International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 4, Number 1, pages 42–52, January 2014

Self-Stabilizing Master-Slave Token Circulation Algorithm in Undirected Rings and Unicyclic Graphs of Arbitrary Size and Their Orientations

Yihua Ding, James Wang and Pradip K Srimani
School of Computing, Clemson University
Clemson, SC, 29634, USA

Received: July 19, 2013
Revised: October 4, 2013
Accepted: November 29, 2013
Communicated by Susumu Matsumae

Abstract

Token circulation is a fundamental task in the distributed systems. In this paper, we propose a constant space randomized self-stabilizing master-slave token circulation algorithm that works for undirected rings and undirected unicyclic graphs of arbitrary size. We consider the recently introduced and studied master-slave model where a single node is designated to be a master node and other nodes are anonymous slave nodes. The expected stabilization time is $O(n \log n)$ steps, and the space requirement at each node is 4 bits for any undirected ring (or unicyclic graph) of size n under an unfair distributed daemon; the nodes do not need the knowledge of the size of the ring and hence the protocol is suited for dynamic graphs. The proposed token circulation algorithm is further extended to achieve orientation in the ring and the unicyclic graph. Disregarding the time for stabilization, the orientation can be done in at most $O(n)$ steps with 1 bit extra storage at each node for the ring and the unicyclic graph.

Keywords: Self-stabilization, Master-slave Model, Token Circulation, Orientation, Undirected Ring, Undirected Unicyclic Graph

1 Introduction

Self-stabilization has been widely used to build various fault tolerant distributed systems since it was introduced by Dijkstra in 1974 [9]. An algorithm is self-stabilizing iff it reaches some legitimate state starting from an arbitrary state. Without any external intervention, a self-stabilizing system can recover from an unlimited number of transient faults (transient fault is an event of corrupting the data but not the program code). The self-stabilizing algorithm can be either uniform or semi-uniform. An algorithm is uniform if all nodes use the same program, and is semi uniform if there exist a single "special" node that behaves differently from the rest of the nodes [5].

Token circulation is a fundamental task in the distributed systems. It has many applications such as mutual exclusion, size computation of system, etc [6]. It has been shown that no uniform deterministic self-stabilizing algorithm for token circulation exists for an undirected ring of even size [5]. Recently, a self-stabilizing algorithm for token circulation in a unidirectional ring of arbitrary size is proposed, which breaks symmetry by using the master-slave model [21] and employing randomization. The ring has a designated node called master node, and other nodes are uniform and anonymous, called slave nodes. The master-slave model is especially suitable for sensor network

or ad hoc networks based on cluster architecture, where a base station or a cluster head has more memory, and is more powerful than other ordinary nodes [8].

In this paper, we propose a new randomized self-stabilizing algorithm for token circulation in an undirected ring of arbitrary size. The proposed algorithm is based on the master-slave model, and thus is a semi-uniform algorithm. The ring is undirected, i.e., each node has two neighbors but does not have specified predecessor or successor. Intuitively, the approach is to impose a unidirectional Eulerian cycle (starting and ending at one end of the master node) on the undirected ring, and then single token traverses the nodes along the directed unidirectional cycle. The expected stabilization time is $O(n \log n)$ steps, and the space requirement at each node is 4 bits, i.e., the space complexity at each node is constant for an undirected ring of size n. The token circulation algorithm is then extended to achieve ring orientation such that each node in the ring have a *consistent* notion of its predecessor and successor (consider an edge (i, j): if node i considers node j as its successor, node j regards node i as its predecessor). Disregarding the time for stabilization, the ring orientation is completed in at most $3n-1$ steps with 1 extra bit storage at each node. The proposed algorithms do not need to know the size of the ring and hence can tolerate dynamic addition and removal of nodes as long as the ring topology is preserved. The protocol works under both central daemon (only one privileged node is selected to move at each step) and distributed daemon (an arbitrary non-empty subset of privileged nodes is selected to move at each step).

We further show that the token circulation algorithm also works for unicyclic graphs using the same principle and extend the token circulation algorithm to orient the unicyclic graph in the sense that each node in the graph has a consistent notion of its predecessor and successor.

