Efficient leader election in complete networks

J. Villadangos, A. Córdoba, F. Fariña, M. Prieto

Dpt. Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadia
31006 - PAMPLONA (SPAIN)
{jesusv, alberto.cordoba}@unavarra.es,
{fitxi, manuel.prieto}@unavarra.es

Abstract

Leader election is a fundamental problem in distributed computing and it has been studied assuming various computation models and network topologies. This paper analyzes the algorithms for leader election in complete networks using asynchronous communication channels. We present a novel algorithm that reduces the information necessary to select a leader compared with other leader election algorithms for complete networks. In this paper, the algorithm works without sense of direction. And, it does not require to know the number of nodes in the system. Our proposal requires $O(n)$ messages and $O(n)$ time, where n is the number of nodes in the system, to elect a leader.

Key words: leader election, complete networks, distributed algorithms, I/O automata.

1. Introduction

A set of processes passing messages to communicate with each other is considered a distributed system. Usually, there is a communication layer responsible to transmit the data between processes. And this communication layer can provide different communication guarantees: FIFO message ordering, total order message reception in group communication protocols ([5] provides a review of group communication and discuss systems guarantees for message delivery).

Nowadays, operating systems provide communication services to exchange information between processes using the TCP/IP protocol stack independent of the network technology. This part of the operating system is responsible for transmit and receive messages which are delivered to the corresponding process. So, processes entrust the communication tasks to the operating system. The communication service requires only the identification of the destination in order to transmit the data. In this way, processes can communicate with each other process in the system, only if it knows the identification of the destination.

Thus, a set of processes exchanging messages using the TCP/IP protocol stack and which know each other their communication identification can be considered a distributed system where processes form a complete network.

From the point of view of control, many algorithms that are called distributed, are in fact centralized: there is a single co-ordinating process that performs certain functions on behalf of the others when it is asked for these. Such algorithms, like any centralized one, suffers from the great disadvantage that a failure of the co-ordinating process results in the failure of the algorithm as a whole. This can be overcome by the remaining processes by conducting a negotiation among themselves with the aim of electing one of them to take over the role of co-ordinator. The algorithms associated with such negotiations are called leader election algorithms.

This algorithms can be applied to select the better suited server to provide a service (i.e., to get a database register, an URL, etc.) from the set of servers belonging to a cluster or from a set of servers distributed in the Internet. In the first case, usually servers are connected together by a high-speed Local Area Network (LAN) forming a cluster. Such architectures are considered between other to provide replication services [14], and to balance servers loads [3, 9].

In the leader election problem it is assumed that there are n nodes in the network, each having a unique identity. Leader election algorithms must verify that, eventually, exactly one process outputs the decision that it is the leader, say by changing a special status component of its state to the value $leader$.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

Leader election is a fundamental problem in distributed computing. It has been studied assuming various computation models and network topologies. For an asynchronous network with an arbitrary topology, it holds a lower bound of $\Omega(e + n\ log\ n)$ messages, where e is the number of edges in the network. A protocol with this message complexity was shown in [8]. The time complexity of this protocol is $O(n\ log\ n)$. In [4], an improved protocol with time complexity $O(n)$ was proposed. Several protocols for ring topologies have also been proposed [7, 10].

In [2], it was proved a lower bound on the message complexity of leader election for synchronous networks of $\Omega(n\ log\ n)$. It also was showed that any protocol sending at most $O(n\ log\ n)$ messages must require at least $\Omega(log\ n)$ time. For synchronous networks, it was shown in [11] that $\Omega(n\ log\ n)$ messages are required for leader election. This paper also proposes a protocol which requires $O(n\ log\ n)$ messages and $O(n\ log\ n)$ time. In [1], the complexity was further improved by giving a number of protocols for asynchronous networks, each with $O(n\ log\ n)$ message complexity and $O(n)$ time complexity.

