

Leader Election in Oriented Star Graphs*

Wei Shi[†], A. Bouabdallah[‡], and Pradip K Srimani[§]

November 17, 2004

Abstract

Leader election in a network plays an important role in the area of distributed algorithm design. Structural properties of the network as well as presence of direction on the edges of the network greatly affects the complexity of the leader election problem which is primarily measure by the message complexity of the protocol. Our purpose in the present paper is to adapt the existing distributed match making concepts to design a linear time leader election algorithm for star graphs. Star graphs have been extensively studied as attractive alternative for the well known hypercubes for network design. Linear election algorithms for oriented hypercubes is known; no such algorithm exist for oriented star graphs.

Keywords: Leader Election, Star Graphs, Distributed Match Making, Tournament Scheme, Message Complexity.

1 Introduction

Leader election in a network is one of the most important problems in the area of distributed algorithm design. Consider any network of N nodes; a *leader* node is defined to be any node of the network unambiguously identified by some characteristics (unique from all other nodes). A leader election process is defined to be a uniform algorithm (code) executed at each node of the network; at the end of the algorithm execution, exactly one node is elected the *leader* and all other nodes are in the non-leader state. Gallager et. al. [1] have developed a leader election algorithm for arbitrary networks whose message complexity is $\mathcal{O}(N \log N + E)$ where N is the number of nodes and E is the number of edges in the network. Santoro in [2] has studied how the knowledge of topology of the network and the orientation of the links in the network affect the message complexity of leader election algorithms in networks. Subsequently, many authors have studied leader election algorithms for various kinds of networks with or without link orientations. Existence of orientation

*Address for Correspondence: Pradip K Srimani, Department of Computer Science, Clemson University, Clemson, SC 29634, Email: srimani@cs.clemson.edu, Voice: (864) 656-7552, Fax: (864) 656-0145

[†]Department of Computer Science, Colorado State University, Colorado State University, Ft. Collins, CO 80523

[‡]Heudiasyc CNRS UMR 6599, Univ. de Technologie de Compiègne, BP 20529, 60205 Compiègne, France

[§]Department of Computer Science, Clemson University, Clemson, SC 29634

of the links have been shown not to improve the message complexity of leader election for either rings or tori [3], while that helps for cliques; the lower bound on message complexity of leader election in oriented cliques is $\mathcal{O}(N)$ [4] and that for unoriented cliques is $\Omega(N \log N)$ [5]. Authors in [4] have considered complete networks with a sense of direction. Authors in [6, 7] has developed leader election for oriented hypercubes whose message complexity is linear in the number of nodes. Research in leader election in both unoriented and oriented graphs have continued [8, 9, 10].

Another very efficient interconnection topology, called star graphs [11, 12], has been extensively investigated in the literature [13, 14] since these graphs seem to enjoy most of the desirable properties of the hypercubes at considerably less cost; they accommodate more nodes with less interconnection hardware and less communication delay. In this paper, our purpose is to propose an election algorithm for the oriented star graphs that uses $\mathcal{O}(N)$ messages. As far as we know, no leader election algorithm exists for these star graphs although the straightforward application of existing algorithms would give a leader election algorithm for unoriented star graphs with $\mathcal{O}(N \log N)$ message complexity. It is not clear if one can design a linear election algorithm for unoriented stars.

2 Star Graph Networks

The n -dimensional star graph, S_n , is an edge- and node-symmetric graph (see [15] for definitions and topological properties) containing $N = n!$ nodes and $n!(n-1)/2$ edges (links). Each node in S_n is assigned a label, a distinct permutation of the set of integers $\{1, \dots, n\}$. Two nodes are joined with a link labeled i (edge with *direction* i) if and only if the label of one can be obtained from the label of the other by swapping the first digit (conventionally the leftmost) and the i th digit where $1 < i \leq n$. For instance, in a S_5 , containing 120 ($= 5!$) nodes, two nodes [12345] and [42315] are neighbors and joined via a link labeled 4. The label of the graph S_n is shown as: $[x_1 x_2 \dots x_n]$ where x_i 's collectively represent all permutations of the first n integers. Consider the graph S_4 as shown in Figure 1; we label it as $[x_1 x_2 x_3 x_4]$; by fixing any single position in the label of S_n to an integer (except for position 1 of course), the label of a specific S_{n-1} will result; .

