Bit-Optimal Election in Synchronous Rings

Hans L. Bodlaender and Gerard Tel

RUU–CS–89–2
January 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

Bit-Optimal Election in Synchronous Rings

Hans L. Bodlaender and Gerard Tel

Technical Report RUU–CS–89–2
January 1989

Department of Computer Science
University of Utrecht
P.O. Box 80.089, 3508 TB Utrecht
The Netherlands.

Bit-Optimal Election in Synchronous Rings

Hans L. Bodlaender and Gerard Tel

Department of Computer Science, University of Utrecht, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands.

Abstract: An election algorithm is presented for synchronous rings with unknown size. The bit complexity of the proposed algorithm is linear in the number of processes. The time complexity is polynomial in the number of processes, but exponential in the smallest identity of any process.

1 Introduction

In recent years considerable attention has been given to the problem of electing a leader in a network of processes. The purpose of an election algorithm is to bring the network in a state where one and only one process has the status *elected*, while all other processes have the status *defeated*. To make a deterministic solution possible we assume that each process has a unique identification number, which is a positive integer. Also we assume that the topology of the network is a ring and we denote the number of processes by N.

A distinction is usually made between *synchronous* and *asynchronous* networks. In an asynchronous system the transmission delay of a message is finite, but unpredictable and not a priori bounded. In a synchronous network the processes act in rounds, or phases, and a message sent in one round is guaranteed to be received in the next round.

For asynchronous rings election algorithms exist that use $O(N \log N)$ messages and this is optimal, regardless whether N is known to the processes or not [DKR82, Pe82, PKR84]. For synchronous rings we have to distinguish between the case that N is known to the processes and the case that N is not known. If the processes know N, there exists an algorithm using $O(N)$ messages of $O(1)$ bits, thus achieving the optimal complexity of $O(N)$ bits [SR85]. For the case where the processes do not know N there exists algorithms that use $O(N)$ messages, but $\Omega(N \log N)$ bits [Vi85, FL87]. The question whether an algorithm exists using $O(N)$ bits remained open [OS89]. In this paper we answer this question in the affirmative by presenting an algorithm with the desired properties.

The work of the second author was supported by the Netherlands Organization for Scientific Research (NWO). The email addresses of the authors are *hansb@ruuinf.uucp* and *gerard@ruuinf.uucp*.

2 Description of the algorithm

The proposed algorithm is a combination of three known algorithmical ingredients. The first one is the election algorithm for (unidirectional) rings by Chang and Roberts [CR79]. The second one is a technique by Vitanyi [Vi85] and Frederickson and Lynch [FL87] to reduce the message complexity of election algorithms when they execute on a synchronous network. The third one is a method to transmit any message using only a constant number of bits in a synchronous network.

In the election algorithm by Chang and Roberts each process sends a token over the ring containing its own identity. A process, say with identity i, that receives a token $<j>$ acts as follows. If $j > i$ the token is purged. If $j < i$ the token is forwarded. If $j = i$ the process obtains the *elected* status and sends a special message around the ring to abort all activity in other processes and force them to obtain the *defeated* status. To establish the correctness of the algorithm, let m be the smallest identity of any process. A process different from m does not obtain the *elected* status because its token does not pass process m. m obtains the *elected* status because all processes forward its token. This algorithm uses $\Omega(N^2)$ messages in the worst case, but only $O(N \log N)$ in the average case. In a variant, which we shall use in this paper, process i also purges a token $<j>$ if i has earlier received a token $<j'>$ with $j' < j$.

As the winning token makes only N steps, future "looser" tokens contribute most to the message complexity. Vitanyi [Vi85] proposed to delay a token $<j>$ by $f(j)$ time units (rounds) in every process, where f is a suitably chosen function. The algorithm now completes in $N.f(m)+O(N)$ rounds, and the total number of token passes for loosers is bounded by $\sum_{i>m} \lceil \frac{N.f(m)+O(N)}{f(i)} \rceil$. Choosing $f(i)$ equal to e.g. 2^i yields an $O(N)$ bound on the number of messages.