Different algorithms have been proposed to resolve the problem of leader election in complete networks. In [15] it is proposed an algorithm which requires $O(n\ log\ n)$ messages and $O(n\ /log\ n)$ time. This algorithm requires to know the total number of nodes in the system. Other proposals, [12, 13], describe algorithms whose cost are $O(n)$ both in messages as in time. But this algorithms assume previous knowledge about the system because they require *sense of direction*.

Sense of direction means, in general, that given a graph $G \equiv (V, E)$, being V the nodes of the graph, E its set of edges between nodes of V and $E(n)$ the edges of node n in G; there exists a labelling function which labels each edge $e \in E(n)$ for all $n \in V$ [6]. There are different alternatives for sense of direction functions, which derive in different sense of direction definitions.

Finally, [16] proposes an efficient algorithm for complete networks with sense of direction which reduces the latency to $O(log\ n)$, being its communication cost $O(n)$. In particular, [16] make use of the chordal definition of sense of direction.

Related work: In [12, 13, 16] algorithms with a linear cost are proposed. These algorithms consider some kind of sense of direction. In fact, [16] considers a complete network and it uses the chordal sense of direction definition. In this case, each node has a label for each of its edges denoting the distance between nodes following a Hamiltonian cycle. These labels allows each node to distinguish the other N-1 nodes of the system before the election is initiated. Moreover, this information is used in the algorithm to select a number of k nodes to initiate the first step of the algorithm. Within the second step of the algorithm, a leader is selected between the nodes that have initiated the first step concurrently.

The proposal presented in [12] consider [13] as the start point to derive a distributed fault–tolerant leader election algorithm. So, without failures the algorithm correspond to [13]. In both cases the algorithm assume that initially each node knows some information. In particular, each node knows the global topological information of the network and has a label for each link with other nodes in the system. Moreover, in these papers, as in [16], it is important to know the complete set of nodes when the algorithm is initiated. The first phase of the algorithms depends on the total number of nodes.

Our algorithm assumes, as other proposals for complete networks, that there is a communication system, which allows to communicate each pair of nodes of the system. However, we do not assume that nodes know the complete set of nodes nor the total number of nodes. But requires that nodes are organized in a logical ring.

The assumption of the existence of a virtual ring can be resolved in different ways. First of all, it could be a parameter of the system. That is, the servers or process of the system are configured in such a way that they define the virtual ring. This consideration is realistic in server farms or when we develop an application where the process knows each other in order to communicate together.

In addition, virtual ring can be built on-line in server farms by using a simple mechanism. Each server can monitor the packets transmitted by other servers. Each server uses two variables (highest, lowest) where it stores the value of the greatest and the lowest IP present in the monitored packets. Whenever server IP is lower than the value present in its highest variable, then the server selects such IP as the next node in the ring. In other case the node selects the value present in its lowest variable.

Moreover, in the case we have a set of servers distributed over Internet, it is possible to perform the virtual ring definition with a message complexity lower than the required to obtain any kind of sense of direction [17], which is $O(n^2)$. It could be the case of P2P networks.

Contributions of the paper:

In this paper we consider the problem of electing a leader in asynchronous complete networks without sense of direction. Its cost is linear in number of messages and latency, $O(n)$. In comparison with other proposals, the algorithm has a similar cost to [13, 12, 16], but these algorithms assume global sense of direction. In addition, they require to know the complete set of nodes present in the system. Other papers that do not consider sense of direction and complete networks, require a cost $O(n\ log\ n)$.

Our algorithm does not require global sense of direction, nor the whole set of nodes present in the system. In our case, we assume there is a virtual ring connecting logically

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

the nodes of the system. However, such assumption can be realistically obtained with a lower cost than the required for global sense of direction.

The paper is organized as follows. Section 2 describes the computation model of system. In Section 3 we present the algorithm and an execution example. Then, we prove that the algorithm is a leader election algorithm and calculate its costs in Section 4. Conclusions and References end the paper.

2. Model of distributed computation

We consider the network as a complete graph with nodes representing the processors and edges representing bidirectional communication channels between the processors. The message transmission time is assumed to be finite but unpredictable. We assume that channels neither loss messages nor corrupt messages. However, messages can be reordered.

