The Bit Complexity of the Predecessor Problem

Y. Afek, M. Cohen, E. Haalman

Computer Science Department
Tel-Aviv University
Israel 69978

Abstract

Tight bound of $2n - 1$ bits on the communication complexity of "The Predecessor" problem in a synchronous ring (previously known as "The last in a synchronous ring" problem) is presented.

Key Words: Distributed Computing, Unidirectional, Anonymous Ring.

1 Introduction

In [6] the following question, which we call the "predecessor problem", in distributed computing on a unidirectional ring was raised: In a unidirectional ring of finite automata, design an algorithm by which a designated node, v, distinguishes the node preceding it from all other nodes. A solution to the predecessor problem plays a key roll in distributed algorithms for uni-directional networks. In particular it serves as a basic building block in algorithms for Depth First Search [1], Leader Election [6], Firing Squad Synchronization [4, 8], and recently for self-stabilization in Unidirectional rings [7] and in SDH/SONET ring synchronization [2].

In [6] a tight bound of $\Theta(n \log n)$ bits of communication on any asynchronous solution to the problem was proved, where n is the total number of nodes in the ring. In [1] an upper bound of $3n-1$ bits and messages of communication is presented for the synchronous case. Here we prove that for the synchronous case, when n is unknown (i.e., no node knows n) and each node has at least $\log n$ bits of memory, the upper and lower bounds on the bit and message communication complexities of the problem are $2n-1$.

Note that the lower bound holds even if the nodes have unique identities and unlimited space, given that no node knows the identity of any of its neighbors. The time complexity of our algorithm is at most $5n$ rounds.

Although we provide here only a constant improvement over the known bounds on the problem, the results are still interesting because: First, we provide absolutely tight upper and lower bounds, second, the problem was identified in [4, 7, 8, 6, 1] as a basic building block for computing in a unidirectional network, and finally, this is a simple and elegant pedagogical example of the differences between the synchronous and asynchronous modes of communication.

We assume the model of synchronous unidirectional communication networks as in [1, 6] which follows the standard model [5, 3]. We consider a unidirectional ring of n processors. In the upper bounds we assume that the nodes are identical (no unique ids) and each node has $O(\log n)$ bits of memory. When counting the bit communication complexity we consider only the bits which convey any information. In particular, our algorithm sends only empty messages, i.e., a signal that a message is present. If other algorithms run in the ring at the same time as our then the messages of the different algorithms should be distinguishable and may be piggybacked. In measuring the complexity of our algorithm messages are assumed to have been already distinguished. Other traffic on the ring may not delay the sending of messages of the algorithm. The lower bound is on the number of messages regardless of their size.

In this paper nodes are assumed not to know n, the total number of nodes in the network. This is the case in many real life rings where nodes are added and removed dynamically from the ring (e.g., IEEE Token Ring standard). However the algorithm assumes that the number of nodes does not change during its execution.

We consider algorithms that terminate in finite time and when the algorithm terminates every node knows whether it is the last on the ring or not. The upper bound is a deterministic algorithm while the lower bound holds also for randomized algorithms, as long as the algorithm always produces a correct answer in finite time.

2 The algorithm

Let the *designated* node be the pre-distinguished node that starts the algorithm and whose predecessor needs to be identified. The predecessor of the designated node that needs to be identified is called the *last* node. Let p_i be the i^{th} processor in the ring, where the designated node is p_0, p_1 is its successor and so on). This numbering is external neither known nor available to the nodes themselves.

The algorithm has two phases (see Figure 1). By the end of the first phase the designated node has computed n, the number of nodes in the ring. In the second phase each node decides whether it is the last on the ring, or not. In the first phase the nodes simply forward around the ring a message. The designated node counts the time it takes the message to travel around, which is n. In the second phase another message is sent around the ring. However, this time the message is delayed one round at each node, aside from the designated node. Thus, if d_i is the number of rounds from the reception of the first phase message at p_i until the reception of the second phase message at p_i, then $d_i = d_{i-1} + 1$ for $i = 2, \ldots, n-1$. The trick is that the designated node delays the starting of the second phase $T(n)$ rounds, such that d_i is a power of two *only* at the last node (i.e., only d_{n-1} is a power of two). (Specifically, if N is the smallest power of 2 that is greater or equal to n then d_{n-1} is $2N$). For example, if $n = 10$ then the algorithm would ensure that d_9 of the last node would be 32. In that case d_1 is 24, d_2 is $25, \ldots,$ and d_9 is 32. $T(n)$ that does the trick is

$$T(n) = 2N - 2n + 2$$

where N is the smallest power of 2 that is greater or equal to n.

