing labels is destroyed and the last in the chain is promoted to round i. The processor that promoted that last envelope is the claimed w and necessarily has recorded a label even smaller than a.

A symmetric argument can be made for an envelope with an odd round number i that encounters a processor with a round number $i-1$ and a larger label.

Suppose in algorithm LE.BASIC, an envelope is promoted from round i to round $i+1$ by a processor y with label a and round i. Then, if early promotion by witness is incorporated into round i, there will be a witness for round i that promotes the envelope before it reaches y. In particular, the processor that promoted a to round $i-1$ is a witness for round i that the envelope meets before y.

That early promotion by witness actually achieves real message savings rather than just pre-

maturely changing round numbers is easily seen by tracing a few rounds of the communication incurred by LE.BASIC with and without early promotion by witness. This will shortly be confirmed in the analysis of our final algorithm.

2.3 Early promotion by distance

Let F_t denote the t^{th} Fibonacci number defined by $F_0 = 0, F_1 = 1$, and for $t \geq 2$, $F_{t+1} = F_{t-1} + F_t$. It can be shown that, for algorithm LE.BASIC, an envelope in round i must travel a distance of at least F_{i+1} before it is promoted to round $i+1$. If the distance travelled is substantially longer than this, then the savings due to early promotion by witness can be jeopardized by being confined to a small portion of the ring.

On the other hand, if a long gap exists, the algorithm has succeeded in eliminating more envelopes by the end of the i^{th} round than would have been eliminated by the end of the i^{th} round from an initial configuration that is designed to keep envelopes alive for as long as possible. So the algorithm is ahead of this worst case scenario and can afford to promote some round i envelopes without confirming that the test for survival is passed. The technique of *early promotion by distance* promotes an envelope in round i if it has travelled a distance of F_{i+2} without encountering any processor with a matching round number. To implement this technique a counter is added to each envelope. When an envelope is promoted to round i, its counter is set to F_{i+2}. The counter is decremented each time the envelope is forwarded without promotion, and if the counter reaches zero, then the envelope is promoted, before being forwarded.

2.4 The algorithm

Our final leader election algorithm, called ELECT, consists of LE.BASIC augmented with early promotion by distance in odd numbered rounds, and early promotion by witness in even numbered rounds.

Figure 2 specifies ELECT from the processors' perspective. Envelopes now contain three fields consisting of a label, a round number, and a counter. The pseudo code assumes the following test in addition to those used by LE.BASIC.

```
Processor(proc-id):
    id ⟵ proc-id ; rnd ⟵ 0 ; cnt ⟵ 0 ;
    fwd_label ⟵ −∞ ; fwd_round ⟵ −1 ;
    repeat
        if not Casualty-test then
            if Promotion-test then
                rnd ⟵ rnd+1 ;
                cnt ⟵ F_{rnd+2} ;
            fwd_round ⟵ rnd ;
            fwd_label ⟵ id ;
            send(id, rnd, cnt-1) ;
        receive(id, rnd, cnt) ;
    until Leader-test .
```

Figure 2: Algorithm ELECT

Promotion-test:

(even(*rnd*) and *fwd_round* = *rnd*−1 and *id* > *fwd_label*) or

(odd(*rnd*) and *cnt*= 0) or

(odd(*rnd*) and *fwd_round* = *rnd* and *id* < *fwd_label*).

2.5 Correctness of the algorithm

We establish safety, progress, and correct termination for algorithm ELECT. Again assume that the scheduler processes envelopes so that all undestroyed envelopes have the same round number.

Suppose, contrary to safety, that some run of ELECT removes all envelopes. Then there is a maximum round number p achieved by any envelope. Let S be the set of identifiers in envelopes that achieve round p. Then if p is odd (respectively, even) the envelope containing the minimum (respectively, maximum) element of S cannot be deleted and will be promoted to the next round or a leader will be declared.

Suppose, contrary to progress, that after some point, $k \geq 2$ envelopes continue to circulate around the ring. Then eventually all envelopes will receive a count at least as large as the ring. At this point each envelope has a large enough count to allow it to travel to the processor that forwarded the successor envelope. So if the round number is odd (respectively, even) the envelope with maximum label (respectively, minimum label) must be destroyed.