We assume that the processes have unique identity (UIDs), chosen from some totally ordered space of identifiers; each process' UID is different from each other's in the network, but there is no constraint on which UIDs actually appear. Therefore, each node have distinct identity. Initially, a node knows only its own identity. In the following, by node i, we will mean a node with identity i.

Nodes do not know the whole set of identities of other nodes in the system. It is only required that each node knows at least one node to which send the messages being each node known by at least one node (virtual ring).

In the problem of leader election, initially all nodes are passive. Usually, an arbitrary subset of nodes, called the candidate nodes, wake up spontaneously and start the protocol. The leader is determined from the set of nodes that are candidates to be leader at the time the algorithm is initiated. When the protocol terminates, one node must announce itself as the leader. In the proposed algorithm, we consider that all nodes can start the algorithm at different times (whenever the preconditions to initiate the algorithm are satisfied).

3. Leader election algorithm

In this section we describe the algorithm and provides an execution example to illustrate its performance.

3.1. Algorithm overview

Initially, the whole set of nodes are $passive$ from the point of view of the algorithm. Under certain conditions, the nodes can wake up spontaneously to start the algorithm. These nodes are then candidates to be the new leader. The variable $status_i$ stores the state of the node i respect to the algorithm.

Each candidate explores part of the virtual ring. In order to reduce the message cost, we want that different candidates do not explore the same part of the ring. Therefore, we establish a collaboration mechanism between the candidates. Candidates are the only nodes that could be elected as leader, so the algorithm treats to change the state of candidates to $dummy$ ($dummy$ nodes do not participate in the leader election algorithm).

The collaboration between candidate nodes is based on a priority schema. Candidates eventually know the identity of the candidates that precedes them in the virtual ring. Then higher priority candidates ask to lower priority ones for information about their predecessors. Lower priority candidates change their status to dummy after replying to higher priority ones. Therefore, when a candidate receives a reply, it knows again the identity of the candidate that precedes it in the virtual ring. Following the same schema, eventually the highest priority candidate will know that all nodes in the system except itself are in dummy state.

The algorithm of Figure 1 formalizes the previous schema. The algorithm is initiated by any $passive$ node, i, that wake-up spontaneously by executing an action $initiate_i$. Such action initiates the state variables of node i and set $status_i$ to $candidate$. In addition, the candidate i sends an $ALG(i)$ message to its successor in the virtual ring. Obviously, this fact can arise (simultaneously or not) in several nodes of the system.

When an $ALG(init)$ message reaches a node i, it is handled by an action $rcvALG_i(j, m)$. If the node i is $passive$, it becomes $dummy$ and forwards the message towards its successor in the virtual ring. However, if the node is $candidate$, it sets to $init$ its variable $cand\_pred_i$ (initially NULL) to store the fact that all the nodes between $init$ and i are $dummy$. Now the candidate coordination mechanism begins.

If $i > init$, then i may send to $init$ an $AVS(i)$ message. Now i waits for information sent by $init$ about the identity of the candidate that precedes $init$ in the virtual ring (this fact is reflected setting $status_i$ to $waiting$). It is also possible that i had also received an $AVS(k)$ message from another node k. In such a case, variable $cand\_succ_i$ would contain the first $candidate$ successor of i in the virtual ring. As k has greater priority than i, i becomes $dummy$ and sends to k an $AVSRSP(init)$ message. Note that this message means that there a path of $dummy$ nodes from $init$ to k. Obviously, if i receives an $ALG(i)$ message, it means that there is no other $candidate$ in the virtual ring, so i becomes the leader of the system.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

$initiate_i$
Pre: $status_i = passive$.
Eff: $status_i \leftarrow candidate$;
$cand\_succ_i \leftarrow$ NULL;
$cand\_pred_i \leftarrow$ NULL;
send ALG(i) to $neigh_i$.