2.1 Correctness

Let d_i be the value of *delay* (see Figure 1) at processor p_i when it receives the second phase message.

Lemma 1 $d_i = 2N - n + i + 1$ *for* $i = 1, \ldots, n-1$.

The algorithm :

Initially:

$delay := 0$

The code for the designated node:

```
Send a message on the outgoing link.
On each round do
                delay := delay + 1
Until a message from last is received.
                                                    /* begin phase 2 */
N = min_{k≥1} {2^k : 2^k ≥ delay}                  /* delay is equal to n */
wait T(n) = 2 · N − 2 · delay + 2  rounds.
send a message on the outgoing link.
```

The code for Processor p_i, $i \geq 1$:

```
Upon receiving the first message:
Send a message on the outgoing link.
On each round do
                delay := delay + 1
until received the second message from its predecessor.
                                                    /* begin phase 2 */
if delay is a power of 2
        I am the last
else
        wait 1 round.
        send a message on the outgoing link.
        I am not the last
fi
```

Figure 1: The Algorithm.

Proof:
By induction on i :
$i = 1$: Clearly, $d_0 = n$ because every processor in the first phase forwards the message immediately. The designated node waits $2N - 2n + 2$ rounds before sending the second phase message. Thus $d_1 = 2N - 2n + 2 + n = 2N - n + 1 + 1$.
$\forall i,\ 1 \leq i < n - 1\ \ d_{i+1} = d_i + 1 = 2N - n + i + 2 = 2N - n + (i + 1) + 1$ since every processor excluding the designated delays the second message one round. ∎

Corollary 1 *The value of d_i is a power of two if and only if p_i is the last processor in the ring.*

Proof:
By lemma 1 $d_{n-1} = 2N - n + (n - 1) + 1 = 2N$, therefor it is a power of 2. Since $d_{i+1} = d_i + 1$,

$$d_{n-2} = 2N - 1,\ d_{n-3} = 2N - 2, \ldots, d_1 = 2N - n + 2 > N$$

Hence $\forall i,\ 1 \leq i < n - 2\ \ d_i$ is not a power of 2. ∎

3 The lower bound

Theorem 1 *The message complexity of any synchronous algorithm for "The Predecessor" problem is $2n - 1$ messages.*

Proof:
Let A be an arbitrary algorithm for "The Predecessor" problem. Let E_{n_1} and E_{n_2} be two executions of A on an n_1 nodes and on an n_2 nodes synchronous rings, such that $n_2 < n_1$ and p_i for $i = 0, \ldots, n_2 - 1$, is in an identical initial state in both rings.

Lemma 1 *The projections of any two equal length prefixes of E_{n_1} and E_{n_2} on processors p_0 to p_{n_2-1}, such that in neither the designated node receives a message, are identical.*

Proof:
By induction on the prefixes length. Both executions start by the designated node (p_0) sending the same message, since the ring size is unknown and the

initial state of p_0 is the same in both executions. The induction step is immediate, since by the hypothesis in both processors p_0 to p_{n_2-1} are in the same state. (If A is a randomized algorithm then the same proof holds by assuming that in both prefixes the sequence of coin flips is the same). ∎

Lemma 2 *The first message process p_i, $i = 1, \ldots, n_2 - 1$ receives in E_{n_1} and in E_{n_2} is the same.*

Proof:
Follows immediately from Lemma 1 because the designated node does not receive a message until node p_i receives its first message. ∎

Lemma 3 *At least one message is received by every processor in E_{n_2}.*

Proof:
Before receiving the first message the processor is asleep. Thus the Lemma is immediate for processors 1 through $n_2 - 1$. Suppose no message is received by the designated node in E_{n_2}, then p_{n_2} would not receive a message in E_{n_1} and the algorithm would fail in E_{n_1}, a contradiction. ∎

Thus the message complexity of A is at least n.

Let K_n be the set of nodes that receive no more messages in E_n after the designated node had received a message for the first time.

Clearly, every node must receive at least one message before the designated node receives any message, thus any node that receives only one message in E_n is in K_n.