Remark 1 *In any S_n the above labeling is not unique; there exist many possible different label assignments with the said property. We arbitrarily choose one such labeling and refer to it as canonical labeling and will refer to the nodes using its canonical label. The canonical label of node u in S_n is denoted by $u_1 u_2 \dots u_n$ (a permutation of $\{1, 2, \dots, n\}$).*

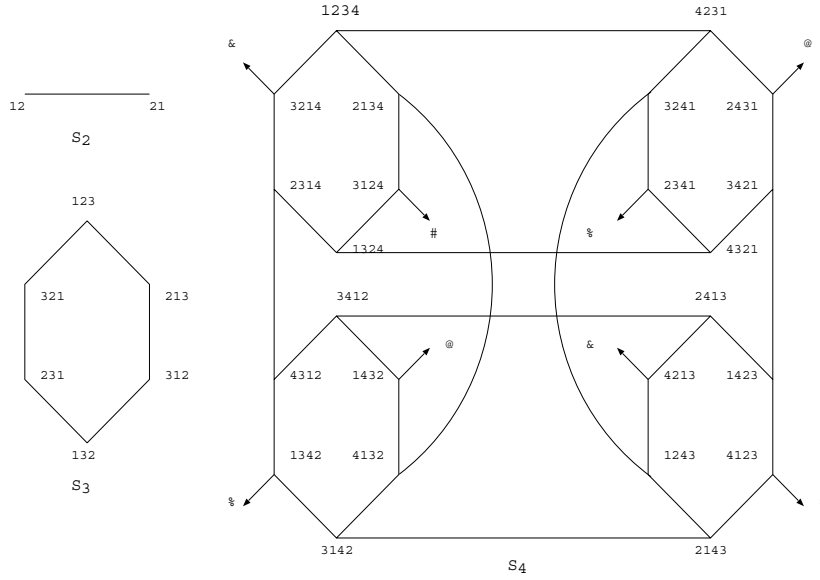


Figure 1: Example Star Graphs: S_2 , S_3 , and S_4

The topological properties of the star graph have been derived and discussed elsewhere [13, 16, 17, 18, 15]. We briefly state the properties that are relevant to leader election.

Remark 2

1. S_n is both vertex and edge transitive and hence symmetric [A vertex-transitive graph is a graph whose automorphism group is transitive; an edge-transitive graph is a graph such that any two edges are equivalent under some element of its automorphism group; a graph is symmetric when it is both vertex and edge transitive [19]].
2. S_n contains $n!(n - 1)/2$ links and has a diameter of $\lceil 3(n - 1)/2 \rceil$.
3. Each S_n contains n mutually vertex disjoint S_{n-1} 's [consider the S_4 in Figure 1; it contains 4 mutually disjoint S_3 's e.g., $\{(1234, 2134, 3124, 1324, 2314, 3214)\}$, $\{4321, 2431, 3421, 4321, 2341, 3241\}$, $\{2413, 1423, 4123, 2143, 1243, 4213\}$, $\{3412, 1432, 4132, 3142, 1342, 4312\}$]; generalizing this, for any integer $i, 1 \leq i \leq n$, the star graph S_n contains $\frac{n!}{i!}$ substars (S_i s) of dimension i .

Definition 1 In a star graph S_n , each node has $n - 1$ links. We assign values from 2 to n as the directions of these $n - 1$ links, corresponding to the position that the first symbol is switched to in the label of neighbor node the link leads to. Note that there is no link or direction 1; we will use the terms link or direction interchangeably.

Definition 2 For any node $x \in S_n$, $S^\ell(x)$ is defined to be the set of nodes that can be reached from node x using only the links of dimensions from 2 to ℓ .

For example, in S_4 , $S^3(1234)$ is a 3 dimensional substar consisting of the 6 nodes: 1234, 1324, 2134, 2314, 3124, 3214; all these nodes can be reached from 1234 using edges with directions 2 or 3.

Remark 3 For any node $x \in S_n$, $S^\ell(x)$ is denoted by $[* \cdots * x_{\ell+1}, \cdots x_n]$; note that given any integer ℓ , $2 \leq \ell \leq n$, any node x can compute the set $S^\ell(x)$ – the node id x is used for notational convenience only; any node x can compute the set $S^\ell(x)$ from the knowledge of the direction of the links adjacent on x , without knowing the id x explicitly; this is not possible in an unoriented star graph. Also note that for any node x , $x \in S^\ell(x)$.