$rcvALG_i(j, m)$
Pre: $channel(j, i) = ALG(init) \bullet \gamma$.
Eff: $channel(j, i) \leftarrow \gamma$;
[$status_i = passive$]
$status_i \leftarrow dummy$;
send ALG($init$) to $neigh_i$;
[$status_i = candidate$]
$cand\_pred_i \leftarrow init$;
[$i > init$]
IF ($cand\_succ_i =$ NULL) THEN
$status_i \leftarrow waiting$;
send AVS(i) to $init$;
ELSE
send AVSRSP($cand\_pred_i$) to $cand\_succ_i$;
$status_i \leftarrow dummy$;
[$init = i$]
$status_i \leftarrow leader$.

$rcvAVS_i(j, m)$
Pre: $channel(j, i)$ = AVS(j) $\bullet\gamma$.
Eff: $channel(j, i) \leftarrow \gamma$;
[$status_i = candidate$]
IF ($cand\_pred_i$ = NULL) THEN
$cand\_succ_i \leftarrow j$;
ELSE
send AVSRSP($cand\_pred_i$) to j;
$status_i \leftarrow dummy$;
[$status_i = waiting$]
$cand\_succ_i \leftarrow j$.

$rcvAVSRSP_i(j, m)$
Pre: $channel(j, i) = AVSRSP(k) \bullet \gamma$.
Eff: $channel(j, i) \leftarrow \gamma$;
[$status_i = waiting$]
IF ($i = k$) THEN
$status_i \leftarrow leader$;
ELSE
$cand\_pred_i \leftarrow k$;
IF ($cand\_succ_i =$ NULL) THEN
IF ($k < i$) THEN
$status_i \leftarrow waiting$;
send AVS(i) to k;
ELSE
$status_i \leftarrow dummy$;
send AVSRSP(k) to $cand\_succ_i$;

Figure 1. Leader election algorithm.

When a $candidate\ i$ receives an $AVS(j)$ message from j, it knows that j is its $candidate$ successor in the virtual ring. It also knows that $j > i$ and that all nodes from i to j are in $dummy$ state. If it knows its $candidate$ predecessor in the virtual ring ($cand\_pred_i \neq$ NULL), then it sends to j such a node with an $AVSRSP(cand\_pred_i)$ message and becomes $dummy$.

Note that an $AVSRSP(n)$ messages from i to j carries the same information that $ALG(n)$ messages. The identity of the $candidate$ predecessor of the node towards the message goes. So, the final effects of this message is simulating the transmission of an $ALG(n)$ message through a path yet known by j. If neither i is $candidate$ nor i knows its $candidate$ predecessor, i stores j in $cand\_succ_i$ in order to communicate j the identity of the $candidate$ that i will receive.

Note that a node i collects information by two different ways. If j is the first $candidate$ predecessor of i in the virtual ring, i will receive an ALG message from j storing the path from j to i. If there are other candidates between i and j, one of them will send to i an $AVSRSP$ message. Thus, the effects of receiving an $AVSRSP$ message are quite similar to those of receiving an ALG message. Those actions differs only in one fact. A node that receives a $AVSRSP$ message have to be a $waiting$ node.

3.2. Execution example

Algorithm execution is illustrated with the following example (figure 2). Initially, we have a set of four nodes as showed in figure 2.i. In this case, nodes are organized logically in a ring but all of them are complete connected. Thus, there is a communication channel between each pair of nodes, as shown in figure 2.i.

Nodes *2* and *4* initiate the algorithm by sending an ALG message to their neighbours in the logical ring (figure 2.ii). The ALG message contains the initiator of the message. Note that nodes do not require to know the complete set of nodes in the system, it is enough that each node knows its neighbour in the logical ring.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