The algorithm cannot prematurely elect a leader or fail to elect a leader because a processor will receive an envelope with *id* equal to its *fwd_label* if and only if there are no other envelopes, thus confirming correct termination.

2.6 Analysis of the algorithm

For an envelope with label a and round number i, let $\text{host}_i(a)$ denote the processor that promoted the envelope from round $i-1$ to round i. For an envelope with label a that is eliminated in round i, let $\text{destroyer}_i(a)$ denote the processor that eliminated this envelope. Let $\delta(x, y)$ denote the distance from processor x to processor y. Since algorithm ELECT never changes the label of an envelope for the duration of its existence, we use *envelope* a as an abbreviation for the envelope with label a, and we say than an envelope is *in round* i if its round number is i. Envelope b is the *immediate successor in round* i of envelope a if the first round i envelope encountered after envelope b in round i, travelling in the direction of the ring, is envelope a.

Our analysis is a consequence of the following lemma.

Lemma 2.1 *If envelope a reaches round $i+1$, and i is odd, then $\delta(host_i(a), host_{i+1}(a)) \geq F_{i+1}$; if envelope a is destroyed in round i, and i is odd, then $\delta(host_i(a), destroyer_i(a)) \geq F_i$.*

Before proving the lemma, we examine its consequences. For any round i of algorithm LE.BASIC, all envelopes in round i travel a total of n links because each envelope travels from its host in round i, to the host of its immediate successor in round i. Algorithm ELECT uses fewer total messages because an envelope does not always travel all the way to the next host before being promoted. We now bound the savings due to early promotion. Let a and b be the labels of two envelopes in round i where envelope b is the immediate successor in round i of envelope a. We say that envelope *a saves k links in round i* if, in round i, the distance envelope a travels is $\delta(\text{host}_i(a), \text{host}_i(b))-k$.

Corollary 2.2 *Every envelope that reaches round $i+1$ where i is even saves at least F_i links in round i.*

Proof: Let a be the label of an envelope that remains alive after an even round i, and let b be the label of the envelope in round i that is the immediate successor in round i of envelope a. Then $a > b$ because a survives an even round. According to algorithm ELECT, if envelope a has not already

been promoted to round $i+1$ before reaching $\text{host}_{i-1}(b)$, it will achieve early promotion by witness at $\text{host}_{i-1}(b)$ thus saving at least $\delta(\text{host}_{i-1}(b), \text{host}_i(b))$. But, by Lemma 2.1, $\delta(\text{host}_{i-1}(b), \text{host}_i(b)) \geq F_i$. ■

Theorem 2.3 *Algorithm* ELECT *sends fewer than* $1.271n \log n + O(n)$ *messages on rings of size* n.

Proof: To bound the number of messages, we bound (1) the total number of rounds, and (2) the number of messages in any *block* of two consecutive rounds consisting of an even round followed by an odd round.

By Lemma 2.1, if round number i is odd, then the distance between any two hosts in round i is at least F_i. Thus, in round i, where i is odd, there can be at most n/F_i remaining envelopes. Denote by $F^{-1}(x)$ the least integer j such that $F_j \geq x$. It follows that algorithm ELECT uses at most $F^{-1}(n) + \mathrm{O}(1)$ rounds for rings of size n.

To estimate the number of messages send in a block, consider even round i followed by an odd round $i+1$. Assume that there are x envelopes in round $i+1$. Then, by Corollary 2.2, the total number of links travelled by envelopes in round i is at most $n - xF_i$. Clearly, the total number of links travelled by envelopes in round $i+1$ is at most n. Since, in odd round $i+1$, each envelope travels at most F_{i+1+2} before promotion, this number is also at most xF_{i+3}. Thus the total number of messages in round i is at most $\min(xF_{i+3}, n)$ and the total number of messages in the block is bounded above by $\min((n + x(F_{i+3} - F_i)), 2n - xF_i)$. Since $n + x(F_{i+3} - F_i) = 2n - xF_i$ for $x = n/F_{i+3}$, this bound is at most $2n - n\frac{F_i}{F_{i+3}}$.

