Self-Stabilizing Token Circulation on Anonymous Message Passing Rings (Extended Abstract)

Lisa Higham*
Dept. Of Comp. Sc.
University of Calgary
Calgary, Alberta, Canada

Steven Myers†
Dept. Of Comp. Sc.
University of Toronto‡
Toronto, Ontario, Canada

Abstract. *A self-stabilizing algorithm that solves the problems of token circulation and leader election on anonymous and uniform, unidirectional, message passing rings of arbitrary, but known, size is developed. From any initial configuration, the expected time to stabilization on a ring of size n is in $\mathcal{O}(n \log n)$. Furthermore, the size of the configuration of the system remains constant throughout the execution; each processor state and message state has size in $\mathcal{O}(\log n)$. The correctness of the algorithm relies upon a novel duality between messages and processes.*

Keywords: Self-stabilization, Token Circulation, Leader Election, Message-passing Ring.

1 Introduction

Many protocols for distributed computing assume the existence of primitives such as token circulation and leader election. Leader election requires that a unique processor be identified as the leader, from a set of indistinguishable processors. Anonymous networks use leader election as a first step in assigning session specific identities to processors. The token circulation problem requires that a token be circulated by all of the processors in some fair fashion. A token circulation algorithm is often used as a basis for solving the mutual exclusion problem.

*Email: higham@cpsc.ucalgary.ca, Research was supported in part by Natural Sciences and Engineering Research Council of Canada grant OGP0041900.

†Email: myers@cs.toronto.edu, Research was supported in part by Natural Sciences and Engineering Research Council of Canada grant OGP0041900, by a university of Calgary research grant, and by a STEP grant.

‡Most of this work was completed while the author was at the University of Calgary.

Such primitives should be able to withstand the transient faults to which distributed systems are susceptible. To this end, Dijkstra [4] defined a distributed system to be *self-stabilizing* if, when started from an arbitrary configuration, the system is guaranteed to reach a legitimate configuration, as execution progresses. Algorithms with small stabilization times are desirable because a system will make progress provided that the time between its faults is longer than the time necessary for it to stabilize.

We present a self-stabilizing algorithm that solves the problems of token circulation and leader election on anonymous, unidirectional, message passing rings of arbitrary, but known, size. From any initial configuration, the expected time until our algorithm stabilizes on a ring of size n is in $\mathcal{O}(n \log n)$. (Henceforth, ring size is denoted by n.) Because, as noted by Gouda and Multari [9], there can be no purely asynchronous self-stabilizing protocols for message passing models, we augment our system with a time-out mechanism. In fact, we present our algorithm in the synchronous model; however, it can be easily adapted for an asynchronous model provided with a time-out mechanism that detect deadlocks. Since it is impossible to deterministically break symmetry on an anonymous ring [3] randomization is used. Randomization also allows us to circumvent the deterministic lower bound of Dolev, Israeli, and Moran [7], who show that the configuration size of any deterministic, self-stabilizing, message driven protocol that solves a weak exclusion task (of which token passing is an example) grows logarithmically with time. In our algorithm the size of each processor state and message state is bounded by $\mathcal{O}(\log n)$.

Self-stabilizing token circulation and leader election have been well studied. Much of this research (see [10] for an extensive bibliography) has assumed Dijkstra's original *composite* atomicity ([5]) shared memory model, where an atomic step may contain several reads and one write operation, and has focused on various additional factors (such as graph topology, existence of identifiers, type of scheduling deamon, use of randomization, and properties of the ring size) and goals (minimizing state space and stabilization time). In contrast, for these problems, fewer algorithms exist for the weaker read/write atomic shared register model [5, 6, 8].

Although it has been established that self-stabilizing algorithms for the atomic read/write shared memory model can be transformed to ones for message passing models (see [15, 13, 14]), these transformations require either bidirectional links or identifiers, (or both) or incur substantial overhead. This leaves open the question of efficient, self-stabilizing token circulation and leader election algorithms for anonymous unidirectional message-passing networks, although a few papers address self-stabilizing solutions for some other problems in the message passing model [7, 2, 9, 14, 1].

2 Model

We assume a synchronous, unidirectional ring consisting of n anonymous processors. At each time step, a processor can send a message which, barring

any transient faults, will arrive at its successor in the next time step. Each processor has access to a statistically independent random bit generator that can produce one bit per processor per time step. It is convenient to think of messages as being contained in *envelopes* that circulate around the ring. The receiving processor may change the envelope contents prior to forwarding it in the next step, or it may destroy the envelope (and its contents). Also, processors may create new envelopes, initialize their contents and add them to the circulating collection of envelopes.

Since token circulation and leader election are closely related [16], it is not surprising that one algorithm can solve both problems. At any point in our algorithm, some processors may be *producers*. For leader election, the producers are the candidate leaders; for token circulation, the envelopes are the tokens. Eventually there will remain exactly one producer and exactly one circulating envelope, thus solving both problems.

In the self-stabilizing model, an initial period, during which the problem requirements are not met, is permitted, so long as there is a guarantee that after a finite amount of time they will be met, and remain met. Therefore, for the leader election (respectively, the token circulation) problem, there initially may be system configurations with no producers or more than one producer (respectively, no envelopes, or more than one envelope) but it must be guaranteed that eventually all succeeding configurations have exactly one producer (respectively, envelope).