2 Related Works

Self-stabilizing token circulation in a unidirectional ring was first introduced in [9], where one algorithm used one distinguished node and n states at each node while the other two used 3 and 4 states at each node but with two distinguished nodes. Subsequently, both deterministic [2] and randomized [13, 19, 11, 20, 3, 7, 4] algorithms for unidirectional rings are proposed. The algorithms in [13, 19, 20, 3, 7, 4] are uniform, while the ones in [11, 2] are semi-uniform. All of the above algorithms except [13] can accommodate rings of arbitrary size, while [13] is limited to rings of odd size. The best reported stabilization time is $O(n^2)$ and the space requirement at each node varies from $2n$ to 2 states. To the best of our knowledge, there is no known self-stabilizing token circulation algorithm in undirected rings.

Another related problem is ring orientation. A ring is oriented if all nodes in the ring have a consistent notion of their predecessors and successors. Authors in [18] show that no uniform deterministic self-stabilizing orientation algorithm exists for rings of arbitrary size and provide a randomized solution. Under distributed daemon, it stabilizes in $O(n^2)$ expected time, and requires 5 bits of storage at each node (each node has 18 local states), where n is the size of ring. A similar randomized solution is presented in [17] which has the same performance as [18]. For rings of odd size, there exist several deterministic self-stabilizing algorithms [15, 22, 16] for ring orientation.

3 Model and Terminology

We consider an undirected ring of size n with a designated node labeled as the **master** node, node 0, and the other nodes are labeled 1 through $n-1$. These node labels are arbitrary and are not used by the algorithms; they are used for reference purposes only. The master node is unique, it knows it's the master and the other nodes are anonymous; we call them **slave** nodes. Each node i, $0 \leq i \leq n-1$ in the undirected ring R has two neighbors; for each node i, there is some fixed but arbitrary local ordering of the neighbors of node i numbered 0 and 1; the local ordering of neighbors of a node is independent of such local ordering of neighbors at any other node in the ring. Figure 1(a) shows an example ring with arbitrary neighbor ordering at the nodes of a ring (the nodes are numbered from 0 through $n-1$ consecutively for notational convenience only).

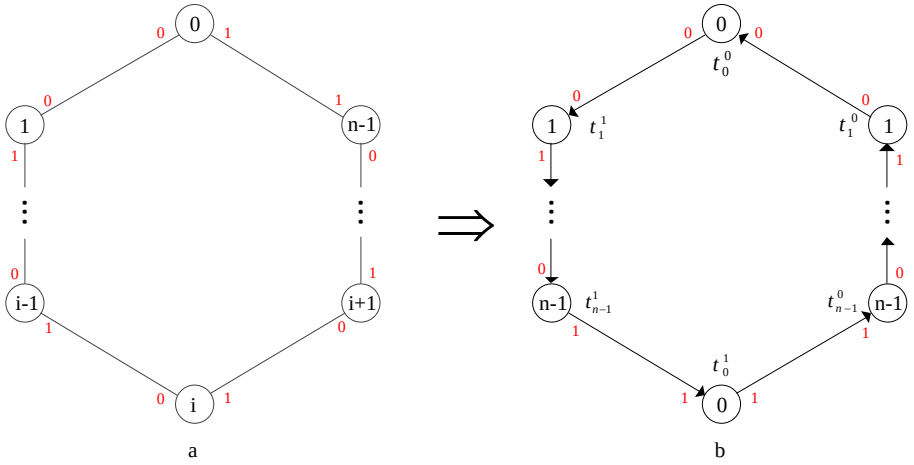

Figure 1: A Ring R and the Directed Eulerian Cycle

As is traditional with many self-stabilizing algorithms, we use the *shared-memory model*: a node communicates with its neighbor(s) by reading and writing on a shared pair of registers; a node reads the states of all its neighbors and write on its state variables, all in one atomic step using composite atomicity [14]. The proposed algorithm does not need to know the size of the ring, it tolerates dynamic additions and removals of nodes as long as the ring topology is preserved. We assume that the behavior of nodes are managed by an adversarial daemon: the daemon selects node(s) from among the privileged nodes to perform local actions. The daemon is called *fair* if it prevents a node being continuously privileged without performing local actions. Otherwise, the daemon is called *unfair*. The proposed algorithm works under both the unfair *central daemon* (only one privileged node is selected to perform local action at each step) as well as the unfair *distributed daemon* (a non-empty subset of privileged nodes can be selected to perform local actions at each step). We focus on central daemon in order to analyze the time complexity of the algorithm and then show how to extend the analysis for a distributed daemon as well.

4 Token Circulation in Undirected Rings

We generalize the concept of master–slave token circulation algorithm in a unidirectional (directed) ring given in [12] and generalize it for adoption in an undirected (bidirectional) ring; we call the result the `TC` algorithm.

