Information Processing Letters 22 (1986) 185–187
North-Holland 17 April 1986

ELECTION IN A COMPLETE NETWORK WITH A SENSE OF DIRECTION *

Michael C. LOUI, Teresa A. MATSUSHITA ** and Douglas B. WEST

Coordinated Science Laboratory, College of Engineering, University of Illinois at Urbana-Champaign, 1101 West Springfield Avenue, Urbana, IL 61801-3082, U.S.A.

Communicated by M.A. Harrison
Received 25 February 1985
Revised 17 May 1985

Keywords: Election, distributed algorithm, message complexity, communication complexity

1. Introduction

Consider a complete network of N asynchronous processors. Every pair of processors is joined by a bidirectional link. Each processor has a unique integer identifier (*id*). The *Election Problem* is to identify the processor with the largest *id*.

Fix a Hamiltonian cycle that includes all the processors. The network has a *sense of direction* [6] if at each processor the label on each link gives the distance along this Hamiltonian cycle to the processor at the other end of the link. In particular, if processor x is at distance d from processor y, then y is at distance N − d from x.

We present an algorithm that uses O(N) messages to solve the Election Problem in a complete network with a sense of direction. In contrast, if at each processor the links are unlabeled, then the network has no sense of direction. In this latter case, Ω(N log N) messages are required to solve the Election Problem [2], and O(N log N) messages suffice [1,2,4].

* Supported by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under Contract N000-14-84-C-0149, by the National Science Foundation under Grant MCS82-17445, and by the Eastman Kodak Company. A preliminary version of this paper was presented at the 1985 Conference on Information Sciences and Systems, Baltimore, MD, U.S.A.

** Current affiliation: AT&T Information Systems, Denver, CO, U.S.A.

2. The algorithm

Every processor executes the same algorithm, which resembles Peterson's algorithm [3]. After we have informally described our algorithm, we specify our algorithm formally.

Initially, all processors are *active*, and eventually all processors except the processor with the largest *id* become *passive*. An active processor becomes passive when it receives a message that contains an *id* larger than its own *id*. Conversely, when an active processor receives a message that contains an *id* j smaller than its own *id*, it sends a message with its own *id* directly to the processor x whose *id* is j; this message ensures that x becomes passive. Every message contains the distance from the processor whose *id* is in the message to the processor that receives the message. Processors use the distance information to determine which links they should use.

Every processor has its own *id* and local variables D, E, and *Newid*. Procedure SEND(d; e, j) sends a message (e, j) along link d to the processor at distance d. Procedure RECEIVE(e, j) waits until a message (e, j) arrives.

```
        D := N − 1;
active: repeat SEND(N − D; N − D, id);
               RECEIVE(D, Newid)
        until  Newid ⩾ id;
        if Newid = id then Announce "elected"
               else
```

passive: RECEIVE(E, *Newid*);
SEND(N − D ; N − (D + E), *Newid*)
end

Call a processor *active* if it sends a message at label "active", *passive* if it reaches label "passive". Fig. 1 presents an example of an execution of the algorithm. Each node represents a processor, and the number in the node is the *id* of the processor. Each arc represents the transmission of a message, and the number on the arc is the *id* in the message. The *id* of a passive processor is replaced by "P".

Initially, every processor is active, and every processor sends its *id* to the processor at distance 1 from it. In general, when a processor y receives a message (e, j), j is the *id* of the processor x such that y is at distance e from x. If y is active and j is less than the *id* of y, then y sends a message with its own *id* directly to x, which is at distance N − e from y, to ensure that x becomes passive. If y is active and j is greater than the *id* of y, then y becomes passive. If y is already passive, then y sends a message (e′, j) with the same *id* j to a processor z at distance N − D from y, and z is at distance e′ = N − (D + e) from x; the *id* of processor z caused y to become passive at some previous time. Notice that the message sent by a passive processor y tells active processors how to bypass y; after y sends this one message, no processor will later send y a message.

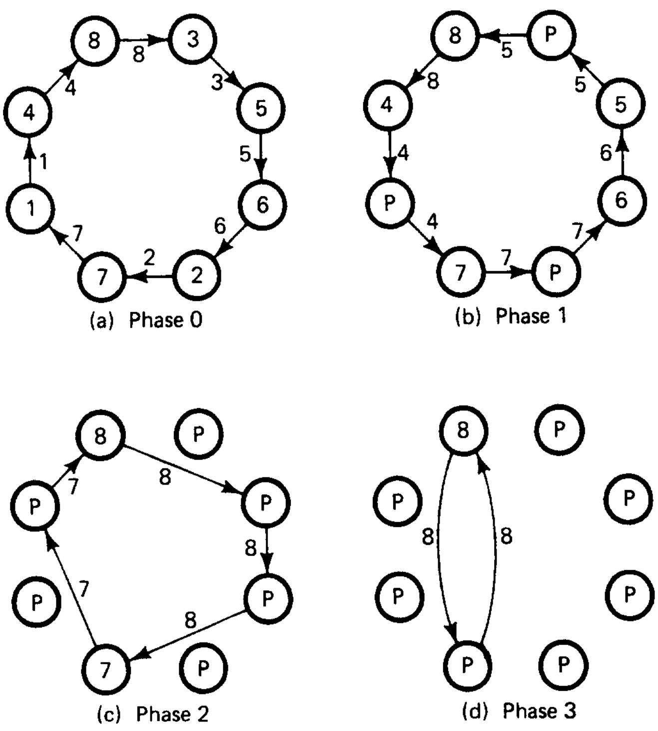

Fig. 1.