3 Self-Stabilizing Token Circulation

3.1 Intuition for SSTC

Our algorithm builds upon a randomized *basic attrition* protocol [11] for reliable asynchronous message passing networks. When basic attrition is initiated by any non-empty collection of processors, called *producers*, we are guaranteed to eventually be left with exactly one producer, which continues to circulate a single envelope around the ring. Unfortunately, basic attrition is not self-stabilizing; we first describe basic attrition and then the enhancements required to make it so.

During each turn, each producer independently tosses an unbiased coin, sends the outcome to the next producer (via the intervening non-producers) and waits to receive the coin toss generated by the preceding producer. A producer becomes a non-producer for the remainder of basic attrition if and only if it sent a tail and received a head. Otherwise it proceeds to its next turn. If, in a fixed turn, all producers have the same flip, then each remains a producer for the next turn; if not all flips are the same, then those that flipped heads are guaranteed to be producers for another turn. Therefore, not all producers can become non-producers. The probability that a given producer sends a tail and receives a head is $1/4$ so long as there is more than one producer. Hence, with probability 1, the number of producers and envelopes decreases to exactly one.

Notice that every time a producer is eliminated, a circulating envelope is also destroyed. Hence, basic attrition maintains a one-to-one correspondence between envelopes and producers, which is crucial for its correctness. If initially this correspondence does not hold, basic attrition will remove all envelopes or all producers, whichever is initially fewer. Our enhancement of basic attrition to make it self-stabilizing is to detect and correct configurations where there are either no envelopes or no producers.

Unfortunately, the absence of envelopes cannot be detected on a purely asynchronous message-driven system; that is why we focus our attention on synchronous systems. Counters are added to both the processors and the messages in the envelopes. A message counter is set to n whenever a message is sent by a producer, and is decremented by 1 each time the message is forwarded by a non-producer. Hence, if a message's counter ever reaches 1, the envelope has not passed a producer for its entire journey around the ring. It therefore causes the receiving processor to become a producer, ensuring that there is at least one producer. Similarly, a producer sets its counter to n when it sends an envelope, and decrements its counter each time step when it does not receive an envelope. Hence, if a producer's counter ever reaches 1, it has not received an envelope during an interval long enough for the last one it sent to circulate back to it. It therefore creates a new envelope, ensuring that there is at least one. Finally, a non-producer behaves similarly to a producer except that it sets its counter to $2n$ rather than n when it sends an envelope, which impedes any non-producer from creating a new envelope and becoming a producer when a producer already exists.

3.2 Specification of SSTC

Each processor on a ring of size n maintains a counter, and each producer stores the result of a coin flip. Thus, a *processor-state* is a triple (Prod?, proc_flip, proc_count). For processor α, the value of α.Prod? $\in$ {TRUE, FALSE} is TRUE if and only if α is a producer. If α is a producer, the value of α.proc_flip $\in$ {HEADS, TAILS, $*$} is the current coin flip of α; if α is a non-producer, it is $*$. The value of α.proc_count $\in \mathbf{Z}$ is the value of the counter of α, which always satisfies $1 \leq \alpha.\text{proc\_count} \leq 2n$; additionally, it satisfies $1 \leq \alpha.\text{proc\_count} \leq n$ if α is a producer.

Each envelope carries a message consisting of a coin flip and a counter. Thus, a *message-state* is a pair (mess_flip, mess_count). For message m, the value of m.mess_flip $\in$ {HEADS, TAILS} is the coin flip of m, and the value of m.mess_count $\in \mathbf{Z}$ is the counter value of m, which satisfies $1 \leq m.\text{ mess\_count} \leq n$.

At each time step the action of a processor α is determined by the value of the pair (α.Prod?,α.Mess?), where Mess? is TRUE if and only if α received an envelope in the given time step. The function *coin-flip* returns a value chosen randomly and independently from the uniform distribution over {HEADS, TAILS}. The procedure *set* updates the processor-state. The procedure *send* creates an envelope, inserts a message with the given state and sends it. Each

processor in the ring executes the algorithm SSTC .

Algorithm 1 SSTC

```
1   repeat for every time step:
2   let (Prod?, proc_flip, proc_count) be the current processor state
3   if a message is received
4       let (mess_flip, mess_count) be the state of the message
5       Mess? ← TRUE
6   else Mess? ← FALSE
7   case (Prod?, Mess?) of
8       (TRUE, TRUE)
9           {if not (mess_flip=HEADS and proc_flip=TAILS)      ▷ Prod & mess survive
10              flip⟵ coin-flip
11              set (TRUE, flip, n) ; send (flip, n)
12          else                                              ▷ Producer & message killed
13              set (FALSE, *, 2n)}
14      (FALSE, TRUE)
15          {if (mess_count=1)                                ▷ Message times-out
16              flip⟵ coin-flip
17              set (TRUE, flip, n) ; send (flip, n)
18          else                                              ▷ Pass on Message
19              set (FALSE, *, 2n) ; send (mess_flip, mess_count−1) }
20      (TRUE, FALSE)
21          {if (proc_count=1)                                ▷ Producer times-out
22              flip⟵ coin-flip
23              set (TRUE, flip, n) ; send (flip, n)
24          else
25              set (TRUE, proc_flip, proc_count−1) }
26      (FALSE, FALSE)
27           {if (proc_count=1)                               ▷ Non-producer times-out
28              flip⟵ coin-flip
29              set (TRUE, flip, n) ; send (flip, n)
30          else
31              set (FALSE, *, proc_count−1)}
```

4 Correctness of SSTC

To prove the correctness of SSTC , we establish that the difference between the number of producers and the number of envelopes never increases, and with probability 1 decreases as time progresses, so long as the difference is greater than zero. We next show that if the difference between the number of producers and envelopes is zero, then with probability 1 the number of producers (and thus number of envelopes) reaches one and remains at one for the remainder of the execution. Due to space constraints, some proofs of lemmas are omitted. All missing proofs are available in the full version of this paper [12].