3 Leader Election in Oriented S_n

As noted in the introduction, a *leader* node is defined to be any node of the network unambiguously identified by some characteristics (unique from all other nodes); leader election is defined to be a protocol (algorithm) which is executed at each node of the network and at termination, exactly one node is *elected* to be the leader. We start with the following observation.

Remark 4 If each node knows its canonical label, this election process is trivial. Consider the node having the label $12 \cdots n$, the natural permutation of the first n integers; we can say that the node with this label is the leader and all other nodes are non-leader.

Orientation: In this paper, as in [6, 7], we assume that the nodes in the star graph do not know their canonical labels. We will still refer to the nodes by some canonical labels for convenience of reference, but these labels or names have no topological significance [6]; these labels are not used in the algorithm. We also assume in this paper that the network is

an *oriented* star in the sense that each node is aware of the direction of the links incident to it (in contrast, a node in an un-oriented star graph distinguishes its adjacent links by different but uninterpreted names), i.e., in an oriented star a node knows which incident edge is of direction 2 or 3 or such while in an unoriented star the node knows its different edges but does not know which edge has what direction.

The election algorithm uses the general approach of using a tournament scheme based on the recursive structure of the star graph. Consider a star graph S_n of dimension n which contains n vertex-disjoint substars S_{n-1} of dimension $n - 1$. The algorithm first recursively elects a leader in each of the S_{n-1} s, and then elects one of these $n - 1$ leaders to be the leader of S_n . The leaders at different stages of recursion are distinguished by the level of recursion. If a node is elected to be the leader of an i -dimensional star S_i , it is called an i -leader. Election of a leader in S_1 is easy; it has only one node which is immediately elected to be the 1-leader. Election of i -leaders for other values of i is done by using a technique called *match making* as explained in the next section. The algorithm works in a bottom-up way.

3.1 Match Making

Assume that an i -leader has been elected for each of the $(i + 1)$ substars S_i of dimension i . To elect the $(i + 1)$ -leader of the substar S_{i+1} comprising all of the $(i + 1)$ substars S_i , each i -leader can send its own name to all other i -leaders; the node receiving no smaller name than its own becomes the $(i + 1)$ -leader and all other i -leaders becomes a non-leader. The problem here is that the i -leaders do not know the identity of the other i -leaders and how to reach them. A simple solution of this problem is to have each i -leader broadcast its name to all the nodes in the star graph S_{i+1} and then each node can concurrently determine if it is the leader node. But, the cost would be $\mathcal{O}(N)$ messages. We develop a better algorithm that uses only $\mathcal{O}(\sqrt{N})$ messages by adapting the concepts from [20] to the star graph topology.

Strategy: We observe that in order to elect the $(i + 1)$ -leader from among the i -leaders, at least one node in the entire S_{i+1} must be informed of the names (ids) of all the i -leaders; this node can then decide the $(i + 1)$ -leader (based on some criteria like whose id is the biggest) and transmit that information to all the i -leaders. Assume $\mathcal{N}_A(x)$ and $\mathcal{N}_B(x)$ are two sets in a given star graph S_n that can be defined for any arbitrary node $x = x_1x_2 \dots x_n \in S_n$; for any two arbitrary nodes $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$ in S_n , we have that the intersection $\mathcal{N}_A(x) \cap \mathcal{N}_B(y)$ contains at least one node z . Then, the node x can broadcast its name to all nodes in $\mathcal{N}_A(x)$, node y can broadcast its name to all nodes in $\mathcal{N}_B(y)$ and

node z can decide the leader of the two nodes x and y and transmit the information to both of them. Intuitively, the set $\mathcal{N}_A(x)$ consists of at least one node from each of the mutually disjoint substars (of a specific size) in S_n ; then any arbitrary node y must belong to exactly one of those substars; $\mathcal{N}_B(y)$ is the set of all nodes of the specific substar containing node y . In order to make match making optimal, we need to design the two sets $\mathcal{N}_A, \mathcal{N}_B$ such that (1) they are of minimal possible size, (2) it takes least number of messages to generate the two sets, and (3) $|\mathcal{N}_A(x) \cap \mathcal{N}_B(y)|$ is one. We develop two algorithms **A** and **B** to respectively compute the sets \mathcal{N}_A and \mathcal{N}_B . Before we design A and B, we need to design a procedure *Mult* that will be used in both the algorithms.