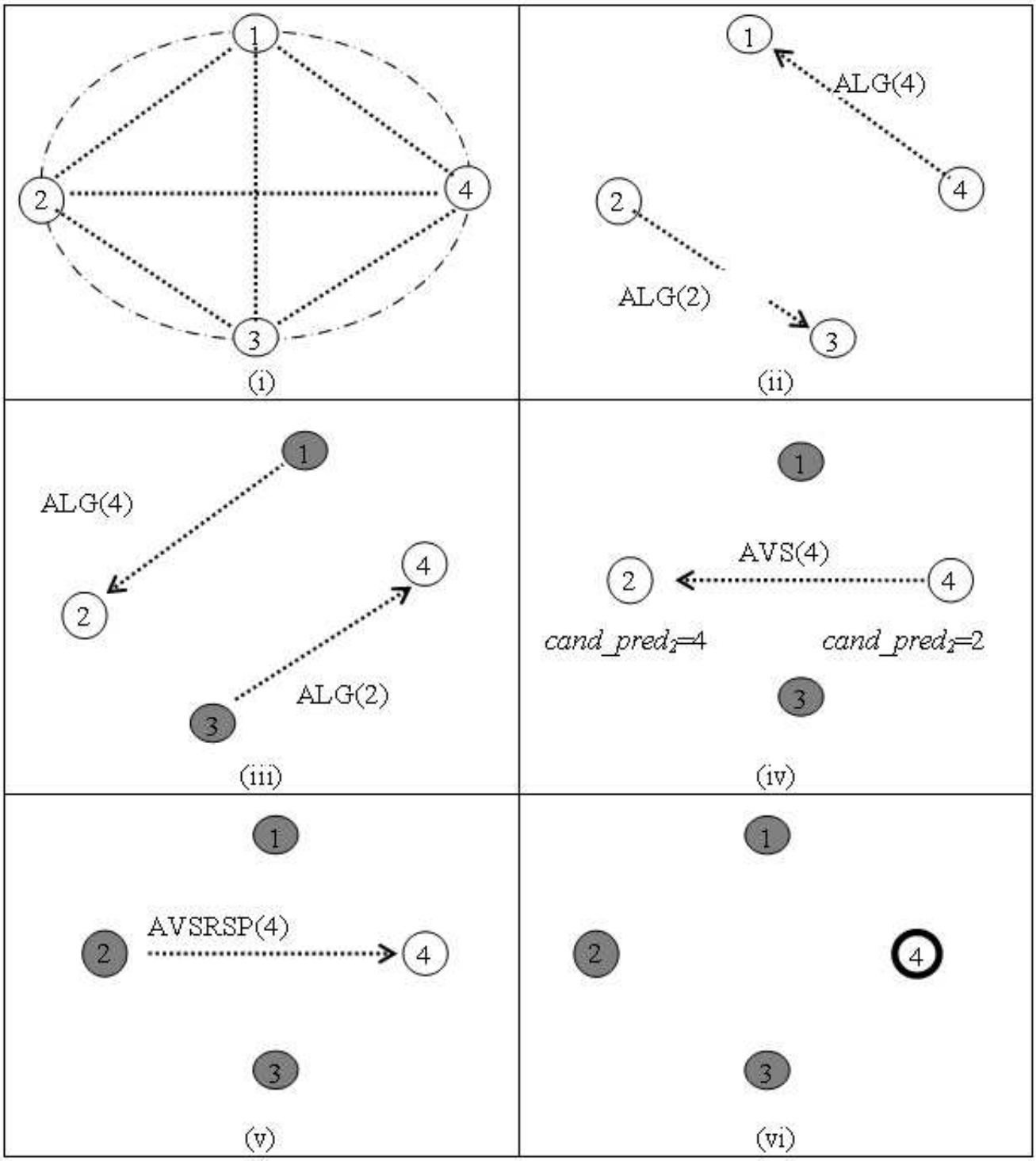

Figure 2. Example of usage of the algorithm.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

When nodes *1* and *3* handle the ALG message, they have not yet initiated the algorithm ($status = passive$). Thus, they become $dummy$ and forward the ALG message (figure 2.iii). Nodes *1, 3* will not initiate the algorithm, because they are not $passive$ nodes, but $dummy$.

