

### 3.3.2 Second $O(n)$ Algorithm: A Uniform Algorithm for Synchronous Rings

This algorithm is:

- uniform,
- message-bounded,
- not time-bounded (it is time-bounded by size of id, not by number of processors),
- no synchronized starts required.

#### Basic idea of the algorithm:

- Processors start either
  - spontaneously, or
  - on receiving a message.
- Messages originating at different processors are forwarded to the left at different rates.
- A processor  $p_i$  that wakes up spontaneously sends  $ID_i$  message. A processor  $p_i$  that wakes up on receiving a message only forwards IDs; it does not send its own  $ID_i$ .
- Suppose  $p_i$  sends message  $ID_i$ :
  - if message is received by a sleeping processor, it is forwarded by that processor in the next round (without delay).
  - if message reaches an awake processor, it is held for  $2^{ID_i}-1$  rounds and then forwarded, or swallowed (swallowed if  $ID_i > \text{min. IDs seen so far}$ ).
- Each processor keeps track of min. ID seen so far. A message with  $ID > \text{min. ID}$  is swallowed. The min. ID is forwarded. (Note that a message being held may be swallowed later)

The formal description of the algorithm is as follows. Note that each processor has three local variables: *min.ID* — the minimum ID seen so far, *counter* — length of time to hold *wait\_msg*, and *wait\_msg* — the message it is waiting to forward.

**Algorithm:** on processor  $p_i$

1. When  $p_i$  wakes up spontaneously,
  - it sets  $min.ID = ID_i$ ,
  - it then sends  $ID_i$  to left.
2. When  $p_i$  wakes up on receiving  $ID_j$ ,
  - it sets  $min.ID = ID_j$ ,
  - and sends  $ID_j$  to left.
3. When processor  $p_i$  receives  $ID_j$  when awake,
  - if  $ID_j = ID_i$ , then it is leader (since it receives back its own message) and sends  $leader_i$  message to the left and terminates.
  - if  $ID_j > min.ID$ , it swallows the message.
  - if  $ID_j < min.ID$ , then it sets  $min.ID = ID_j$ , sets  $counter = 2^{ID_j} - 1$  and sets  $wait\_msg = ID_j$ .
4. When a processor  $p_i$  is awake and  $counter$  is set, it decrements  $counter$  after every round. When  $counter$  reaches 0, it sends  $wait\_msg$  to left and resets  $wait\_msg$  to null.
5. When a processor  $p_i$  receives  $leader_j$ , it decides to be non-leader, forwards  $leader_j$  message and terminates.

It is clear that leader is the processor with minimum ID among those that wake up spontaneously. Note that a message is either forwarded on the next round, or waits for  $counter$ . Once a message is slowed down (is not forwarded on the next round), it won't get faster again. (Proof will be given as assignment).

We now define two kinds of messages.

**Definition 3.4.** A **fast stage** message is a message that is sent out in step 1 or 2 in the above algorithm. These messages are forwarded at rate of 1 edge per round.

A **delay stage** message is a message that is sent in step 4 of the algorithm. These messages are forwarded at rate of 1 edge every  $2^{ID_i}$  rounds. Note that messages for a processor with ID 0 are counted as delay stage messages by definition even though they are forwarded at the rate 1 edge per round.

**Definition 3.5.** A processor is **participating** if it wakes up spontaneously. Only a participating processor can become a leader.

**Lemma 3.8.** Assume that  $p_i$  is processor with minimum ID among participating processors, then:

1.  $p_i$  sends  $leader_i$  message.

2.  $\forall j, j \neq i, p_j$  does not send  $leader_j$  message.

This correctness proof is similar to the correctness proof for the earlier algorithms studied. Note that it does not depend on the synchrony of the system.

*Proof.* We first prove statement 2 by contradiction. Suppose  $p_j$  sends a  $leader_j$  message.  $p_j$  must therefore receive its  $ID_j$  back, i.e.  $ID_j$  has been forwarded around the ring. This implies that  $p_i$  forwarded  $ID_j$  message. This contradicts the algorithm since processor  $p_i$  is the processor with min. ID among participating processors, and therefore, message  $ID_j$  will get swallowed at  $p_i$ .