In a synchronous system any message, regardless of its contents, can be transmitted using $O(1)$ bits by coding its contents in time. Assume the contents of a message is a positive integer. To send a message, say M, in round x, a process sends <Open> in round x and <Close> in round $x+M$. If a process receives <Open> in round s and <Close> in round t, it accepts the message $t-s$. Note that although coding in time assumes a synchronous underlying network, the system is no longer synchronous when it is used because the transmission time of a message is no longer bounded.

We shall demonstrate in the following that it is possible to combine these three ingredients into one algorithm. First we describe in detail the protocols for the communication of tokens. To transmit a token $<j>$, a process sends an <Open> message and j rounds later a <Close> message. If, however, the transmission of a new message is required in the meantime, the current transmission is implicitly aborted by sending a new <Open> message. The sending protocol is as follows (initially *SendTimer* $= -1$).

```
procedure transmit (M):
    begin send <Open> ; SendTimer := M end
```

```
At the beginning of each round:
    SendTimer := SendTimer − 1
```

```
At the end of each round:
    if SendTimer = 0 then send <Close>
```

Thus the executions of the *transmit* procedure result in a message pattern $(\text{<Open>}^{+}\,\text{<Close>})^{*}$ to be sent over the link. The receiving process accepts a message each time it receives a <Close> message. The receiving protocol is as follows.

```
Upon receipt of <Open>:
    RecTimer := 0
```

```
Upon receipt of <Close>:
    accept(RecTimer)
```

```
At the beginning of each round:
    RecTimer := RecTimer + 1
```

Using these protocols, a message M is accepted $M+1$ rounds after it is transmitted (unless the transmission is aborted for a new one).

Each process in the ring implements the sending protocol, the receiving protocol, and the actual election procedure. We shall now describe the latter. The procedure (for process i) includes the spontaneous transmission of a token $<i>$. Transmission is started at the latest when an <Open> message is received, and it is done at most once in every process. The variable min_i initially has the value i. On acceptance of a token $<j>$, j is compared to min_i and i. If $j > min_i$, nothing is done and the token ends its existence. If $j = i$ the process obtains the *elected* status and sends a special <Stop> message around the ring. The <Stop> message aborts all activity in other processes. If $j < min_i$, min_i is set to j and, after a delay of $f(j)$ rounds, $<j>$ is transmitted. The complete election procedure is described as follows (initially $min_i = i$).

```
Spontaneously or triggered by the receipt of an <Open> message:
    transmit <i>

procedure accept (j):
    if $j < min_i$ then
        begin $min_i := j$ ; $ElecTimer := f(j)$ end
    else if $j = i$ then { $i$ is elected }
        begin send <Stop> ; status := elected end

Upon receipt of <Stop>: { $min_i$ is elected }
    if $i \neq min_i$ then
        begin send <Stop> ; status := defeated end
    abort all activity

At the beginning of each round:
    $ElecTimer := ElecTimer - 1$

At the end of each round (but before the possible sending by the sending protocol):
    if $ElecTimer = 0$ then transmit <$min_i$>
```

3 Correctness of the algorithm

The correctness of the algorithm is shown in the same way as for the algorithm in [CR79]. Let m be the smallest of the identities. The token of any process other than m, if it reaches m at all, is purged by m, hence no process other than m obtains the *elected* status. A process that spontaneously starts the procedure transmits its identity and thus triggers the start of the procedure in the next process in the ring also. It follows that if any process starts the procedure, eventually m starts the procedure. The token $<m>$ is forwarded by every process, and its transmission or its delay are never aborted. It follows that eventually m accepts its own token and obtains the *elected* status.

4 Complexity analysis

The complexity of the algorithm is derived in the same way as for the algorithms in [Vi85]. Again let m be the smallest identity. At the latest in the N^{th} round of the execution of the algorithm m transmits its token for the first time. It takes $m+1$ rounds to be transmitted and is delayed for $f(m)$ rounds in every process, hence it returns to m within $N.(f(m)+m+1)$ rounds. The <Stop> message completes its tour in less than N rounds so the total running time of the algorithm is bounded by $T = 2N + N(f(m)+m+1)$ rounds.