In algorithm `TC`, each node i has two t-variables t_i^0 and t_i^1, each of which can be in one of three states, say $\mathbb{X}, \mathbb{Y}$, or $\mathbb{Z}$; we use notations s_i^k, $k \in \{0,1\}$ to denote the state of the k-neighbor of node i as sensed (read) at node i. We observe:

- Consider an edge (i,j) in the ring between nodes i and j: we view this undirected edge as two parallel directed edges: $i \to j$ and $j \to i$. Assume node j is the k-neighbor of node i, $k \in \{0,1\}$ (according to the unique ordering of neighbors at node i) and node i is the k'-neighbor of node j (according to the unique numbering of neighbors at node j), where $k, k' \in \{0,1\}$; then, s_i^k is the value of $t_j^{k'}$ (stored variable at node j) as read by node i; similarly $s_j^{k'}$ is the value of t_i^k (stored variable at node i) as read by node j. The s-variables are used only for ease of notation.

- Pairs of corresponding token variables at neighboring nodes simulate directed edges in the ring and essentially impose a directed Eulerian cycle on the ring starting at the 0-neighbor of the master node using a directed walk along the ring starting and ending at the 0-th neighbor of the master node. Note that the length of the Eulerian cycle is $2n$ and each node of the ring is

```
Rule for the Master Node (i = 0):
    { if (t_0^0 = s_0^0)
          then set t_0^0 to {𝕏, 𝕐, ℤ} − {t_0^0} uniformly at random;
      if (t_0^1 ≠ s_0^1)
          then set t_0^1 = s_0^1;

Rule for a Slave Node (i > 0):
    { if (t_i^0 ≠ s_i^1)
          then set t_i^0 = s_i^1;
      if (t_i^1 ≠ s_i^0)
          then set t_i^1 = s_i^0;
```

Figure 2: Algorithm TC for Token Circulation in An Undirected Ring

visited twice in the complete Eulerian cycle. The Eulerian cycle corresponding to the ring in Figure 1(a) is shown in Figure 1(b).

The rules for a slave node i, $0 < i < n$ is as follows: For any k, $k \in \{0, 1\}$, if $t_i^k \neq s_i^{k'}$, then $t_i^k = s_i^{k'}$, where $k' = 1 - k$. When a node senses that the token variable at one of its two neighbors is different from its own token reserved for its other neighbor, we say that node i has received a token from its neighbor (**the node i is privileged**) and it changes its own token for the next neighbor to match with the sensed token; we say node i passes the received token to its next neighbor.

The rule for the master node is somewhat different: The master node ($i = 0$) treats its two neighbors differently: (1) we say the master node receives a token (**node 0 is privileged**) from its 1-neighbor when it senses that the token variable at its 1-neighbor is different from its its own token reserved for its 1-neighbor (i.e., $t_0^1 \neq s_0^1$); it then passes the token to its 1-neighbor by executing $t_0^1 = s_0^1$; (2) the master node receives a token (**node 0 is privileged**) from its 0-neighbor when it senses that the token variable at its 0-neighbor is same as its own token reserved for its 0-neighbor (i.e., $t_0^0 = s_0^0$); it passes the token to its 0-neighbor by updating its token by randomly choosing one of the two remaining token values. The resulting algorithm TC is shown in Figure 2.

Remark 1

1. When a privileged node i, $0 \neq i < n$, is tapped by the central daemon, it becomes unprivileged, i.e., does not hold any token in the next system state (the central daemon taps only one node at a time).
2. A slave node receives a token from one neighbor and moves it to its other neighbor along the directed Eulerian cycle.
3. The master node also receives a token from one neighbor and moves the token to its neighbor along the directed Eulerian cycle; but, since the master node behaves differently, it always passes the token to the same neighbor from which it receives the token (see Figure 1(b) for the Eulerian cycle of our example ring).
4. Any token in the system will move along the directed Eulerian cycle, if it does not die along the way and will eventually reach the master node via its 0-neighbor.

The desired goal behavior is that the master node changes its t_0^0 value, this token then circulates the entire directed Eulerian cycle, and then the master node changes its value again. Thus, we have a legitimate state (final configuration) iff the number of tokens in the system is 1.

Definition 1 *A global system state is* **legitimate** *or* **stable** *if and only if there is exactly one token in the unidirectional Eulerian cycle; in a legitimate state, the number of privileged nodes (the master or a slave) is exactly one.*

Lemma 1 *In any arbitrary system state, each node has at most two tokens; thus the total number of tokens in the ring is at most* $2n$.