Recall that ϕ denotes $\frac{1+\sqrt{5}}{2}$, and let $\hat{\phi}$ denotes $\frac{1-\sqrt{5}}{2}$. Observe that for even i:

$$\begin{aligned}
\frac{F_i}{F_{i+3}} &= \frac{\frac{1}{\sqrt{5}}(\phi^i - \hat{\phi}^i)}{\frac{1}{\sqrt{5}}(\phi^{i+3} - \hat{\phi}^{i+3})} = \frac{\phi^i - \phi^{-i}}{\phi^{i+3} + \phi^{-(i+3)}} = \frac{(\phi^i - \phi^{-i})(\phi^{i+3} - \phi^{-(i+3)})}{(\phi^{i+3} + \phi^{-(i+3)})(\phi^{i+3} - \phi^{-(i+3)})} \\
&> \frac{\phi^{2i+3} - \phi^3 - \phi^{-3}}{\phi^{2i+6}} \\
&> \phi^{-3} - \phi^{-2i}
\end{aligned}$$

Therefore, the number of messages in a block starting with even round i is at most:

$$2n - n\frac{F_i}{F_{i+3}} < n(2 - \phi^{-3} + \phi^{-2i}) = n\left(2 - \frac{8}{(1+\sqrt{5})^3} + \phi^{-2i}\right) = n(4 - \sqrt{5} + \phi^{-2i})$$

Hence there are at most

$$\sum_{\text{even}(i) \text{ and } 2 \le i \le F^{-1}(n)} n(4 - \sqrt{5} + \phi^{-2i}) \quad + \mathrm{O}(n) \;=\; \frac{4-\sqrt{5}}{2} n \log_\phi n + \mathrm{O}(n)$$

$$< \frac{1.764}{2}(1.441) n \log n + \mathrm{O}(n)$$

$$< 1.271 n \log n + \mathrm{O}(n)$$

messages sent by any computation of ELECT on a ring of size n. ■

It remains to prove Lemma 2.1.

Proof: (of Lemma 2.1) The proof is by induction on the odd round numbers. The basis, round 1, holds trivialy because each envelope travels $1 = F_1 = F_2$ link. So suppose that the lemma holds for round k where $k \geq 1$ and k is odd. Let a and b be labels of two envelopes in round $k+2$ where envelope b is the immediate successor in round $k+2$ of envelope a. According to the algorithm, envelope a travels F_{k+4} links in round $k+2$ unless it reaches $\mathrm{host}_{k+2}(b)$ before travelling this distance. Therefore, we need to estimate $\delta(\mathrm{host}_{k+2}(a), \mathrm{host}_{k+2}(b))$.

First observe two facts:

Fact 2.4 *A witness for round $k+1$ that promotes an envelope to round $k+2$ was a host for round k.*

Proof: An envelope with label c and round $k+1$ is promoted to round $k+2$ at its first encounter with a processor that has round number k and label, say d, less than c. The processor that promoted envelope d to round k, $\mathrm{host}_k(d)$, has label d and round number k and all other processors with label d and round number k must follow $\mathrm{host}_k(d)$. Envelope c could not have reached $\mathrm{host}_k(d)$ in round k since otherwise it would have been destroyed by $\mathrm{host}_k(d)$, so it encounters $\mathrm{host}_k(d)$ in round $k+1$. ■

Fact 2.5 *$\delta(host_k(b), host_{k+2}(b)) \geq F_{k+2}$.*

Proof: Consider b's travel in odd round k. If b was promoted to round $k+1$ after travelling F_{k+2} links then the observation is immediate. Otherwise, b was promoted by some processor, say $\mathrm{host}_k(c)$ and $b < c$, and, by the induction hypothesis, $\delta(\mathrm{host}_k(b), \mathrm{host}_k(c)) \geq F_{k+1}$. Since b reaches round $k+2$, in round $k+1$, b travels from $\mathrm{host}_{k+1}(b)$ $(= \mathrm{host}_k(c))$ to some witness w for round

$k+1$ with label $d < b$ that promotes b to round $k+2$. By the inequalities, $c \neq d$, and hence w must be some processor not in the interval $[\text{host}_k(c), \text{host}_{k+1}(c))$. Hence $\delta(\text{host}_{k+1}(b), \text{host}_{k+2}(b)) \geq \delta(\text{host}_k(c), \text{host}_{k+1}(c)) \geq F_k$ by the induction hypothesis. The combined distance is therefore at least $F_{k+1} + F_k = F_{k+2}$. ∎