Remark 5 [Note:] *Although there is no link or direction 1, sometimes in the algorithm description a node may send a message along direction 1 which means no physical message is generated.*

3.1.1 Procedure *Mult*

Algorithm *Mult*(k, M) can be invoked at an arbitrary node $x = x_1 \dots x_n$ in star graph S_n . The algorithm takes two parameters: the first one is an integer $k, 1 < k \leq n$, and the other one is M which is a value of any type. As stated in Remark 3, the set $S^k(x)$ consists of all nodes in the substar $[* \dots * x_{k+1} \dots x_n]$, i.e., k mutually disjoint $(k-1)$ -substars $[* \dots * x_1 x_{k+1} x_n], \dots, [* \dots * x_k x_{k+1} \dots x_n]$. The purpose of algorithm *Mult*(k, M) is to transmit the value M to a set of nodes which contains exactly one node from each of those $(k-1)$ -substars. This algorithm is only a carrier of message M . It does not directly process M . Each node receiving M determines to keep or not to keep a local copy of M for later reference. Only the nodes that keep M can use M at a later time. A node, receiving M , may re-transmit it and may choose not keep a local copy of M . The algorithm is executed in a distributed fashion: it consists of sending two types of messages, *MessageS*(k, M) and *MessageP*(k, M) (the messages carry the parameter k and some value M) from the origin (node x). Any node receiving any of the messages responds synchronously. When a node receives *MessageS*(k, M), it keeps a local copy of message M for future processing. When a node receives *MessageP*(k, M), it does not keep M but executes a procedure to relay the message. The details are described in the following pseudo-code.

Algorithm *Mult*(k, M) at x

Initial Conditions:

Node x has message M to be delivered.

Invocation of the Algorithm

Node x sends message $MessageP(k, M)$ and $MessageS(k, M)$ along direction 1.

Action at any node:

- **Upon receiving $MessageP(k, M)$ from the link (direction) dir :**
 for ($i = \lceil \log_2 dir \rceil, i < \lceil \log_2 k \rceil, i++$)
 {
 $d = dir + 2^i$;
 if $d < k$ then
 send $MessageP(k, M)$ along direction d ;
 }
 send message $MessageS(k, M)$ along direction k .

- **Upon receiving $MessageS(k, M)$:**
 Copy M into local memory.

Remark 6 When a node receives $MessageP$, it executes a two-step procedure. The first part is a while loop where the message $MessageP$ is propagated along a spanning tree of nodes with different first symbols in their labels. The second part is to send the message $MessageS$ along the edge with direction k .

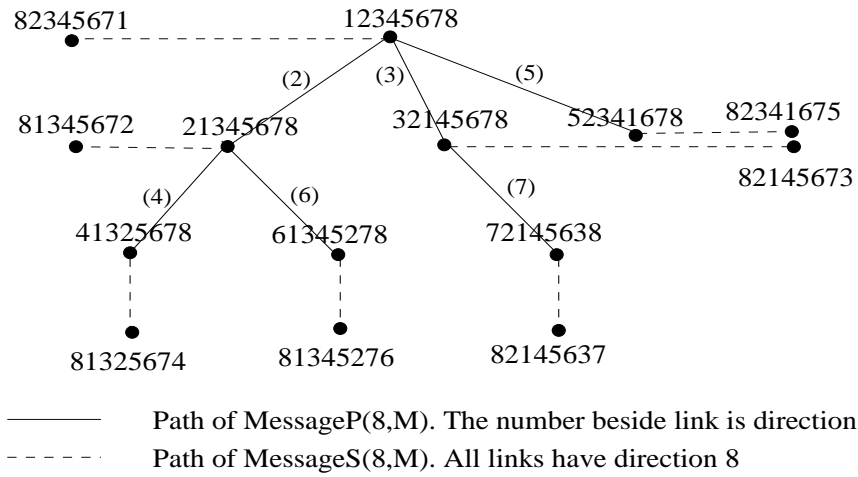


Figure 2: **Execution of $Mult(8, M)$ at Node 12345678 in an 8 Dimensional Star**

Example 1 Figure 2 shows the execution of algorithm $Mult(8, M)$ at node $x = 12345678$ in an S_8 .