Proof: Each node has exactly two neighbors and a node can have at most one token corresponding to a neighbor. The proof follows. □

Lemma 2 *The number of tokens in the system cannot increase.*

Proof: When a slave node i $(0 < i < n)$ is privileged, i.e., it has a token, say from its k-th neighbor, $k \in \{0, 1\}$ and it is chosen by the daemon to move, it becomes unprivileged, i.e., it loses its token(s); each such token either dies or is received only by one neighbor of node i (the unique k'-neighbor, $k' = 1 - k$, i.e., the successor of node i in the unidirectional Eulerian cycle), thus making that neighbor privileged. Thus the number of tokens can never go up. Similarly, when the master node has only one token from its 1-neighbor, and is chosen by the daemon to move, the token either dies or the neighbor receives the token; when the master node has a token from its 0-neighbor and is chosen to move, the token dies and its 0-neighbor receives a token. Thus, the number of tokens can never go up. □

Lemma 3 *The* `TC` *algorithm cannot terminate, i.e., in any system state at least one node will be privileged.*

Proof: Consider the case when no slave node is privileged (by either of its neighbors) and the master node is not privileged by its 1-neighbor. [Consider our example in Figure 1(b); when no slave node is privileged we must have $t_1^0 = t_2^0 = t_3^0 = \cdots t_{n-1}^0 = t_0^1$ and $t_{n-1}^1 = t_{n-2}^1 = \cdots t_1^1 = t_0^0$; since the master node 0 is not privileged by its 1-neighbor, we must have $t_0^1 = t_{n-1}^1$] Then, both token variables at each slave node and the master node are identical. Thus, $t_0^0 = s_0^0$, i.e., the master node is privileged by its 0-neighbor; when it moves, the 0-neighbor of the master node will be privileged. □

Lemma 4 *Starting from an arbitrary system state, the master node is forced to send a new token to its* 0*-neighbor after finitely many (actually* $O(n)$*) moves.*

Proof: After a privileged node i $(0 \leq i < n)$ performs its local actions, it becomes unprivileged, and does not hold any token under central daemon. Any token in the system takes at most $2n$ steps to reach the 0-neighbor of the master node (along the directed Eulerian cycle) or it dies along the way. □

Thus, in order to reach a legitimate state starting from an arbitrary system state, we need to show that the number of privileged nodes decreases. It has been shown in [1, 12] that it is possible to enter into an infinite loop if what enters the master node must continually be the same what leaves it; thus for converging to a legitimate state, it is necessary and sufficient to choose a sequence for the t_0^0 such that the sequence of tokens is aperiodic. The randomization incorporated in the protocol for the master node provides the necessary aperiodicity of the sequence of tokens at the master node for convergence of the global state to a legitimate one starting from an arbitrary state. It is to be noted that if the sequence of distinct values of t_0^0 is periodic, there always exists some initial global system state in an undirected ring such that the protocol does not converge to a legitimate final state; This can be easily proved using a similar theorem given in [12] by replacing an unidirectional ring with the unidirectional Eulerian cycle imposed on the undirected ring. We omit the details for lack of space.

Definition 2 *A round is defined as the time taken by a token to circulate around the unidirectional cycle (of size* $2n$*) imposed on the given ring. Thus each round is equivalent to* $2n$ *steps using central daemon.*

Theorem 1 *The algorithm* `TC` *stabilizes in* $O(n \log n)$ *expected time using central daemon, i.e., the expected time to converge to one circulating token is* $O(n \log n)$ *steps regardless of the behavior of the daemon.*

Proof: The goal of the daemon is to maintain multiple tokens in circulation. Most moves simply advance a token from one node to the next without changing the set of tokens. The moves that can potentially destroy a token are (a) tapping the master node; or (b) advancing one token to overwrite the next (for example, if three consecutive slave nodes have $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$ and the daemon taps the middle node then the $\mathbb{Y}$ token dies.

Consider a situation where the daemon performs (a). Say the first token in circulation is $\mathbb{X}$ and the daemon and its predecessor are in $\mathbb{Y}$. Then there is a 50% chance that the token generated by the daemon is the same as that, and hence the token dies. Thus, if the daemon never does (b), then the expected number of tokens generated by the master node before we get to one token is at most $2n$. Thus, approximately, if we start with $2n$ tokens, after all have passed through the master we should expect to have about n tokens, and so on, and so after $O(\log n)$ rounds we would have only one token. To make the previous sentence more mathematically accurate, if we consider that there is a 50% chance that the token dies when it passes through the master, then the number of rounds that a token lives is a geometric random variable with mean 2. Since the coin tosses (randomization) at the master node are independent of each other, the expected number of rounds for all tokens to die is then the expected value of the maximum of n IID geometric random variables, and this is known to be $O(\log n)$ [10].