Suppose a is eliminated in round $k+2$. Since $k+2$ is odd, $a > b$. Since a reached round $k+2$, a must have been promoted by a witness for round $k+1$. By Fact 2.4, the witness for round $k+1$ that could promote a and most closely precedes $\text{host}_{k+2}(b)$ is $\text{host}_k(b)$. Thus, $\delta(\text{host}_{k+2}(a), \text{host}_{k+2}(b)) \geq \delta(\text{host}_k(b), \text{host}_{k+2}(b)) \geq F_{k+2}$ by Fact 2.5. Therefore, the lemma holds for round $k+2$ in this case.

Suppose a survives round $k+2$. If a survives because it travels a distance of F_{k+4} without encountering a processor with round number $k+2$ then the lemma holds trivially for round $k+2$. Otherwise, a travels to $\text{host}_{k+2}(b)$ and is promoted because $a < b$. Now, a was promoted from round $k+1$ to round $k+2$ by some witness for round $k+1$ that had label smaller than a. Since $b > a$, that witness cannot be $\text{host}_k(b)$. By Fact 2.4, the witness for round $k+1$ that promotes a must be $\text{host}_k(g)$ for some envelope with label g that is between envelope a and envelope b in round $k+1$. By the induction hypothesis, $\delta(\text{host}_k(g), \text{host}_{k+1}(g)) \geq F_{i+1}$. Then,

$$\begin{aligned}
\delta(\text{host}_{k+2}(a), \text{host}_{k+2}(b)) &= \delta(\text{host}_k(g), \text{host}_{k+2}(b)) \\
&= \delta(\text{host}_k(g), \text{host}_{k+1}(g)) + \delta(\text{host}_{k+1}(g), \text{host}_{k+2}(b)) \\
&\geq \delta(\text{host}_k(g), \text{host}_{k+1}(g)) + \delta(\text{host}_k(b), \text{host}_{k+2}(b)) \\
&\geq F_{k+1} + F_{k+2} = F_{k+3}
\end{aligned}$$

So the lemma holds for round $k+2$ in this case as well. ∎

3 Comparison to previous algorithms

Two previous papers established algorithms for maximum finding that used $O(n \log n)$ messages. In this section we compare the ideas and techniques used in these papers to the ones used for algorithm ELECT. Although the the original descriptions assumed the processor perspective, we adopt the envelope viewpoint when this interpretation simplifies the description.

3.1 Basic algorithm

There in an obvious algorithm for maximum finding on a bidirectional ring derived from the perspective of processors sending and receiving messages in phases. Initially each processor is *active.* In each phase, each active processor sends its identifier to each of its active neighbours and receives the identifier of each of its two active neighbours. An active processor stays active for the next phase only if its identifier is larger than that of both of its active neighbours. Otherwise, an active processor either receives at least one identifier larger than its own and becomes *passive* for the remainder of the algorithm, or it receives its own identifier which necessarily is the maximum. Passive processors simply act as relays. Since, in each phase, only the processors with locally maximum identifiers survive, and each phase requires $2n$ messages, this basic algorithm sends at most $2n \log n$ messages on rings of size n.

Both Peterson [28] and Dolev *et al.* [15] saw how to simulate this algorithm on a unidirectional ring, where processors are constrained to send to the right and receive from the left. In phase 1, each processor is a *host* and its *current-id* is its identifier. In phase i, each host obtains the current-ids of the two hosts that immediately precede it. (This happens in two separate *rounds.* First, each host sends its current-id and receives its predecessors current-id. Second, each host forwards its predecessor's current-id and receives its predecessor's predecessor's current-id.) Let u, v, and w be three hosts with u immediately to the left of v and v immediately to the left of w. Let $\mathrm{cid}_i(x)$ denote the current-id of processor x in phase i. Processor w is a host in phase $i+1$ and sets its value of $\mathrm{cid}_{i+1}(w)$ to $\mathrm{cid}_i(v)$ if and only if $\mathrm{cid}_i(v) > \mathrm{cid}_i(u)$ and $\mathrm{cid}_i(v) > \mathrm{cid}_i(w)$. Otherwise, w is passive for the remainder of the algorithm. If a host receives its own current-id, then it is necessarily the maximum identifier and the algorithm terminates after one more phase to announce the result. We call this algorithm MAX.BASIC.