In figure 2.iv, $candidate$ nodes receive the message initiated by other nodes. As node *2* has lower priority than *4*, it stores in its $cand\_pred_2$ the initiator of the received message *4*.

In the same way operates node *4*. However, it is a higher priority node than *2*. Then, node *4* knows that it has a lower priority predecessor, candidate node *2*. Thus, *4* asks node *2* about its known candidate predecessor by sending an $AVS(4)$ message to *2*.

When a node receives an AVS message, it responds as soon as it knows its candidate predecessor node. In this case, node *2* sends an $AVSRSP(4)$ to answer node *4* (figure 2.v) and it becomes $dummy$. The $AVSRSP$ message contains the information collected at node *2*, which will be analyzed at node *4* In this case, figure 2.vi, node *4* verifies that it satisfies the leader condition and declares itself as leader of the system.

4. Algorithm correctness and performance measurements

In this section we prove that the algorithm is a leader election one. We also show that its communication cost and latency are O(n).

First, we prove that algorithm is a leader election algorithm. To do so, we have to show that eventually the $status$ of a node i is set to $leader$ (liveness), but also that when it arises there is not another node with the same status (safety). We begin showing this second property.

Theorem 1 *If there is a node i whose $status_i$ is $leader$, then the $status$ of the rest of nodes is $dummy$.*

Proof: Initially, all nodes in the system are in $passive$ status. Some of them will initiate the algorithm becoming thus $candidate$ and sending an ALG message. Each $passive$ node that handles an ALG message —$rcv\_ALG_i, (j, m)$—, becomes $dummy$. It is obvious that a $dummy$ node never changes its $status$, so all the nodes of the virtual ring from a candidate i and a message $ALG(i)$ are $dummy$ nodes (1).

If a node becomes leader by effects of a $rcv\_ALG_i, (j, m)$ action, by (1) the rest of nodes are dummy and the theorem holds.

The other case arise when more than one node spontaneously initiates the algorithm. Once an $ALG(i)$ message reaches to a candidate j —$rcv\_ALG_j, (k, ALG(i))$—, this message is lost after setting $cand\_pred_j$ to i. Thus, by (1) the status of all nodes from i to j are $dummy$. Therefore, $cand\_pred_j \neq NULL$ implies that all nodes from j to $cand\_pred_j$ are $dummy$ (2). In some cases, j replies to i with an $AVS(j)$ message and changes its status to $waiting$. So if a message $AVS(j)$ goes towards a node i then the path from i to j only contains $dummy$ nodes (3).

When $AVS(j)$ reaches i —$rcv\_AVS_i, (j, AVS(j))$—, $cand\_succ_i$ may change to j. Thus, by (3) when $cand\_succ_i$ changes to j, all nodes from i to j are dummy. If i knows its previous candidate ($cand\_pred_i \neq NULL$), it sends to j a message $AVSRSP(cand\_pred_i)$ changing its own status to $dummy$. Thus, by (2) and (3) the path from $cand\_pred_i$ to the node towards goes the $AVSRSP(cand\_pred_i)$ message only contains $dummy$ nodes (4).

The reception of an $AVSRSP(j)$ message by a node i —$rcv\_AVSRSP_i, (k, AVSRSP(i))$— can make that i changes its status to $leader$. This arises when $j = i$, but by (4), in such a case, all the nodes of the system are in $dummy$ status and the theorem holds. ■

In conclusion, when the algorithm terminates, it terminates with the desired results. Now we show that the algorithm actually terminates.

Theorem 2 *Once the algorithm begins, the status of one node will be eventually set to $leader$.*

Proof: During a run of the algorithm only a subset $S \subseteq N$ of nodes of the system will spontaneously change their status from $passive$ to $candidate$. The execution of $initiate_i$ has the effect of sending an $ALG(i)$ message towards the successor of i in the ring. The nodes whose status is $passive$ forwards $ALG(i)$ messages, so if $\mid S \mid = 1$, this message will be forwarded through the ring and will eventually reach i setting $status_i$ to $leader$.