3. Analysis

We divide the computation into *phases*. Without loss of generality we shall assume that all active processors send their messages simultaneously at the beginning of a phase. During the phase, some passive processors may transmit messages. Phase p ends when all active processors receive the messages sent during phase p. Number the phases so that phase 0 is the first phase, and phase p + 1 begins when phase p ends. Let n_p be the number of processors active at the beginning of phase p. By definition,

$$n_0 = N. \tag{1}$$

During the execution of the algorithm a total of N − 1 messages are sent by passive processors, and

$$\sum_{p \geqslant 0} n_p$$

messages are sent by active processors. We shall bound this sum.

A processor remains active at the end of phase p − 1 only if a neighboring active processor became passive during phase p − 2. Thus,

$$n_p \leqslant n_{p-2} - n_{p-1} \quad \text{for } p \geqslant 2. \tag{2}$$

Let $\phi = \frac{1}{2}(1 + \sqrt{5})$. We shall prove that

$$\sum_{p \geqslant r} n_p < \phi^2 n_r \tag{3}$$

by induction on the number of terms in the sum. If this sum has one or two terms, then since $n_{r+1} \leqslant n_r$,

$$n_r < \phi^2 n_r,$$

$$n_r + n_{r+1} \leqslant n_r + n_r < \phi^2 n_r,$$

and (3) holds. Assume inductively that (3) holds for r + 1 and r + 2. Then

$$\sum_{p \geq r} n_p = n_r + \sum_{p \geq r+1} n_p < n_r + \phi^2 n_{r+1}, \tag{4}$$

$$\sum_{p \geq r} n_p = n_r + n_{r+1} + \sum_{p \geq r+2} n_p$$

$$< n_r + n_{r+1} + \phi^2 n_{r+1}. \tag{5}$$

Applying (2) to (5) yields

$$\sum_{p \geq r} n_p < n_r + n_{r+1} + \phi^2 (n_r - n_{r+1})$$

$$= (1 + \phi^2) n_r + (1 - \phi^2) n_{r+1}. \tag{6}$$

The upper bounds (4) and (6) are equal when

$$n_r + \phi^2 n_{r+1} = (1 + \phi^2) n_r + (1 - \phi^2) n_{r+1},$$

$$(2\phi^2 - 1) n_{r+1} = \phi^3 n_{r+1} = \phi^2 n_r,$$

$$n_{r+1} = n_r / \phi.$$

Consequently,

$$\sum_{p \geq r} n_p < \max_{n_{r+1}} \min\{n_r + \phi^2 n_{r+1},$$

$$(1 + \phi^2) n_r + (1 - \phi^2) n_{r+1}\}$$

$$= (1 + \phi) n_r = \phi^2 n_r,$$

as claimed in (3).

Ergo, by (1) and (3), the number of messages used by the algorithm is

$$N - 1 + \sum_{p \geq 0} n_p < N - 1 + \phi^2 N < 3.62N.$$

To obtain an upper bound on the total message delay, observe that the message delay in phase p is at most 1 plus the number of processors that became passive at the end of phase p − 1. By (2), there are $\log_\phi N + O(1)$ phases [3]. Furthermore, N − 1 processors become passive during the execution of the algorithm. Thus the total message delay is at most

$$N + \log_\phi N + O(1).$$

Acknowledgment

Our thanks are due to an unknown referee who noticed that our original upper bound on the number of messages could be improved to 3.62N.

References

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

[2] E. Korach, S. Moran and S. Zaks, Tight lower and upper bounds for some distributed algorithms for a complete network of processors, Proc. 3rd Ann. ACM Symp. on Principles of Distributed Computing (1984) 199–207.

[3] G.L. Peterson, An $O(n \log n)$ unidirectional algorithm for the circular extrema problem, ACM Trans. Programming Language Systems 4 (1982) 758–762.

[4] G.L. Peterson, Efficient algorithms for elections in meshes and complete networks, Tech. Rept. TR-140. Dept. of Computer Science, Univ. of Rochester, 1984.

[5] J. Sack, N. Santoro and J. Urrutia, $O(n)$ election algorithms in complete graphs with sense of direction, Tech. Rept. SCS-TR-49, Carleton Univ., 1984.

[6] N. Santoro, Sense of direction, topological awareness, and communication complexity, SIGACT News 16 (2) (1984) 50–56.