Now, we prove statement 1 by contradiction as well. Suppose, for contradiction,  $p_i$  did not send  $leader_i$  message. Then, by statement 2 proven above, no processor sent  $leader$  message. Therefore, no processor can terminate since each processor terminates either after sending or receiving a  $leader$  message. Now since  $p_i$  does not terminate, the only reason it does not send a  $leader_i$  message must be because  $ID_i$  did not make it around the ring, and so it must have been stopped at some processor  $p_j$ . This could happen only if:

- $p_j$  has terminated before receiving  $ID_i$ .

This, however, contradicts the above statement that no processor can terminate.

- $p_j$  swallowed  $ID_i$  message.

This can only happen if  $p_j$  has  $min.ID$  lower than  $ID_i$ . This contradicts our assumption that  $p_i$  is the processor with minimum ID among the participating processors.

□

### 3.3.3 Analysis on Message Complexity

**Lemma 3.9.** *Let  $p_i$  be the elected leader. No  $ID_j$  message is forwarded after  $ID_i$  returns to  $p_i$ .*

*Proof.* When  $ID_i$  returns to  $p_i$ , all processors have already forwarded  $ID_i$  and their  $minID$  is set to  $ID_i$ . Therefore, they will not forward any other ID larger than  $ID_i$ . □

We now define 3 categories of messages.

**Category I:** Fast stage messages.

**Category II:** Delay stage messages that are sent on rounds before the round at which leader's ID message is in delay stage.

**Category III:** Delay stage messages that are sent on and after the round where ID message of leader gets to delay stage.

**Lemma 3.10.** *Total no. of messages in Category I is  $n$ .*

*Proof.* By definition of fast stage messages, whenever a processor wakes up, it sends a Category I message (either its own ID message or other processor's ID message). Since there are  $n$  processors, there will be  $n$  Category I messages.  $\square$

Let  $r$  be the first round in which some processor is awake, and let  $p_i$  be some processor that awakes in round  $r$  (so  $p_i$  is a processor that wakes up spontaneously).

**Lemma 3.11.** *No processor terminates before round  $r+n$ .*

*Proof.* A processor terminates only after it sends *leader* message to its left. Let  $p_j$  be leader. So, first processor to terminate is  $p_j$ , after sending *leader<sub>j</sub>* message. Since  $p_j$  sends *leader<sub>j</sub>* message after it receives *ID<sub>j</sub>*, this takes at least  $n$  rounds. Since  $p_j$  sends its *ID<sub>j</sub>* message in round  $r' \geq r$ , it receives *ID<sub>j</sub>* message in no earlier than  $r'+n \geq r+n$ . Thus, no processor terminates before  $r+n$ .  $\square$

**Lemma 3.12.**  $\forall k < n$ , if  $p_j$  is at distance  $k$  to the left of  $p_i$ , then a Category I message is received by  $p_j$  no later than round  $r+k$ .

*By Induction on  $k$ .* From Lemma 3.11, we know that no processor has terminated yet at round  $r+k$  since  $k < n$ . So all processors are either awake or sleeping at round  $r+k$ .

Let  $p_j = p_{i+k}$  where processor sequence is

$$p_i, p_{i+1}, p_{i+2}, \dots, p_{i+k}$$

**Basis case:**

If  $k=1$ ,  $p_i$  wakes up at round  $r$  and sends a Category I message to  $p_j$  immediately, which is received by  $p_j$  at round  $r+1$ .

**Inductive hypothesis:**

$p_{i+k-1}$  receives Category I message no later than  $r+k-1$ .

**Inductive step:**

Since  $p_{i+k-1}$  wakes up on or before receiving a Category I message, it follows that  $p_{i+k-1}$  wakes up no later than round  $r+k-1$ . Since it sends a Category I message as soon as it wakes up, it will send such message no later than round  $r+k-1$ . Thus  $p_{i+k}=p_j$  receives a Category I message no later than round  $r+k$ .  $\square$