Assume $\mid S \mid = 2$. Let i and j be the nodes that change their status to candidate. As effects of this change, both nodes send an $ALG(i)$ message to their successors in the virtual ring. Passive nodes forward those messages so, eventually j receives an $ALG(i)$ message and i an $ALG(j)$ message. Assume that $i > j$. In such a case, when i receives $ALG(j)$, as $cand\_succ_i$ = NULL, it sends an $AVS(i)$ to j. Candidate j may receive the $ALG(i)$ message before receiving the $AVS(i)$ message. In such a case, $cand\_pred_j$ will be set to i. Therefore, when j receives the $AVS(i)$ message, it sends an $AVSRSP(i)$ to i. This message will eventually reach i setting $status_i$ to leader.

Assume $\mid S \mid \geq 2$. We can always find three nodes of S verifying that i is the candidate successor of j in the virtual ring and j is the candidate successor of k in the virtual ring being $i > j$ and $k > j$. Eventually, i and j will receive the ALG messages sent by j and k. i will reply to j with an $AVS(i)$ message. (1)If this message reaches j before the $ALG(k)$, when i receives the $ALG(k)$ message $cand\_succ_j$ will be set to i, and $AVSRSP(k)$

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

message will be sent to j. Additionally, the status of j will be set to dummy. (2) If $ALG(k)$ reaches j before $AVS(i)$, then $cand\_pred_j$ is set to k. Therefore, when $AVS(i)$ reaches j, the status of j changes to dummy while j sends $AVSRSP(k)$ message to i. In both cases, when $AVSRSP(k)$ is handled by i, the effects are the same that we would have obtained if j never had been $candidate$ and $ALG(k)$ would be handled by i. So, the pairs of messages AVS $AVSRSP$ reduce the number of candidates (the number of elements of S). Eventually $\mid S \mid$ will be 2 and the previous case concludes. ■

We have declared that the algorithm has lineal cost. This fact arises because each node initiates the algorithm only one time. Therefore, during a run of the algorithm at most only n ALG messages are sent.

Lemma 1 *Each node handles at most one ALG message.*

Proof: ALG messages are sent (towards the successor in the virtual ring) by effects of $initiate_i$ actions. This action is enabled for those nodes i verifying $status_i = passive$. After its execution, the status of node i changes to candidate. Additionally, only passive nodes forward ALG messages to its successors in the virtual ring, but this action change their status to $dummy$. Therefore, as no action transforms a node into $passive$ each $ALG(i)$ message is sent by the candidate that created it and the nodes between two candidates. Consequently, each node only receives one ALG messages. ■

The leader in the system will be the highest priority node that initiates the algorithm. Other candidates will be requested to send the information about the ring they collect along the algorithm execution. Thus, other source of messages in the algorithm is the number of AVS messages a node can receive, but the next lemma shows that each node only receives one of such messages.

Lemma 2 *Each node receives at most one AVS message.*

Proof: Candidates, j that receives an $ALG(i)$ message may send an $AVS(j)$ to i. All nodes from i to j are in $dummy$ state and j does not forward $ALG(i)$, so no other node can send an AVS message to i. Node i eventually replies to j with $AVSRSP(cand\_pred_i)$. It is obvious that $cand\_pred_i \neq i$, so despite the reception of such a message can make that j sends an $AVS(j)$ message, this message goes towards $cand\_pred_i$, not towards i. ■

Finally, the number of $AVSRSP$ messages sent during a run is the same as the number of AVS messages.

Lemma 3 *Each node sends at most one $AVSRSP$ message.*

Proof: Only a node that have received an AVS message send an $AVSRSP$ message, so the previous lemma concludes. ■

The above lemmas allows to establish the communication cost to be $O(n)$.