To conclude the proof, we must consider that in our case convergence occurs when the penultimate token dies (see Lemma 3). Second, the daemon can destroy any token at any time. But this can only decrease the expected running time. □

Remark 2 While it is possible that all nodes are privileged in an initial arbitrary system state, this state cannot continue indefinitely due to the randomization introduced to generate the new token for the 0-th neighbor of the master node in `TC` algorithm.

Remark 3 The `TC` algorithm operates correctly under a distributed daemon. If the distributed daemon does not choose all the nodes simultaneously, then in any unidirectional cycle algorithm (the tokens in the algorithm always move along the unidirectional Eulerian cycle imposed on the undirected ring), this is equivalent to a sequence of individual moves (move the frontmost node first, then the second and so on). So the only issue with considering a distributed daemon is what happens if the daemon selects all nodes simultaneously. This can, of course, only happen if every node is privileged. And we have just argued that that state cannot continue indefinitely.

5 Ring Orientation

In this section, we extend the token circulation algorithm `TC` to orient the original undirected ring. After the `TC` algorithm reaches a legitimate state, ring orientation stabilizes in at most $3n - 1$ steps, i.e., all nodes in the ring have a consistent notion of their predecessors and successors.

The idea is simple: each node in the ring R has an extra orientation bit indicating its predecessor: if the orientation bit of node i is k, $k \in \{0, 1\}$, node i considers its k-neighbor as its predecessor. The master node behaves differently; it unilaterally decides to consider its 1-neighbor as its predecessor. When the system stabilizes (the system has only one token), the master node broadcasts its own orientation preference to the slave nodes via the unique token so that the predecessor and successor relationships at each node become consistent.

5.1 Algorithm `ORIENT`

The integrated token circulation and the orientation algorithm `ORIENT` for the ring is shown in Figure 3. In the `ORIENT` algorithm, each node i maintains the data structure:

- Two 2-bit t-variables t_i^0 and t_i^1, each of which can have a value from $\{\mathbb{X}, \mathbb{Y}, \mathbb{Z}\}$, as before.
- A 1-bit boolean flag $orient_i$, which indicates the predecessor of node i.

```
Rule for the Master Node (i = 0):
    { if (t_0^0 = s_0^0) then
          { set t_0^0 to {𝕏, 𝕐, ℤ} − {t_0^0} uniformly at random;
            orient_0 = 1;
      if (t_0^1 ≠ s_0^1) then
          set t_0^1 = s_0^1;

Rule for a Slave Node (i > 0):
    { for k = 0 to 1 do
          if (t_i^k ≠ s_i^{1−k})
          { if (t_i^0 = t_i^1) then orient_i = 1 − k;
            set t_i^k = s_i^{1−k};
```

Figure 3: Algorithm `Orient` for Orientation of An Undirected Ring

The master node 0, when it is privileged by its 0-neighbor and is tapped by the daemon, sets $orient_0 = 1$. Any slave node $i > 0$, when it is privileged by its k-neighbor, $k \in \{0, 1\}$ and $t_i^0 = t_i^1$, i.e., when both resident tokens at node i are equal, it sets $orient_i = k$.

5.2 Analysis

Theorem 2 *After the system reaches a legitimate state, all nodes in the ring have a consistent notion of their predecessors and successors in at most* $3n - 1$ *steps.*

Proof: The master node $i = 0$ will be privileged in at most $2n$ steps after the system stabilizes (reaches a state when there is only one token in the system). Thus, $t_i^0 = s_i^0$ and all t-variables at all nodes in the system are equal by Lemma 3 (none of the slave nodes is privileged). The master node sends a new token to its 0-neighbor, say node j; node j becomes privileged by the token received from one of its neighbor; at this point, the two resident tokens at node j are equal; it then adjusts its $orient_j$ bit and passes the token to its other neighbor. The process continues until the unique token in the system reaches the 1-neighbor of the master node in $n - 1$ steps and all slave nodes have adjusted their *orient* bits in a consistent way since the token moves along the directed Eulerian cycle. The token then moves along the other half of the Eulerian cycle visiting the slave nodes in the opposite sequence; during this traversal, when the token visits any slave node the resident tokens at that node are not equal and hence the *orient* bit is not adjusted again. It is obvious that *orient* bits are not changed ever again as long the system is in a legitimate state. □