Next a bound on the number of token transmissions is derived. $<m>$ is passed N times. A token $<j>$ is delayed for $f(j)$ rounds in each process and takes $j+1$ rounds to be transmitted, so in T rounds its transmission is initiated at most $\left\lceil \frac{T}{f(j)+j+1} \right\rceil$ times. The total number of token passes is thus bounded by $N + \sum_{j \in I} \left\lceil \frac{T}{f(j)+j+1} \right\rceil$, where I denotes the set of process identities excluding m.

The analysis is completed by supplying a suitable function f. Take $f(i) = 2^i - i - 1$ and let C denote the number of initiated token transmissions. Now

$$\begin{aligned}
C &\le N + \sum_{j \in I} \left\lceil \frac{2N + N(f(m)+m+1)}{f(j)+j+1} \right\rceil \\
&< N + N + \sum_{j \in I} \frac{2N + N(f(m)+m+1)}{f(j)+j+1} \\
&< N + N + \sum_{j > m} \frac{2N + N.2^m}{2^j} \\
&= 2N + (2N + N.2^m) \sum_{j > m} \frac{1}{2^j} \\
&= 2N + (2N + N.2^m) \frac{1}{2^m} \\
&\le 4N
\end{aligned}$$

We find that the number of <Open> and <Close> messages is bounded by $4N$. The number of <Stop> messages is n, hence the total number of messages (<Open>, <Close>, and <Stop>) is bounded by $9N$. Each of the three messages is represented in a constant number of bits, hence the bit complexity of the algorithm is $O(N)$. Under this assignment of f the time complexity of the algorithm is $O(N.2^m)$.

5 Conclusions

An election algorithm was given for synchronous rings with unknown size, using a linear number of bits. This settles an open question of [OS89].

The algorithms in [OS89] operate in global phases, in which processes gain more and more information about N and m. In our algorithm each process has its own token running around the ring. Each process acts, so to speak, individually. In this case the latter approach yields a lower complexity.

The approach can be adapted to networks with a weaker synchronicity property, such as Archimedian networks [Vi85]. Also it is possible to use the same approach to derive algorithms for topologies other than a ring.

By improving details of the algorithm or a more carefull complexity analysis the bound on the complexity can be improved, but by no more than a constant factor. We chose to keep the presentation clear and short, for our only purpose was to show that an algorithm exists with a linear bit complexity.

References:

[CR79] Chang, E., R. Roberts, *An Improved Algorithm for Decentralized Extrema Finding in Circular Arrangements of Processes,* Comm. ACM 22 (1979) 281–283.

[DKR82] Dolev, D., M. Klawe, and N. Rodeh, *An O(n log n) Unidirectional Algorithm for Extrema Finding in a Circle,* J. Algorithms 3 (1982) 245-260.

[FL87] Frederickson, G.N., and N.A. Lynch, *Electing a Leader in a Synchronous Ring,* J. ACM 34 (1987) 95-115.

[OS89] Overmars, M., N. Santoro, *Time vs Bits: An Improved Algorithm for Leader Election in Synchronous Rings,* Proceedings STACS89.

[PKR84] Pachl, J., E. Korach, and D. Rotem, *Lowerbounds for Distributed Maximum-Finding Algorithms,* J. ACM 31 (1984) 905-918.

[Pe82] Peterson, G.L., *An O(nlogn) Unidirectional Algorithm for the Circular Extrema Problem,* ACM ToPLaS 4 (1982) 758-762.

[SR85] Santoro, N., and D. Rotem, *On the Complexity of Distributed Elections in Synchronous Graphs,* in: Proc. 11th Workshop on graph-theoretic concepts in computer science, 1985, pp. 337-346.

[Vi85] Vitanyi, P.M.B., *Time-Driven Algorithms for Distributed Control,* Tech. Rep. CS-R8510, CWI, Amsterdam, The Netherlands, 1985.