Node 12345678 has message M and receives $MessageP(8, M)$ at invocation stage. After receiving $MessageP(8, M)$ from direction 1, node 12345678 execute a loop from $i = \lceil \log_2 dir \rceil = 0$ to $i < \lceil \log_2 k \rceil = \log_2 8 = 3$. $MessageP(8, M)$

is then sent through direction $1 + 2^0 = 2$, $1 + 2^1 = 3$, and $1 + 2^2 = 5$ and reaches node 21345678, 32145678, and 52341678 respectively. After the loop is finished, node 12345678 also send message $MessageS(8, M)$ along direction 8 to node 82345671.

When node 21345678 receives $MessageP(8, M)$ through direction 2, it executes a loop from $i = 1$ to $i < 3$, hence sends $MessageP(8, M)$ to node 41325678 and 61345278. And after the loop, it also sends $MessageS(8, M)$ to 81345672.

For the same reason, node 32145678 executes a loop from $i = 2$ to $i < 3$ and sends $MessageP(8, M)$ to 72145638 along direction 7. After that, it also sends $MessageS(8, M)$ to 82145673 along direction 8.

Other nodes receiving $MessageP(8, M)$ include 41325678, 61345278, 72145638, and 52341678. They do not generate any more $MessageP(8, M)$ due to the restraint $d < k$, but they still send $MessageS(8, M)$ along direction 8 and these messages reach 81325674, 81345276, 82145638, and 82341675 respectively.

As we can see that there are 7 nodes that receive $MessageP(8, M)$ and 8 nodes that receive $MessageS(8, M)$ (including node 12345678 that gets $MessageS(8, M)$ at invocation stage). Every node receiving $MessageP(8, M)$ have different first digit in their label, and every node receiving $MessageS(8, M)$ have different 8-th digit in their label, with each in a different dimension 7 star.

Lemma 1 When a node $x = x_1 \dots x_n$ executes algorithm $Mult(k, M)$, it sends the message $MessageP(k, M)$ to exactly $k - 1$ different nodes; each of these nodes receives the message via distinct directions, from 1 to $k - 1$.

Proof: Consider synchronized execution of the *while* loop at different nodes by variable i . Just before the i th loop ($0 \leq i \leq \lceil \log_2 k \rceil$), there are 2^i nodes that have received $MessageP(k, M)$. And these nodes received $MessageP(k, M)$ from directions 1 to 2^i . After that loop, 2^i more nodes receive the message along the links with direction ranging from $2^i + 1$ to $2^i + 2^i$. With the restriction that $d < k$, the message $MessageP(k, M)$ will not be sent along direction k . Thus, when the “while” loop ends, There are exactly $k - 1$ different nodes that have received the message $MessageP(k, M)$.
□

Lemma 2 When a node $x = x_1 \dots x_n$ executes algorithm $Mult(k, M)$, exactly one node from each of the $(k - 1)$ -dimensional substars, $[* \dots * x_1 x_{k+1} x_n]$, \dots , $[* \dots * x_k x_{k+1} x_n]$, receives $MessageS(k, M)$ and keeps a local copy of

M .

Proof: From Lemma 1, $k - 1$ nodes receive $MessageP(k, M)$ along directions $1, \dots, k - 1$, so these nodes have x_1 to x_{k-1} as their first digits in their label. After the *while* loop, all these nodes send $MessageS(k, M)$ along the link (direction) k ; thus, the $k - 1$ nodes receiving $MessageS(k, M)$ will have x_1 to x_{k-1} as their k -th digit in the label. The starting node $x = x_1 \dots x_k x_{k+1} \dots x_n$ has already received $MessageS(k, M)$ at the initial phase of the algorithm and hence there are total k nodes getting $MessageS(k, M)$ and each of them belongs to a different $(k - 1)$ -dimensional substar $[* \dots * x_j x_{k+1} \dots x_n]$, $1 \leq j \leq k$. \square

Lemma 3 *Given a node x and an integer k , algorithm $Mult(k, M)$ takes $\mathcal{O}(\log k)$ time and generates $2k - 1$ messages.*

Proof: The algorithm is executed concurrently at different nodes receiving $MessageP(k, M)$ message. At each node, the *while* loop takes no more than $\lceil \log k \rceil$ time units and the transmission of the $MessageS(k, M)$ takes one time unit. So the entire algorithm takes $\mathcal{O}(\log k)$ time. From Lemmas 1 and 2, there are $k - 1 + k = 2k - 1$ nodes that have received either $MessageP$ or $MessageS$. (no node receives both messages or any message multiple times). Therefore, the algorithm generates $2k - 1$ messages. \square