6 Unicyclic Graph Orientation

We extend our token circulation algorithm and subsequent orientation of an undirected ring to a larger class of graphs, e.g., that of undirected unicyclic graphs. An undirected graph is unicyclic when there is a single cycle in the graph. We consider a subclass of the general unicyclic graphs where there is at least one node in the cycle with two neighbors, i.e., the node has no subtree rooted at it. Figure 4(a) shows an example of such a unicyclic graph. Orientation of such a unicyclic graph is defined to be a consistent orientation of the edges in the unique cycle (each node in the cycle will know its unique predecessor) and the subtree of a node in the cycle will be rooted at that node (each node in the subtree will know its unique predecessor in the rooted subtree).

In a unicyclic graph with n nodes, an arbitrary node on the unique cycle with two neighbors is designated as the master node, labeled 0 and the other nodes are labeled 1 through $n - 1$; they are called slave nodes as before. These node labels are arbitrary and are not used in the algorithms; they are used for reference purposes only. Each node i has δ_i many neighbors (note that $\delta_0 = 2$, and

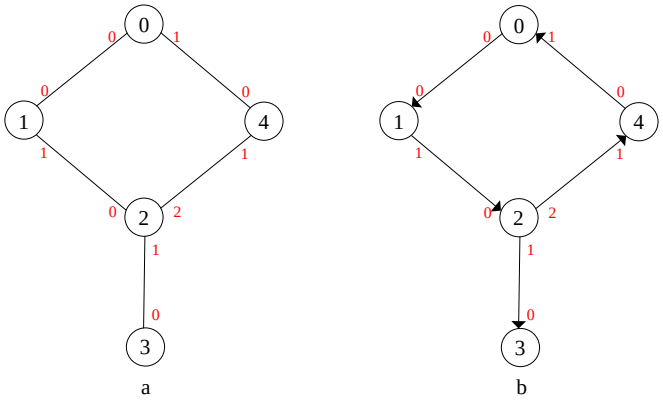

Figure 4: A Unicyclic Graph and Its Orientation

for node i, $1 < i < n$, $\delta_i \geq 2$ if node i belongs to the cycle and $\delta_i \geq 1$ otherwise. For each node i, there is some fixed but arbitrary ordering of the neighbors of node i numbered 0, 1, ..., $\delta_i - 1$. The neighbor ordering scheme is the same as in the previous section.

Each undirected edge (i, j) between nodes i and j is viewed as two parallel directed edges in the opposite directions. Given the neighbor orderings at the nodes in the unicyclic graph, one can impose a directed Eulerian cycle using a depth first walk along the unicyclic graph starting and ending at the master node. Note that the length of the Eulerian cycle is $2n$ and each edge of the unicyclic graph is visited twice in the complete Eulerian cycle.

Remark 4 The directed Eulerian cycle is different for different choices of neighbor orderings at slave nodes in the unicyclic graph, but the resulting orientation remains the same as long as the neighbor ordering at the master node is fixed.

Figure 4(a) shows an example unicyclic graph with arbitrary neighbor orderings at the nodes of the unicyclic graph. The Eulerian cycle imposed on the unicyclic graph is given by 0, 1, 2, 3, 2, 4, 0, 4, 2, 1, 0. If the ordering of neighbors of node 2 such that neighboring nodes 1, 4, 3 are numbered 0, 1, 2 respectively, then the imposed Eulerian cycle is given by 0, 1, 2, 4, 0, 4, 2, 3, 2, 1, 0. Although the imposed Eulerian cycles are different for different choices of neighbor orderings at slave nodes, the resulting orientation of the unicyclic graph remains the same as shown in Figure 4(b) (if (i, j) is a tree edge, the arrow $i \rightarrow j$ indicates i is a predecessor of node j).

The underlying concept in token circulation and subsequent orientation of the unicyclic graph is similar to token circulation and orientation of the ring:

(a) Impose a directed Eulerian cycle on the unicyclic graph starting at the designated master node, visiting each edge of the graph and coming back to the master node using a depth first walk. The previous token circulation algorithm in a ring can be readily adapted to a slightly restrictive class of unicyclic graphs where the unique cycle has at least one node with exactly degree two.