An identifier is *alive in phase i* if it is the current-id of a host in phase i. Notice that for any initial cyclic arrangement of distinct identifiers, the cyclic sequence of alive identifiers in phase i is the same for the bidirectional algorithm and the undirectional simulation. On the unidirectional ring, the alive identifiers are just cyclically "shifted" from their corresponding position on the bidirectional ring. Thus algorithm MAX.BASIC uses at most $2n \log n + \mathrm{O}(n)$ messages on a ring of size n.

From the processor's perspective, identifiers are being shifted around the ring in every second

round. From the envelope perspective, however, the shifting is just the natural consequence of an envelope keeping its original identifier. Thus, from the envelope perspective, the rounds without shifting are the more complicated ones since envelopes have to change identifiers.

3.2 Peterson's improvement

Peterson [28] improved upon MAX.BASIC by returning to the bidirectional algorithm, making an improvement there, and resimulating. Each phase of the bidirectional algorithm can be implemented as two rounds: first, identifiers are sent to the right and received from the left; second, identifiers are sent to the left and received from the right. Again, let u, v, and w be three hosts with w immediately to the right of v and v immediately to the right of u. If v receives $u > v$ from the left, it can go passive immediately without waiting for w. Thus the next round is played in the other direction and with a reduced set of hosts. Peterson used the same technique as previously to simulate this improved bidirectional algorithm on a undirectional ring. He obtained a new unidirectional algorithm that uses fewer than $1.441n \log n + O(n)$ messages. His algorithm preserves the maximum identifier by using shifting in only every second round. If his algorithm were changed to one with shifting in every round, the result would be the leader election algorithm LE.BASIC, which is a still simpler algorithm once the envelope perspective is adopted.

3.3 Dolev *et al.* improvements

Dolev *et al.* [15] improve upon MAX.BASIC by adding two techniques to reduce the cost of each phase below $2n$.

Since their algorithm always keeps the maximum identifier alive, there is no harm done if the other identifiers are eliminated by passive processors that have previously seen a larger identifier (provided there is no danger of deadlocking). Consider three successive envelopes a, b and c, in phase i of MAX.BASIC and suppose $b < a$. (We use the notation $\text{host}_i(p)$ as before except that now i is the phase number of MAX.BASIC.) Then $\text{host}_i(c)$ will necessarily be passive in phase $i+1$. This situation will be discovered by $\text{host}_i(b)$ after the first round of phase i when it receives a. Thus $\text{host}_i(b)$ need not send the second message of phase i to $\text{host}_i(c)$. Instead it can append a flag to the first message of phase $i+1$ to belatedly inform the next host ($\text{host}_i(c)$), which is still waiting for its second message of phase i, that phase i is over and it became passive at the end of phase

i. This technique, which we call *early stopping by witness*, is comparable to early promotion by witness. Both techniques are possible only because of shifting. That is, the saving can be realized because if a current-id, d, survives phase i, then $\text{host}_{i+1}(d)$ is the host in phase i of the successor of d. Furthermore, early *stopping* by witness can be achieved only because the comparisons made in successive rounds are of the same type (maximum identifiers survive); whereas early *promotion* by witness can be achieve only because the comparisons made in successive rounds alternate between types (minimums survive in odd numbered rounds; maximums in even numbered rounds).

One algorithm due to Dolev *et al.* incorporates early stopping by witness and early promotion by distance (where the distance chosen is 2^i) to the second round of each phase of MAX.BASIC. The resulting algorithm uses at most $1.5n \log n + O(n)$ messages. Finally, they added the same two improvements (with the distance bound adjusted to a Fibonacci number) to Peterson's improved algorithm, to achieve a message complexity of $1.356n \log n + O(n)$.