3.1.2 Match-making Algorithms A and B

We design two procedures **A** and **B** that can generate match making sets $\mathcal{N}_A(x, p, \ell)$ and $\mathcal{N}_B(y, p, \ell)$ given any two arbitrary nodes x and y in the star graph S_n . ℓ is the dimension of the star that the two sets of nodes are in. An ℓ value smaller than n means the match-making happens in a smaller star of dimension ℓ instead of the entire star S_n . p , $1 \leq p < \ell$ is an integer that determines the size of both $\mathcal{N}_A(x, p, \ell)$ and $\mathcal{N}_B(y, p, \ell)$. These procedures generate the match making sets by sending specific messages to the certain set of nodes. The algorithms are basically recursive invocations of the procedure $Mult$ described in the previous section. We present these algorithms as stand-alone procedures without any explicit reference to algorithm $Mult$.

Algorithm A

The procedure $Mult(k, M)$ transmits the value M from a source node k -substar S_k to k nodes of different $(k - 1)$ -substars in S_k . Using the hierarchical structure of the star graphs (Remark 2(3)), this procedure can be recursively invoked to distribute the value M to any level of substars. Algorithm $A(p, \ell, M)$ is such an algorithm which carries the value M from a node in ℓ -star S_ℓ to $(\ell!/p!)$ nodes of different p -substars in S_ℓ . The algorithm takes three parameters: ℓ is the dimension of star that this match-making is working on. p is an integer, $1 \leq p < \ell$, and M (as in procedure $Mult$) is some value.

The algorithm uses two messages: $MessagePA(p, k, M)$ and $MessageSA(p, k, M)$. The messages carry p and M from the parameters of the algorithm. The value of k , $p < k \leq \ell$ is determined at different nodes. Since the algorithm is executed in a distributed fashion, there is no global information other than the size n of the star graph. The nodes determine the next action by the message type and parameters, and the direction in which they receive any message. The details are shown in the following pseudo-code.

Procedure $A(p, \ell, M)$ at x

Initial Conditions:

Node x has the value M to be delivered.

Invocation of the Algorithm

Node x sends message $MessagePA(p, \ell, M)$ and $MessageSA(p, \ell, M)$ along direction 1.

Action at any node:

- Upon receiving $MessagePA(p, k, M)$ from the link (direction) dir :

```
for ( $i = \lceil \log_2 dir \rceil, i < \lceil \log_2 k \rceil, i++$ )
{
     $d = dir + 2^i$ ;
    if  $d < k$  then
        send  $MessagePA(p, k, M)$  along direction  $d$ ;
}
send message  $MessageSA(p, k, M)$  along direction  $k$ .
```

- Upon receiving $MessageSA(p, k, M)$:

```
Copy  $M$  into local memory.
if  $(k - 1 > p)$  then
    send  $MessagePA(p, k - 1, M)$  and  $MessageSA(p, k - 1, M)$  along direction 1
```

Remark 7

- The procedure A is very similar to $Mult$ except the handling of $MessageSA(p, k, M)$. When a node receives $MessageSA(p, k, M)$, it not only keeps a local copy of M , but also sends $MessagePA(p, k - 1, M)$ to itself if k is larger than p . By doing this, the node recursively invokes algorithm $Mult$ at a lower level until dimension p .
- Parameter ℓ of the algorithm does not have to be the same as n which is the size of the entire star that x is in. Since we do not use any direction larger than ℓ in this algorithm, all nodes involved in this algorithm should belong to $S^\ell(x)$ (Remark 2), which is an ℓ -dimensional star.