For the purpose of the proof only, imagine that the processors are numbered from 1 to n. A *configuration* of the ring is given by a sequence of n

pairs, where the i^{th} pair is a description of the state of processor i and the state of the message at processor i, if there is one. (If there is no envelope at a processor then the message-state is null, and since we are dealing with synchronous systems we can assume that there is at most one message at each processor.) Execution of SSTC starts from an arbitrary initial configuration denoted by χ. Given χ and an infinite sequence of coin flips, E, the configuration immediately following χ is determined by applying the repeat loop SSTC once to each processor. If processor i requires a coin flip, it takes the value of the i^{th} bit of E. After the application, E is updated by removing its first n bits. Given a initial configuration χ and an infinite sequence of coin flips, E, the configuration that results *after t time steps* is determined by repeating the above t times, and is denoted $\text{Config}(\chi, E, t)$.

The first three lemmas establish that within the first $3n$ time steps, regardless of E, a configuration is achieved such that each envelope has been sent by a producer, and each processor has sent an envelope and non-producers will henceforth only act as relays. This allows us to argue that there is a correlation between some of the message and processor states after $3n$ time steps.

Lemma 4.1 *For any initial configuration χ and any coin-flip sequence E and for all $r \geq n$, all envelopes in Config(χ, E, r) have been sent by a producer.*

Lemma 4.2 *For any initial configuration χ, and any coin-flip sequence E, every processor has sent an envelope by time step $3n$.*

Say that a processor (respectively, envelope) *times-out* when its counter reaches 1 and it does not receive an envelope (respectively, arrive at a producer).

Lemma 4.3 *A non-producer cannot time-out after time $3n$.*

Given an execution from an initial configuration χ, and assuming some coin-flip sequence E, let $\text{Prod}(\chi, r)$ denote the number of producers in $\text{Config}(\chi, E, r)$, and let $\text{Env}(\chi, r)$ denote the number of envelopes in $\text{Config}(\chi, E, r)$.

Consider any configuration, χ, arising after time step $3n$, and having *at least as many producers as envelopes.* Lemma 4.4 shows that the next time-out cannot be an envelope time-out. Only an envelope time-out, however, can increase the number of producers while leaving the number of envelopes unchanged. Thus the number of producers minus the number of envelopes cannot increase. Furthermore, as established by Lemma 4.5, a configuration with more envelopes than producers can never succeed χ.

Lemma 4.4 *For every configuration χ and for every $r \geq 3n$, if Prod$(\chi, r) \geq$ Env(χ, r) then no envelopes time-out in the first round after r in which any time-outs occur.*

Lemma 4.5 *For every configuration χ and for every $r \geq 3n$, if Prod$(\chi, r) >$ Env(χ, r) and the first time-out after r is at time T, then Prod$(\chi, T) \geq$ Env(χ, T).*

After the initial $3n$ steps when non-producers cease timing out, and act only as relays, there is no substantial difference between the state and behaviour of producers and that of messages. That is, the configuration that results from interchanging the role of producers and messages and sending messages backwards, is the same as that obtained from the original configuration and sending messages forward. The next two lemmas are dual to the previous two for the case when, at some time r, the number of envelopes is at least as big as the number of producers, and their proofs can be derived as a consequence of this duality between envelopes and producers. The theorem that establishes this duality is proved in the appendix.

Lemma 4.6 *For every configuration χ and for every $r \geq 3n$, if $Env(\chi, r) \geq Prod(\chi, r)$ then no producers time-out in the first round after r in which any time-outs occur.*

Lemma 4.7 *For every configuration χ and for every $r \geq 3n$, if $Env(\chi, r) > Prod(\chi, r)$ and the first time-out after r is at time T, then $Env(\chi, T) \geq Prod(\chi, T)$.*

The four lemmas 4.4, 4.5, 4.6, and 4.7 form the core of the proof of correctness of algorithm SSTC. They allow us to argue that as the computation progresses, steps arise that either decrease the difference between the number of producers and envelopes or decrease the total number of producers and envelopes until eventually exactly one producer with a matching envelope remains.

A *competition* takes place when a producer receives an envelope. Call any time-out or any competition a *significant event*, and call a time step t *significant* if a significant event occurs at time t. Observe that the number of envelopes or producers can change only at significant time steps. The proof of the next lemma is obvious.

Lemma 4.8 *For any initial configuration, significant events are guaranteed to continually arise at intervals of at most $2n$.*

Let $R_0 = 3n$, and let R_i be the i^{th} significant time step after R_0.