(b) When the system stabilizes to the legitimate state with a single token, the unique token travels along the imposed directed Eulerian cycle repeatedly.

(c) Each node has an extra orientation bit indicating its predecessor: if the orientation bit of node i is k, $0 \leq k < \delta_i$, node i considers its k-neighbor as its predecessor.

(d) The master node behaves differently; it unilaterally decides to consider its 1-neighbor as its predecessor. The master node broadcasts its own orientation preference to the slave nodes via the unique token so that the predecessor and successor relationships at each node become consistent.

```
Rule for the Master Node (i = 0):
    { if (t_0^0 = s_0^0) then
          { set t_0^0 to {𝕏, 𝕐, ℤ} − {t_0^0} uniformly at random;
            orient_0 = 1;
      if (t_0^1 ≠ s_0^1) then
          set t_0^1 = s_0^1;

Rule for a Slave Node (i > 0):
    { for k = 0 to δ_i − 1 do
          if (t_i^k ≠ s_i^((k−1) mod δ_i))
          { if (t_i^k = t_i^((k−1) mod δ_i)) then orient_i = (k − 1) mod δ_i;
            set t_i^k = s_i^((k−1) mod δ_i);
```

Figure 5: Algorithm `Orient2` for Orientation of A Unicyclic Graph

6.1 Algorithm `ORIENT2`

The integrated token circulation and the orientation algorithm `ORIENT2` for the undirected unicyclic graph is show in Figure 5. In the `ORIENT2` algorithm, each node i maintains the following data structure:

- 2-bit t-variables $t_i^0, t_i^1, ..., t_i^{\delta_i - 1}$, each of which can have a value from $\{\mathbb{X}, \mathbb{Y}, \mathbb{Z}\}$, as before.
- A 1-bit boolean flag $orient_i$, which indicates the predecessor of node i.

The master node 0, when it is privileged by its 0-neighbor and is tapped by the daemon, sets $orient_0 = 1$. Any slave node $i > 0$, when it is privileged by its k-neighbor and $t_i^{(k+1) \bmod \delta_i} = t_i^k$, it sets $orient_i = k$. The complete pseudo code of the algorithm is shown in Figure 5.

6.2 Analysis

Theorem 3 *Remarks 1-3, Lemmas 1-4 and Theorem 1 hold for Algorithm `ORIENT2`.*

Proof: The proofs are similar to those for Algorithm `TC`. The only difference is that the directed Eulerian cycle imposed on the ring by Algorithm `TC` is replaced by the directed Eulerian cycle imposed on the unicyclic graph by Algorithm `ORIENT2`. □

Remark 5 Any edge (i, j) of the unicyclic graph is visited exactly twice in the complete Eulerian cycle: (1) first time as $i \to j$ when the edge is from the predecessor to the successor (with respect to the master node); and (2) second time as $j \to i$ when the edge is from the successor to the predecessor.

Lemma 5 *When a slave node j receives a token from its k-neighbor, say node i after the system reaches a legitimate state. If $t_j^{(k+1) \bmod \delta_j} = t_j^k$, then the edge (i, j) is visited by the token for the first time in the complete Eulerian cycle, i.e., node i is the predecessor of node j with respect to the master node (see Remark 5).*

Proof: When a slave node j receives a token from its k-neighbor, node j must have $t_j^{(k+1) \bmod \delta_j} \neq s_j^k$ (see the rule for a slave node), where the value of s_j^k is matched with the circulating token (either type $\mathbb{X}$, $\mathbb{Y}$ or $\mathbb{Z}$). If the edge (i, j) has been visited by the token before from node j to i, then the value of t_j^k must be matched with the unique t-token, it follows that $t_j^k = s_j^k$, further, $t_j^{(k+1) \bmod \delta_j} \neq t_j^k$. □

Theorem 4 *After the system reaches a legitimate state, all nodes in the unicyclic graph have a consistent notion of their predecessors and successors in at most* $4n - 1$ *steps.*