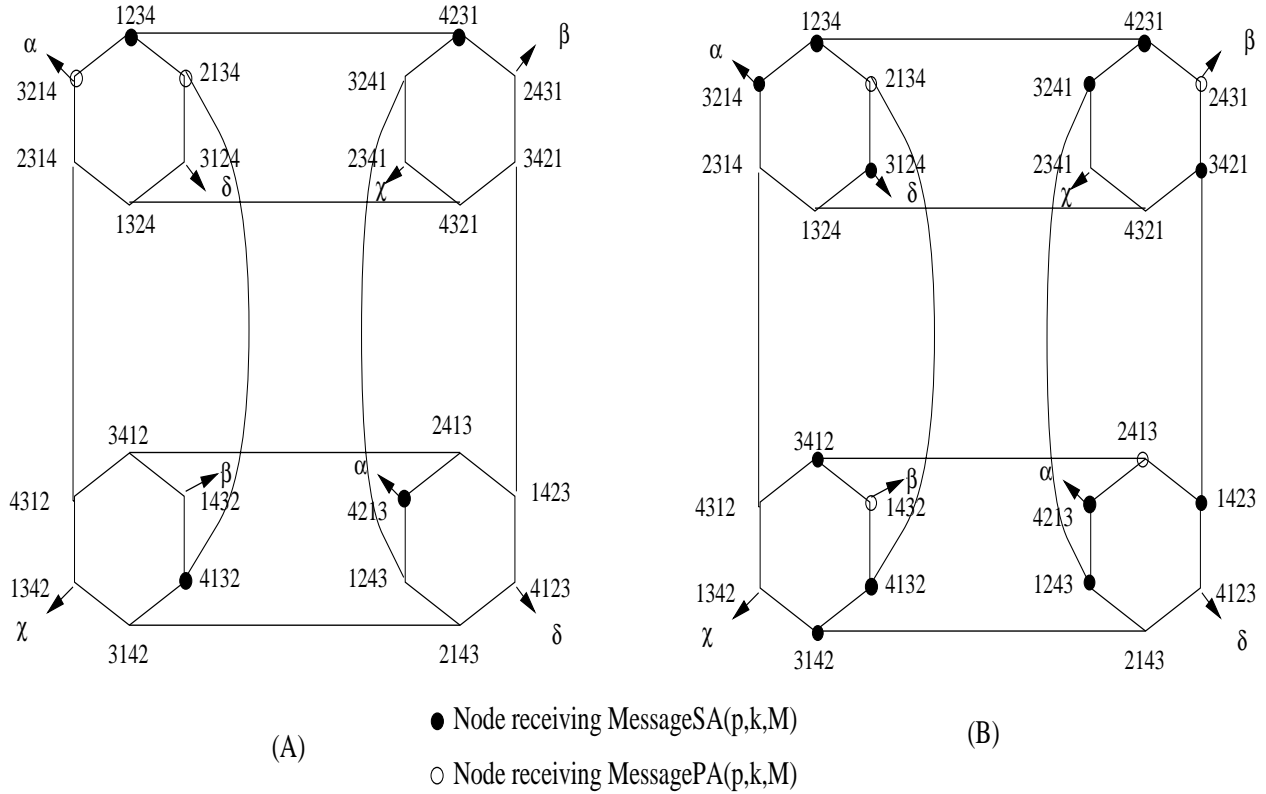


Figure 3: Execution of Algorithm $A(2, 4, M)$ at node 1234 in S_4 : (A) After the first stage, 4 nodes get $MessageSA(2, 4, M)$ (B) After the second stage, 12 nodes get $MessageSA(2, 3, M)$

Example 2 Figure 3 shows the execution of $A(2, 4, M)$ from node 1234 in an S_4 ($n = 4, p = 2$).

Initially, M is at node 1234 and it sends $MessagePA(2, 4, M)$ and $MessageSA(2, 4, M)$ along direction 1 to itself. Just as the execution of $Mult(4, M)$ at node 1234, $MessagePA(2, 4, M)$ will be propagated to node 3214 and 2134 ($k = 4$). Then each of the nodes that receive $MessagePA(2, 4, M)$ will send $MessageSA(2, 4, M)$ along direction

$k = 4$. So, node 4231, 4213, and 1432 will receive $MessageSA(2, 4, M)$. This finishes the first stage of the algorithm.

After the first stage, four nodes, 4231, 4132, 4213, and 1234, have received $MessageSA(2, 4, M)$ ($p = 2, k = 4$). These four nodes will send $MessagePA(2, 3, M)$ along direction 1 to themselves. This starts the second stage of the algorithm, which is similar to the execution of algorithm $Mult(3, M)$ at each of the four nodes. After the second stage, each of the nodes that received $MessageSA(2, 4, M)$ at the first stage will propagate $MessageSA(2, 3, M)$ to 3 different nodes. Hence, there will be totally 12 nodes that receive $MessageSA(2, 3, M)$, each from a different 2-dimensional star.