Lemma 4.9 *If for some configuration χ and for some $t \geq 0$, $Prod(\chi, R_t) = Env(\chi, R_t)$ then for every $j \geq t$, $Prod(\chi, R_j) = Env(\chi, R_j)$ and, with probability 1, there is a k such that for every $l \geq k$ $Prod(\chi, R_l) = Env(\chi, R_l) = 1$.*

Proof: Consider step R_{t+1}. By Lemma 4.3 there cannot be a non-producer time-out; by Lemma 4.6 there cannot be a producer time-out; by Lemma 4.4 there cannot be an envelope time-out. Therefore, the next significant event must be a competition, and so there must be at least one producer and one envelope. However, every competition that is won leaves the number of producers and envelopes unchanged, and every competition that is lost removes exactly one producer and one envelope. Hence, $\mathrm{Prod}(\chi, R_{t+1}) = \mathrm{Env}(\chi, R_{t+1})$. It follows by induction that for every $j \geq t$, $\mathrm{Prod}(\chi, R_j) = \mathrm{Env}(\chi, R_j)$. Furthermore, if $\mathrm{Prod}(\chi, R_j) = \mathrm{Env}(\chi, R_j) = s \geq 2$ then, with probability at least

1/4, $\text{Prod}(\chi, R_{j+1}) < s$. So with probability 1, eventually, say at time step R_k, there will be one producer and one envelope. Every competition that follows will be won, ensuring that for every $l \geq k$, $\text{Prod}(\chi, R_l) = \text{Env}(\chi, R_l) = 1$. ■

Theorem 4.10 *For any initial configuration χ, algorithm* SSTC *eventually converges to (and henceforth remains in) a configuration with one envelope and one producer.*

Proof: There are three cases depending upon the relationship between $\text{Prod}(\chi, R_0)$ and $\text{Env}(\chi, R_0)$. If $\text{Prod}(\chi, R_0) = \text{Env}(\chi, R_0)$ then the theorem follows from Lemma 4.9.

Suppose that $\text{Prod}(\chi, R_0) > \text{Env}(\chi, R_0)$. We show that, with probability 1, there is a $k > 0$ such that $\text{Prod}(\chi, R_k) = \text{Env}(\chi, R_k)$. The theorem will then follow from Lemma 4.9.

It follows from Lemmas 4.3, 4.4 and 4.5 that all time-outs after step R_0 must be producer time-outs. Hence, all significant events in the computation after time R_0 are either competitions or producer time-outs. Let $\delta_t = \text{Prod}(\chi, R_t) - \text{Env}(\chi, R_t)$ and consider how δ_t changes over time. Each competition leaves δ_t unchanged. Each producer time-out increases the number of envelopes by one and leaves the number of producers unchanged, so each time-out reduces δ_t by one. Furthermore, it follows from induction, Lemma 4.9 and Lemma 4.5, that for all t, $\text{Prod}(\chi, R_t) \geq \text{Env}(\chi, R_t)$, so $\delta_t \geq 0$. Therefore, there can be at most $\text{Prod}(\chi, R_0) - \text{Env}(\chi, R_0)$ time-outs.

Each competition that is lost removes exactly one envelope and one producer. So after any time R_t there can be at most $\text{Env}(\chi, R_t)$ lost competitions before another producer time-out occurs. Since competitions are lost with probability 1/4 as long as there is more than one producer, there are a bounded number of competitions expected before the next producer time-out as long as $\text{Prod}(\chi, R_t) \geq 2$. Thus with probability 1 there will eventually be a k such that $\delta_k = 0$.

The proof for the final case when $\text{Env}(\chi, R_0) > \text{Prod}(\chi, R_0)$ follows from that of the previous case and the duality theorem. ■

5 Stabilization Complexity

Theorem 5.1 *For any ring of size n, in any initial configuration, the expected time until* SSTC *stabilizes to a configuration with exactly one producer and one envelope is $\mathcal{O}(n \log n)$.*

Proof: Consider the number of producers and envelopes in the configuration ψ that is achieved after the first $3n$ time steps.
CASE 1. Configuration ψ has at least as many envelopes as producers.

Let $\mathcal{M} = \{m_1, m_2, ..., m_k\}$ be the set of envelopes in ψ. Because ψ occurs after $3n$ time steps, it must contain at least one producer, and by assumption, at most k. We assume that $|\mathcal{M}| \geq 2$, since otherwise the theorem holds trivially.

Let SSTC run for an additional $2n$ steps. Then partition $\mathcal{M}$ into two sets $\mathcal{A}$ and $\mathcal{B}$ where $\mathcal{A}$ is the set of envelopes that did not have a competition during the $2n$ steps, and $\mathcal{B}$ is the set that did.

Claim 5.2 *For every envelope in $\mathcal{A}$, there is a unique envelope in $\mathcal{B}$ that was eliminated during the $2n$ time steps.*

Proof: Any envelope $m \in \mathcal{A}$ cannot be received by a producer in the first n steps, therefore it has timed-out at some processor α by step n, forcing α to become a producer. However m cannot be received by any producer by step $2n$, which implies that α is no longer a producer. The only way α could have become a non-producer is through a lost competition with an envelope m_α, in which m_α also would have been eliminated. So it suffices to show that $m_\alpha \in \mathcal{B}$. Because the $2n$ time steps were applied to configuration ψ, which arose after SSTC had executed for $3n$ steps, we know by Lemmas 4.3, 4.6, and 4.7 that non-producers and producers did not time out during the $2n$ time steps. Since the only way to create a new envelope is by a processor time-out, there are no new envelopes. Therefore m_α existed in ψ and so $m_\alpha \in \mathcal{B}$. ■

Corollary 5.3 $|\mathcal{A}\,| \leq 1/2|\mathcal{M}|$.