Proof: The master node $i = 0$ will be privileged in at most $2n$ steps after the system stabilizes to a state when there is only one token in the system. Thus, $t_i^0 = s_i^0$ and all t-variables at all nodes in the system are equal by Lemma 3 (none of the slave nodes is privileged). The master node sends a new token to its 0-neighbor, say node j; node j becomes privileged by the token received from the master node (without loss of generality, the master node is assumed to be the k-neighbor of node j, $0 \leq k < \delta_j$); at this point, node j checks to see: (1) if the master node is the predecessor of node j with respect to the master node (i.e., $t_j^{(k+1) \bmod \delta_j} = t_j^k$ based on Lemma 5), it then adjusts its $orient_j$ bit to k; otherwise (2) it does not change its $orient_j$ bit. Then, node j passes the token to its next neighbor (using the neighbor ordering). The process continues until the unique token in the system comes back to the 0-neighbor of the master node in $2n - 1$ steps and all slave nodes have adjusted their *orient* bits in a consistent way since the token moves along the directed Eulerian cycle. □

7 Conclusion

In this paper, we propose a randomized self-stabilizing master-slave token circulation algorithm for an undirected ring of arbitrary size. The expected stabilization time is $O(n \log n)$ steps, and the space requirement at each node is 4 bits for any undirected ring of size n. The proposed token circulation algorithm is then extended to achieve ring orientation. Disregarding the time for stabilization, the ring orientation is completed in at most $3n-1$ steps with 1 bit additional storage at each node. The proposed algorithms do not need to know the size of the ring, and thus tolerate dynamic additions and removals of the nodes as long as the ring topology is preserved. We also showed that the token circulation algorithm also works for undirected unicyclic graph of arbitrary size with similar expected stabilization time and space requirement at each node. We developed an orientation algorithm for an undirected unicyclic graph that completes in at most $4n-1$ steps with 1 bit of additional storage at each node.

Acknoledgement

The work was supported in part by NSF awards # CCF-0832582, # DBI-0960586, and # DBI-0960443.

References

[1] Y. Afek and G.M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7:27–34, 1993.

[2] G. Alari and A. K. Datta. Almost two-state self-stabilizing algorithm for token rings. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, pages 52–59, 1996.

[3] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 199–208, 1999.

[4] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Distributed Computing*, 20:75–93, 2007.

[5] J.E. Burns and J. Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11:330–344, 1989.

[6] S. Cantarell, A.K. Datta, F. Petit, and V. Villain. Token based group mutual exclusion for asynchronous rings. In *21st International Conference on Distributed Computing Systems*, pages 691–694, 2001.

[7] A. K. Datta, M. Gradinariu, and S. Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium*, pages 465–470, 2000.

[8] J. Deng, Y. S. Han, W. B. Heinzelman, and P. K. Varshney. Scheduling sleeping nodes in high density cluster-based sensor networks. In *ACM/Kluwer Mobile Networks and Applications (MONET) Special Issue on Energy Constraints and Lifetime Performance in Wireless Sensor Networks*, pages 825–835, 2004.

[9] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[10] B. Eisenberg. On the expectation of the maximum of IID geometric random variables. *Statist. Probab. Lett.*, 78(2):135–143, 2008.

[11] M. Flatebo and A. K. Datta. Two-state self-stabilizing algorithms for token rings. *IEEE Transactions on Software Engineering*, pages 500–504, 1994.

[12] W. Goddard and P.K. Srimani. Self-stabilizing master-slave token circulation and efficient size-computation in a unidirectional ring of arbitrary size. *International Journal of Foundations of Computer Science*, 23:763–777, 2012.

[13] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[14] Lisa Higham and Colette Johnen. Relationships between communication models in networks using atomic registers. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 9–pp. IEEE, 2006.

[15] J. H. Hoepman. Uniform deterministic self-stabilizing ring-orientation on odd-length rings. In *8th Int. Workshop on Distributed Algorithms*, page 265279, 1994.

[16] J. H. Hoepman. Self-stabilizing ring orientation using constant space. *Information and Computation*, 144:18–39, 1998.

[17] H. James Hoover, Piotr Rudnicki, and C Fl. The uniform self-stabilizing orientation of unicyclic networks. Technical report, University of Alberta, 1991.

[18] A. Israeli and M Jalfon. Self-stabilizing ring orientation. *Lecture Notes in Computer Science*, 486:1–14, 1991.

[19] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *21st Symposium on Reliable Distributed Systems*, pages 80–89, 2002.

[20] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parallel and Distributed Systems*, pages 154–163, 1997.

[21] K.C. Lee, C.H. Tzeng, and S.T. Huang. A space-efficient self-stabilizing algorithm for measuring the size of ring networks. *Information Processing Letters*, 94:125–130, 2005.

[22] N. Umemoto and H. Kakugawa. A self-stabilizing ring orientation algorithm with a smaller number of processor states. *IEEE Transactions on Parallel and Distributed Systems*, 9:579–584, 1998.