It is instructive to enquire why the final algorithm of Dolev *et al.* is more expensive in terms of messages than algorithm ELECT and why its description and especially its analysis are so complicated. First, if the viewpoint is changed to that of envelope traffic, the description is simplified. In particular, the various states of processors are unnecessary. Processors need only record the last message sent. However, the algorithm would still be significantly more complicated than ELECT because it needs to keep the maximum identifier alive. This in turn means that shifting (which is the natural scenario from the envelope perspective) is only possible in even numbered rounds. In odd numbered rounds, processors have to change the labels in envelopes. Because early stopping by witness requires shifting, it must be added to even numbered rounds. But early promotion by distance also occurs only in even numbered rounds. This not only decreases the saving but complicates the analysis because every second round sends n messages and all the savings are achieved in the alternate rounds.

4 Future Research

In spite of the large amount of existing research on distributed maximum finding, our understanding of even the simplest and most studied case — that of unidirectional rings with unit weight links and distinct identifiers — is incomplete. The exact message complexity of this problem is unknown. We speculate that a further improvement by a constant factor may be possible. Algorithm ELECT

interleaves two types of rounds: one uses early promotion by distance; the other uses early promotion by witness. However, both techniques could be incorporated into every round. Peterson's original algorithm sends $2n$ messages for every block of 2 rounds. The final algorithm of Dolev *et al.* reduces this cost by $\phi^{-4}n$ messages and algorithm ELECT reduces it by $\phi^{-3}n$ messages. We conjecture, however, that the additional savings achievable are limited and that the complexity of maximum finding in this model cannot be improved beyond a savings per block of $\phi^{-2}n$ messages. Lower bounds for this problem remain below $n \log n$.

This paper addressed the message complexity of maximum finding under the assumption that all edges incur the same cost. We are currently investigating the extension of some of the techniques in this paper to maximum finding on weighted rings, that is rings where each edge has an associated cost.

References

[1] 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.

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

[3] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Probabilistic leader election on rings of known size. In *Lecture Notes in Computer Science #519*, pages 481–495. Springer Verlag, 1991. Workshop on Algorithms and Data Structures.

[4] Y. Afek and E. Gafni. Simple and efficient distributed algorithms for election in complete networks. In *Proc. 22nd Ann. Allerton Conf. on Communication, Control, and Computing*, pages 689–698, 1984.

[5] Y. Afek and E. Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 186–195, 1985.

[6] Y. Afek and Y. Matias. Elections in anonymous networks. Technical Report SC-TR-91-115, UMIACS, 1991.

[7] 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.

[8] H. Attiya, N. Santoro, and S. Zaks. From rings to complete graphs — $\theta(n \log n)$ to $\theta(n)$ distributed leader election. Technical Report SCS-TR-109, Carleton University, 1987.

[9] H. Attiya and M. Snir. Better computing on the anonymous ring. *J. Algorithms*, 12(2):204–238, 1991.

[10] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *J. Assoc. Comput. Mach.*, 35(4):845–875, 1988.

[11] H. L. Bodlaender. A better lower bound for distributed leader finding in bidirectional asynchronous rings of processors. *Information Processing Letters*, 27:287–290, 1988.

[12] 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.

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

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

[15] 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):245–260, 1982.

[16] P. Duris and Z. Galil. Two lower bounds in asynchronous distributed computation. *Journal of Computer and System Sciences*, 42:254–266, 1991.

[17] E. Gafni. Improvements in the time complexity of two message-optimal election algorthms. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 175–184, 1985.

[18] E. Gafni and Y. Afek. Election and traversal in unidirectional networks. In *Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing*, pages 190–198, 1984.

[19] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Trans. on Prog. Lang. and Systems*, 5(1):66–77, 1983.

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

[21] D. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 23(11):627–628, 1980.

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

[23] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 163–174, 1985.

[24] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of procesors. In *Proc. 3rd Annual ACM Symp. on Principles of Distributed Computing*, pages 199–207, 1984.

[25] G. LeLann. Distributed systems — towards a formal approach. In *Information Processing 77*, pages 155–160, New York, 1977. Elsevier Science.

[26] S. Moran and M. Warmuth. Gap theorems for distributed computation. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 131–140, 1986.

[27] J. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum finding. *J. Assoc. Comput. Mach.*, 31(4):905–918, 1984.

[28] G. Peterson. An $O(n \log n)$ algorithm for the circular extrema problem. *ACM Trans. on Prog. Lang. and Systems*, 4(4):758–752, 1982.