To determine a lower bound on the expected number of envelopes eliminated after the $2n$ steps, set x_i to 1 if m_i is in a lost competition in the $2n$ steps. Otherwise set x_i to 0. Thus $x = \sum_i x_i$ is the number of messages lost in competitions during the $2n$ steps. Let $\mathcal{R}_\mathcal{A}$ be a random set, which contains the messages that will be in $\mathcal{A}$ at the end of the $2n$ steps. Conditioned on belonging to $\mathcal{R}_\mathcal{A}$ we assign probabilities to x_i. Since every envelope in $\mathcal{R}_\mathcal{A}$ will not have any competitions during the $2n$ steps, the probability of any of these envelopes being in a lost competition is 0. Since $|\mathcal{M}| \geq 2$, the coin tosses of all producers and envelopes holding competitions are independent. Thus, for each $m_i \notin \mathcal{R}_\mathcal{A}$, the probability of being in a lost competition is at least $\frac{1}{4}$. That is:

$$\Pr[x_i = 1 | m_i \in \mathcal{R}_\mathcal{A}\,] = 0 \text{ and } \Pr[x_i = 1 | m_i \notin \mathcal{R}_\mathcal{A}\,] \geq \frac{1}{4}$$

We now determine a lower bound on the expected value of x conditional on $\mathcal{R}_\mathcal{A}$, and use it to derive a lower bound on the expectation of x.

$$\begin{aligned}
\mathrm{E}\,[x|\mathcal{R}_\mathcal{A}\,] &= \mathrm{E}\,[\textstyle\sum_i x_i | \mathcal{R}_\mathcal{A}\,] \\
&= \mathrm{E}\left[\textstyle\sum_{m_i \in \mathcal{R}_\mathcal{A}} x_i | m_i \in \mathcal{R}_\mathcal{A}\right] + \mathrm{E}\left[\textstyle\sum_{m_i \notin \mathcal{R}_\mathcal{A}} x_i | m_i \notin \mathcal{R}_\mathcal{A}\right] \\
&= \sum_{m_i \in \mathcal{R}_\mathcal{A}} \mathrm{E}\,[x_i | m_i \in \mathcal{R}_\mathcal{A}\,] + \sum_{m_i \notin \mathcal{R}_\mathcal{A}} \mathrm{E}\,[x_i | m_i \notin \mathcal{R}_\mathcal{A}\,] \\
&= \sum_{m_i \in \mathcal{R}_\mathcal{A}} 1 \cdot \Pr[x_i = 1 | m_i \in \mathcal{R}_\mathcal{A}\,] + \sum_{m_i \notin \mathcal{R}_\mathcal{A}} 1 \cdot \Pr[x_i = 1 | m_i \notin \mathcal{R}_\mathcal{A}\,] \\
&\geq 0 + \frac{1}{4}(|\mathcal{M} \setminus \mathcal{R}_\mathcal{A}\,|) = \frac{1}{4}(|\mathcal{M}| - |\mathcal{R}_\mathcal{A}\,|)
\end{aligned}$$

By Corollary 5.3 $|\mathcal{R}_{\mathcal{A}}| \leq \frac{1}{2}|\mathcal{M}|$. Therefore, the expected value of x conditioned on $\mathcal{R}_{\mathcal{A}}$ is bounded by $\mathrm{E}[x|\mathcal{R}_{\mathcal{A}}] \geq \frac{1}{4}(|\mathcal{M}| - |\mathcal{R}_{\mathcal{A}}|) \geq \frac{1}{8}|\mathcal{M}|$ which is independent of $\mathcal{R}_{\mathcal{A}}$ implying that $\mathrm{E}[x] \geq \frac{1}{8}|\mathcal{M}|$.

Call each $2n$ steps a *phase*. It follows from Lemmas 4.6 and 4.7 that the configuration resulting after the phase must also have at least as many envelopes as producers and no new envelopes. Therefore the argument can be iterated, reducing the number of remaining messages with each $2n$ time steps. Let Ψ^i be the configuration at the end of phase i. Let Y^i be the number of envelopes in Ψ^i, and let X^i be the number of envelopes that have lost a competition by the end of phase i. If $M = |\mathcal{M}|$ is the number of envelopes initially in ψ, then $X^i + Y^i = M$, and so, provided $M \geq 2, \mathrm{E}\left[Y^{i+1}|Y^i = M\right] \leq \frac{7}{8}M$. This inequality leads to $\mathrm{E}\left[Y^{i+1}\right] < \frac{7}{8}\mathrm{E}\left[Y^i\right]$ (see the full version of this paper for details).

Since there are at most n envelopes in ψ, after at most $\log_{\frac{8}{7}} n$ phases we expect at most 2 envelopes to remain on the ring. Thus, after $\log_{\frac{8}{7}} n$ phases the probability that there are more than 4 envelopes remaining is less than $1/2$. Therefore, the expected number of phases until there are at most 4 envelopes is $c \log n$ for a small constant c. It is easy to see that in an additional expected $\mathcal{O}(1)$ phases the number of messages will reduce to 1. So in expected time $(2nc \log n + \mathcal{O}(n)) \in \mathcal{O}(n \log n)$ the number of messages will be reduced to 1, and the ring will be stabilized.