Theorem 3 *The algorithm selects a leader with O(n) messages.*

Proof: Each node can initiate the algorithm. Because of the relative priorities between the nodes, nodes store messages created by higher priority nodes and request for information to nodes with lower priority nodes. By lemma 3, each node at most handles one AVS and responds with the corresponding $AVSRSP$ message becoming a $dummy$ node. By theorem 2, the algorithm selects a leader, then at most each node sends its ALG message and the AVS and the corresponding $AVSRSP$. ■

Finally, the latency to resolve the election problem is similar to the communication cost and it is $O(n)$.

Theorem 4 *The algorithm selects a leader in O(n) time units.*

Proof: In order to measure the latency of the algorithm, the worst case occurs when the node that will be selected as leader requests the information of the other nodes sequentially (one after the other). ■

5. Conclusions

We have shown a leader election algorithm for complete networks that selects the leader interchanging $O(n)$ messages during $O(n)$ time units. Our algorithm achieves these performance without making use of *sense of direction*. It only needs the existence of a virtual ring. Therefore, the cost of the algorithm is similar to the cost of previous proposals as [13, 12, 16], but it needs less knowledge about the system. Other proposals for complete networks that do not consider sense of direction require $O(n\ log\ n)$ messages. In addition, we do not require to know the total number of nodes in the system.

Acknowledgment

This work was supported by the Gobernment of Navarra and CICYT under grant number TIC2003-09420-C02-02.

References

[1] Y. Afek and E. Gafni. simple and efficient distributed algorithms for election in omplete networks. In *Proceedings of the 22nd annual Allerton Conference on Communication, Control and Computing*, pages 689–698, 1984.

[2] Y. Afek and E. Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. *SIAM J. Comput.*, 20:376–394, 1991.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE

[3] G. Attiya and Y. Hamam. Two phase algorithm for load balancing in heterogeneous distributed systems. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP'04*. IEEE Society, 2004.

[4] B. Awerbuch. Optimal distributed algorithms for minimal weight spanning tree, counting, leader election and related problems. In *Proceedings of the ACM symposium on Theory of Computing*, pages 230–240. ACM Press, 1987.

[5] A. Bartoli. Implementing a replicated service with group communication. *Journal of System Architecture*, 50:493–519, 2004.

[6] P. Flochini, B. Mans, and N. Santoro. Sense of direction: Definitios, properties and classes. *Networks*, 3(32):165–180, 1998.

[7] G. Frederikson and N. Lynch. The impact of synchorous communication on the problem of election a leader in a ring. In *Proceedings of the ACM symposium on Theory of Computing*, pages 493–503. ACM Press, 1987.

[8] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimal spanning tree. *ACM Trans. Program. Lang. Syst.*, 30:66–77, 1983.

[9] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER02)*. IEEE Society, 2002.

[10] D. Hirschberg and J. Sinclair. Descentralized extrema-finding in circular confgiruations of processors. *Comm. Assoc. Comput. Mach.*, 23:627–628, 1980.

[11] E. Korach, S. Kutten, and S. Moran. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theoret. Comput. Sci.*, 64:125–132, 1989.

[12] T. Masuzawa, N. Nishikawa, K. Hagihara, and N. Tokura. Optimal fault-tolerant distributed algorithms for election in complete networks with a global sense of direction. In *Proceedings of the 3rd International Workshop on Distributed Algorithms*, pages 171–182. Springer-Verlag, 1989.

[13] C. L. Michael, A. M. Teresa, and B. W. Douglas. Election in a complete network with a sense of direction. *Inf. Process. Lett.*, 28(6):327, 1988.

[14] B. K. G. A. R. Jiménez, M. Patiño. Improving scalability of fault-tolerant database clusters. In *Proceedings of the 22nd IEEE Int. Conf. on Distributed Systems*, pages 477–485. IEEE Society, 2002.

[15] G. Shingh. Leader election in complete networks. *SIAM J. Comput.*, 26(3):772–785, 1997.

[16] G. Singh. Efficient leader election using sense of direction. *Distributed Computing*, 10(3), 1997.

[17] G. Tel. *Distributed Algorithms*. Cambridge University Press, 2nd Edition - 2001.

Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euromicro-PDP'05)
1066-6192/05 $ 20.00 IEEE