Lemma 4 $n_2 < n_1 \Longrightarrow K_{n_1} \cap K_{n_2} = \emptyset$

Proof:
Suppose to the contrary that there is an i such that $p_i \in K_{n_1} \cap K_{n_2}$. By Lemma 2 the first message p_i receives in E_{n_1} and in E_{n_2} is the same. Then, in any round processor p_i is in exactly the same state in E_{n_1} and in E_{n_2}. Therefore, inductively processors $p_{i+1} \ldots p_{n_2-1}$ receive the same series of messages in the same rounds, thus processor $n_2 - 1$ will decide the same decision about being last in both executions, a contradiction. ∎

Lemma 5 *Either (C1.) there exists an infinite set of naturals N s.t., $\forall n \in N$, $K_n = \emptyset$, or (C2.) there exists n_0 s.t., $n > n_0 \Longrightarrow |K_n| \leq 1$.*

Proof:
Suppose the lemma is false. Thus neither (C1) nor (C2) hold. Since (C1) does not hold there exists n_0 s.t., $n > n_0 \Longrightarrow |K_n| \geq 1$. Since (C2) does not hold, there is an infinite set M s.t., $\forall n \in M$ $|K_n| \geq 2$. Choose an $m \in M$ s.t., m is larger than at least n_0 elements of M. $|\bigcup_{n=1}^{m} K_n| \geq m$ since at least n_0 of $K_1 \ldots K_m$ have size ≥ 2, at least $m - n_0$ of $K_1 \ldots K_m$ are of size ≥ 1 and they are disjoint sets (by Lemma 4). But $K_n \subseteq \{1, 2, \ldots, n-1\}$ therefore $|\bigcup_{n=1}^{m} K_n| \leq |\{1, 2, \ldots, m-1\}| = m - 1$ contradiction. ∎

If C1 holds then the lower bound is proved since at least 2 messages are transferred through every edge except the last one on which at least 1 message is transferred.

Otherwise, by Lemma 5, there exists n_0 s.t., $n > n_0 \Longrightarrow |K_n| \leq 1$ thus in any case the message complexity of A is at least $2n - 2$. To prove Theorem 1 assume contrary to the theorem that an infinite set of naturals N s.t., $\forall n \in N$ at least $2n - 1$ messages are sent in E_n does not exist. Thus there exists n_0 s.t., for each $n > n_0$:

1. $|K_n| = 1$, and

2. Every node receives exactly 2 messages in E_n except the designated node and exactly one other node (denoted p_{l_n}) both of which receive exactly one message in E_n (i.e., $K_n = \{p_{l_n}\}$).

From now on we analyze that possibility to reach a contradiction.

Choose an $n_1 > n_0$. Execute the algorithm with a ring of size n_1. After a finite number of rounds d, the algorithm terminates. Now execute the algorithm with $n_2 = n_1 + d + 1$ nodes. Node $p_{l_{n_1}}$ must have received the second message at most d rounds after it received it's first one. Otherwise processors $p_{l_{n_1}}, p_{l_{n_1}+1}, \ldots, p_{n_1-1}$ would have received the same messages and in the same time gaps in both executions. Thus processor p_{n_1-1} would decide the same decision about being last in the size n_2 ring. Therefore, $p_{l_{n_1}}$ must have received it's second message before the designated node received it's first in E_{n_2}. But by point 2 above, $p_{l_{n_1}}$ received only 2 messages and therefore it received its last message before the designated node received its first one, thus $p_{l_{n_1}} \in K_{n_2}$, Contradicting Lemma 4. Hence the assumption is false. ∎

Acknowledgments: We would like to thank Moty Ricklin for very helpful remarks that have simplified the algorithm. We would like also to thank the referee for very helpful and thorough comments.

References

[1] Y. Afek and E. Gafni. Distributed algorithms for unidirectional networks. *Siam J. on Computing*, 23(6), December 1994.

[2] Y. Afek and T. Lev. Distributed synchronization protocols for SDH networks. Submitted for publication, November 1995.

[3] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, October 1985.

[4] S. Even, A. Litman, and P. Winkler. Computing with snakes in directed networks of automata. In *Proc. of the 31st IEEE Ann. Symp. on Foundation of Computer Science*, pages 740–745, October 1990.

[5] G. N. Frederickson and N. A. Lynch. Electing a leader in a synchronous ring. *Journal of The ACM*, 34(1):98–115, January 1987.

[6] E. Gafni and Y. Afek. Election and traversal in unidirectional networks. In *Proc. of the 3rd Ann. ACM Symp. on Principles of Distributed Computing*, August 1984.

[7] A. Mayer, R. Ostrovsky, and M. Yung. General distributed self-stabilizing algorithms for unidirectional rings. In *Proc. of 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, January 1996.

[8] Nishitani and Honda. The firing squad synchronization problem for graphs. *Theoretical Computer Science*, 14:39–61, 1981.