CASE 2. There are at least as many producers as messages at time $3n$.

This follows as a result of the duality theorem and the proof of CASE 1. ■

6 Acknowledgements

The authors thank Zhiying Liang for her early involvement in the project, and Eric Ruppert, and Wayne Eberly for their comments and suggestions, which have improved the final version of this paper.

References

[1] Y. Afex, A. Bremler, "Self-Stabilizing Unidirectional Network Algorithms by Power-Supply", Proceedings of the Symposium on Discrete Algorithms, 1997

[2] Y. Afek, G.M. Brown, "Self-Stabilization over unreliable communication media", *Distributed Computing*, Vol. 7, 1993, pp. 27-34

[3] D. Angluin, "Local and Global Properties in Networks of Processors", *Proceedings of the 12 ACM Symposium on Theory of Computing* , 1980, pages 82-93

[4] E.W. Dijkstra, "Self-stabilizing Systems in Spite of Distributed Control", *Communications of the ACM*, November 1974, pp. 643-644

[5] S. Dolev, A. Israeli, S. Moran, "Self-Stabilization of dynamic systems assuming only read/write atomicity", *Distributed Computing*, Vol.1, 1993, pp. 3-16

[6] S. Dolev, A. Israeli, S. Moran, "Uniform Self-Stabilizing Leader Election Part 1: Complete Graph Protocols", *World Wide Web*, [http://www.cs.bgu.ac.il/~dolev], 1995

[7] S. Dolev, A. Israeli, S. Moran, "Resource Bounds for Self-Stabilizing Message Driven Protocols", *SIAM Journal of Computing*, Vol.26, No. 1, 1997, pp. 273-290

[8] S. Dolev, A. Israeli, S. Moran, "Uniform Dynamic Self-Stabilizing Leader Election", *IEEE Transactions on Parallel and Distributed Computing*, Vol.8, No. 4, 1997, pp. 424-440

[9] M. G. Gouda and N.J. Multari, "Stabilizing communication protocols",*IEEE Transactions on Computing*, Vol.40,1991, pp.448-458

[10] T. Herman, "Comprehensive Self-Stabilization Bibliography", http://www.cs.uiowa.edu/ftp/selfstab/bibliography/, August 1998

[11] L. Higham, D.Kirkpatrick, K.Abrahamson, and A.Adler. "The Bit Complexity of Randomized Leader Election on a Ring", *SIAM Journal of Computing* , Vol. 18,No. 1, 1989

[12] L.Higham and S. Myers, "Self-Stabilizing Token Circulation on Anonymous Message Passing Rings", Technical Report TR 98/634/25, Dept. Of Computer Science, University of Calgary, 1998

[13] S.T. Huang, L.C. Wuu, M.S. Tsai, "Distributed Execution Model for Self-Stabilizing Systems", *Proc. Int'l Conf. Distributed Computing Systems*, 1994, pp. 432-439

[14] S. Katz, K.J. Perry, "Self-Stabilizing extensions for message-passing systems", *Distributed Computing*, Vol. 7, 1993, pp. 17-26

[15] M. Mizuno, H. Kakugawa, "A Transformation of Self-Stabilizing Programs for Distributed Computing Environments", *Proc. 10 Int'l Workshop Distributed Algorthims*, 1996.

[16] A. Mayer, Y. Ofek, R. Ostrovsky, M. Yung , "Self-Stabilizing Symmetry breaking in Constant Space", *Distributed Computing*, Vol. 7, 1993, pp. 17-26

7 Appendix: The Duality Theorem

In algorithm SSTC , there is a duality between the behaviour of messages and producers after a small initial time interval. We first described and prove this duality, and then exploit it to get simple proofs of Lemmas 4.6 and 4.7.

It is helpful to represent non-messages explicitly in order to highlight the duality. Thus, we redefine a *message-state* as a triple (Mess?, mess_flip, mess_count). For message m the value of m.Mess? $\in$ { TRUE, FALSE } is TRUE if and only if m is a message. The value or m.mess_flip $\in$ { HEAD, TAIL,$*$} is the current coin flip of m, if m is a message, and $*$ if it is not a message, and the value or m.mess_count $\in Z \cup \{*\}$ is the current counter value of m, $1 \leq m.\text{mess\_count} < n$, if m is a message and is $*$ otherwise.

Also recall that after time $3n$ non-producers cannot time out (Lemma 4.3). This means that after $3n$ steps, the counters of non-producers can be ignored since they no longer will ever reach one and thus will not influence the behaviour of the algorithm. Therefore, by eliminating non-producer counters and by "sending" non-messages, algorithm SSTC can be rewritten (slightly), without changing its behaviour *for use after time* $3n$, as follows:

Algorithm 2 DUAL-SSTC

```
1  repeat for every time step:
2  processor(Prod?,proc_flip,proc_count) receives message(Mess?,mess_flip,mess_count)
3  case (Prod?, Mess?)
4      (TRUE, TRUE)
5          {if not (mess_flip=HEAD and proc_flip=TAIL)
6              flip← coin-flip
7              (new-proc, new-mess) ← ((TRUE, flip, n), (TRUE, flip, n))
8          else
9              (new-proc, new-mess) ⟵ ((FALSE, *, *), (FALSE, *, *)}
10     (FALSE, TRUE)
11         {if (mess_count=1)
12             flip← coin-flip
13             (new-proc, new-mess) ← ((TRUE, flip, n), (TRUE, flip, n))
14         else
15             (new-proc, new-mess) ← ((FALSE, *, *), (TRUE, mess_flip, mess_count−1))
               }
16     (TRUE, FALSE)
17         {if (proc_count=1)
18             flip← coin-flip
19             (new-proc, new-mess) ← ((TRUE, flip, n), (TRUE, flip, n))
20         else
21             (new-proc, new-mess)←((TRUE, proc_flip, proc_count−1),(FALSE,*,*))
               }
22     (FALSE, FALSE)
23         { (new-proc, new-mess) ← ((FALSE, *, *), (FALSE, *, *)) }
24 processor-state ← new-proc; send (new-mess)
```

Define a *local configuration* of a processor, α, to be a pair (PS, MS) where PS is the processor-state of α and MS is the message-state of the message at α. (If there is no message at processor α then the message-state at α is represented by (FALSE, *,*).) A *configuration* of the ring is given by a *cyclic* sequence of n local configurations.

Let $\chi = (\text{PS}_1, \text{MS}_1), \cdots, (\text{PS}_n, \text{MS}_n)$ be a ring configuration. Given χ,

one iteration of the repeat loop of DUAL-SSTC determines the new ring-configuration by *updating* each local configuration $(\mathrm{PS}, \mathrm{MS})$ in χ according to which of the four possible boolean combinations holds for (PS.Proc?, MS.Mess?) and then *shifting* each message component in the updated configuration one position to the right. We capture these actions by defining:

The update function U by: $\mathrm{U}(\chi) = \mathrm{U}\left((\mathrm{PS}_1, \mathrm{MS}_1), \cdots, (\mathrm{PS}_n, \mathrm{MS}_n)\right) = \left(\mathrm{u}(\mathrm{PS}_1, \mathrm{MS}_1), \cdots, \mathrm{u}(\mathrm{PS}_n, \mathrm{MS}_n)\right)$. where $\mathrm{u}(\mathrm{PS}, \mathrm{MS})$ produces the new local configuration that results from applying the appropriate case from the code of DUAL-SSTC .

The shift function Sh by: $\mathrm{Sh}(\chi) = \left((\mathrm{PS}_1, \mathrm{MS}_n), \cdots, (\mathrm{PS}_n, \mathrm{MS}_{n-1})\right)$.

Further, define:

The interchange function C by: $\mathrm{C}(\chi) = (\mathrm{MS}_1, \mathrm{PS}_1), \cdots, (\mathrm{MS}_n, \mathrm{PS}_n)$.

The reverse-shift function RSh by: $\mathrm{RSh}(\chi) = (\mathrm{PS}_1, \mathrm{MS}_2), \cdots, (\mathrm{PS}_n, \mathrm{MS}_1)$.

The inverse function Inv by: $\mathrm{Inv}(\chi) = (\mathrm{PS}_n, \mathrm{MS}_n), \cdots, (\mathrm{PS}_1, \mathrm{MS}_1)$.

For any configuration χ, let $(f \circ g)(\chi)$ denote the function composition $f(g(\chi))$.

Claim 7.1 *Let χ be any configuration, t be any time step greater than $3n$, and E be a fixed infinite sequence of coin tosses. Then Config$(\chi, E, t+1)$ = $(\mathrm{Sh} \circ \mathrm{C} \circ \mathrm{U} \circ \mathrm{C})($Config$(\chi, E, t))$.*

Proof: Since $t \geq 3n$, $\mathrm{Config}(\chi, E, t+1)$ can be computed using algorithm DUAL-SSTC . Therefore $\mathrm{Config}(\chi, E, t+1) = (\mathrm{Sh} \circ \mathrm{U})(\mathrm{Config}(\chi, E, t))$. The equality then follows by examining algorithm DUAL-SSTC . For all cases of (Prod?, Mess?), after one iteration beginning with processor-state PS and message-state MS, (new-proc, new-mess) $= \mathrm{u}(\mathrm{PS}, \mathrm{MS}) = \mathrm{C}(\mathrm{u}(\mathrm{MS}, \mathrm{PS}))$. ■

Let id denote the identity function. The following identities are immediate from the definitions.

identity 1: $\mathrm{id} = \mathrm{Inv} \circ \mathrm{Inv}$ **identity 4:** $\mathrm{Inv} \circ \mathrm{C} = \mathrm{C} \circ \mathrm{Inv}$

identity 2: $\mathrm{id} = \mathrm{C} \circ \mathrm{C}$ **identity 5:** $\mathrm{Sh} = \mathrm{Inv} \circ \mathrm{RSh} \circ \mathrm{Inv}$

identity 3: $\mathrm{Inv} \circ \mathrm{U} = \mathrm{U} \circ \mathrm{Inv}$ **identity 6:** $\mathrm{RSh} = \mathrm{C} \circ \mathrm{Sh} \circ \mathrm{C}$

Theorem 7.2 *Let χ be any configuration, t be any time step greater than $3n$, and E be a fixed infinite sequence of coin tosses. Then for every $s \geq 0$*

$$\textit{Config}(\chi, E, t+s) = (\mathrm{Inv} \circ \mathrm{C})\left(\textit{Config}((\mathrm{C} \circ \mathrm{Inv})(\textit{Config}(\chi, E, t), E, s))\right).^{1}$$

[1] We assume that E always denotes the *remaining* sequence of coin-flips; the ones already used are deleted from the front of the sequence.

Proof: The configuration s steps after $\text{Config}(\chi, E, t)$ is produced by s applications of $(\text{Sh} \circ \text{U})$. That is $\text{Config}(\chi, E, t+s) = (\text{Sh} \circ \text{U})^s(\text{Config}(\chi, E, t))$. However,

$$\begin{aligned}
\text{Sh} \circ \text{U} &= \text{Inv} \circ \text{RSh} \circ \text{Inv} \circ \text{C} \circ \text{U} \circ \text{C} && \text{by identity 5 and claim 7.1} \\
&= \text{Inv} \circ \text{C} \circ \text{Sh} \circ \text{C} \circ \text{Inv} \circ \text{C} \circ \text{U} \circ \text{C} && \text{by identity 6} \\
&= \text{Inv} \circ \text{C} \circ \text{Sh} \circ \text{Inv} \circ \text{U} \circ \text{C} && \text{by identities 4 and 2} \\
&= \text{Inv} \circ \text{C} \circ \text{Sh} \circ \text{U} \circ \text{C} \circ \text{Inv} && \text{by identities 3 and 4.}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\text{Config}(\chi, E, t+s) &= (\text{Sh} \circ \text{U})^s(\text{Config}(\chi, E, t)) \\
&= (\text{Inv} \circ \text{C} \circ \text{Sh} \circ \text{U} \circ \text{C} \circ \text{Inv})^s(\text{Config}(\chi, E, t)) \\
&= (\text{Inv} \circ \text{C}) \circ (\text{Sh} \circ \text{U})^s \circ (\text{C} \circ \text{Inv})(\text{Config}(\chi, E, t))) \\
&\qquad \text{by identities 1 and 2.} \\
&= (\text{Inv} \circ \text{C})\left(\text{Config}((\text{C} \circ \text{Inv})(\text{Config}(\chi, E, t)), E, s)\right).
\end{aligned}$$

■

Lemma 7.3 *For every configuration χ and for every $r > 3n$, if $Env(\chi, r) \geq Prod(\chi, r)$ then no producers time-out in the first round after r in which any time-outs occur.*

Proof: Let χ be a configuration satisfying $\text{Env}(\chi, r) \geq \text{Prod}(\chi, r)$ for $r \geq 3n$, and suppose that the first time-out after r is at time $r + s$. Let $\hat{\chi} = (\text{C} \circ \text{Inv})(\text{Config}(\chi, E, r))$. Note that $\text{Env}(\hat{\chi}, 0) \leq \text{Prod}(\hat{\chi}, 0)$. By Theorem 7.2, $\text{Config}(\chi, E, r+s) = (\text{Inv} \circ \text{C})(\text{Config}(\hat{\chi}, E, s))$. Thus, if the time-out at step $r + s$ from χ is a producer, then there is a producer time-out for $(\text{Inv} \circ \text{C})(\text{Config}(\hat{\chi}, E, s))$. implying that there is a message-time out at step s from $\hat{\chi}$. Furthermore, this must be the first time-out for $\hat{\chi}$ since otherwise there would have been a time-out earlier than s steps after r for χ. However, by lemma 4.4, the first thing to time out for $\hat{\chi}$ cannot be a message, implying that the time out for χ was not a producer. ■

Lemma 7.4 *For every configuration and for every χ $r \geq 3n$, if $Env(\chi, r) > Prod(\chi, r)$ and the first time-out after r is at time T, then $Env(\chi, T) \geq Prod(\chi, T)$.*

Proof: Let χ be a configuration satisfying $\text{Env}(\chi, r) > \text{Prod}(\chi, r)$ for $r \geq 3n$, and suppose that the first time out after r is at time $r + t$. Let $\hat{\chi} = (\text{C} \circ \text{Inv})(\text{Config}(\chi, E, r))$. Note that $\text{Env}(\hat{\chi}, 0) < \text{Prod}(\hat{\chi}, 0)$. By Theorem 7.2, $\text{Config}(\chi, E, r+t) = (\text{Inv} \circ \text{C})(\text{Config}(\hat{\chi}, E, t))$. By Lemma 4.5, at the first time-out, at time $\hat{t}$, $\text{Env}(\hat{\chi}, \hat{t}) \leq \text{Prod}(\hat{\chi}, \hat{t})$. Furthermore $\hat{t}$ must equal t because otherwise Theorem 7.2 implies there would have been an earlier time out for χ. So $\text{Env}(\hat{\chi}, t) \leq \text{Prod}(\hat{\chi}, t)$ and thus the number of producers in $(\text{Inv} \circ \text{C})(\text{Config}(\hat{\chi}, E, t))$ is less than or equal to the number of messages in $(\text{Inv} \circ \text{C})(\text{Config}(\hat{\chi}, E, t))$. That is $\text{Env}(\chi, r+t) \geq \text{Prod}(\chi, r+t)$. ■