The algorithm stops after the second stage because the node receiving $MessageSA(2, 3, M)$ ($p = 2, k = 3$) will not generate more message due to the restraint $k - 1 > p$.

In this example, some nodes get multiple messages. For instance, node 1234 gets $MessagePA(2, 4, M)$ and $MessageSA(2, 4, M)$ at the first stage, and $MessagePA(2, 3, M)$ and $MessageSA(2, 3, M)$ at the second stage. Similarly, node 2134 receives $MessagePA(2, 4, M)$ at the first stage and $MessagePA(2, 3, M)$ at the second stage. This indicates that a queue must be maintained at each node to store the messages it receives and the messages are processed in the order of time it was received. A node can not respond to any message in the middle of the processing of another message. In this manner, the messages at different levels will not interfere with each other and the correctness of the algorithm is kept.

Lemma 4 When algorithm $A(p, \ell, M)$, $1 \leq p < \ell$, is executed at node x , $P(\ell, \ell - p)$ nodes receive $MessageSA(p, k, M)$, $p < k \leq \ell$; each of these nodes belong to a mutually disjoint p -dimensional substar $[* \cdots * t_1 t_2 \cdots t_{n-p}]$, where $t_1 t_2 \cdots t_{n-p}$ denotes all possible permutations of the set $\{x_1, x_2, \cdots, x_n\}$ and $P(\ell, p)$ is defined as $P(\ell, p) = \binom{\ell}{p} \times p!$, $1 \leq p < \ell$.

Proof: For a given value of p , there are $\ell - p$ stages. After the first stage, exactly ℓ nodes will receive $Message(p, \ell, M)$, each from ℓ mutually disjoint $(\ell - 1)$ -dimensional substars $[* \cdots * x_i]$, $1 \leq i \leq \ell$. Each of these nodes will send $MessagePA(p, \ell - 1, M)$ to itself and actually invoke procedure $Mult(\ell - 1, M)$ at level $\ell - 1$. At the end of the second stage, exactly $\ell(\ell - 1) = P(\ell, 2)$ nodes will receive $MessageSA(p, \ell - 1, M)$ each from $\ell(\ell - 1)$ mutually disjoint $(\ell - 2)$ -dimensional substars $[* \cdots * t_1 t_2]$ where $t_1 t_2$ denotes all possible permutations of the set $\{x_1, x_2, \cdots, x_\ell\}$. The algorithm ends after the message reaches all p -dimensional substars in S_ℓ . So extending the argument to the end of the $(\ell - p)$ -th stage, the claim follows. \square

Remark 8 Note that some nodes will receive the *MessageSA* and/or *MessagePA* multiple times. Each node maintains a queue to buffer the messages it receives in the middle of processing of other messages; it does not respond to other messages until the current message is completely processed. Thus, the messages from different level are handled without interference.

Corollary 1 An arbitrary node x can execute algorithm $A(\ell, p, M)$ to generate a set of size $P(\ell, \ell - p) = \frac{\ell!}{p!}$, i.e. the set of nodes which receive message $\text{MessageSA}(p, k, M)$, $p < k \leq \ell$. This set is denoted by $\mathcal{N}_A(x, p, \ell)$.

Lemma 5 Execution of the algorithm $A(p, \ell, M)$ for a given p , $p \geq 3$, at an arbitrary node x in S_ℓ , $\ell \geq 4$ requires at most $3P(\ell, \ell - p) = 3\frac{\ell!}{p!}$ messages and at most $\mathcal{O}(\ell \log \ell)$ time.

Proof: Note that the algorithm has $\ell - p$ stages. At each stage k , $p < k \leq \ell$, there are actually $(\ell!/k!)$ nodes executing algorithm $\text{Mult}(k, M)$. We know from Lemma 3 that $\text{Mult}(k, M)$ generates $(2k - 1)$ messages. Thus, the total number of messages generated by algorithm $A(p, \ell, M)$ is given by

$$M_A = \sum_{k=p+1}^{\ell} \frac{\ell!}{k!} \times (2k - 1) < \sum_{k=p+1}^{\ell} \frac{\ell!}{k!} \times 2k = \sum_{k=p+1}^{\ell} \frac{2\ell!}{(k-1)!} = 2 \left(\sum_{k=p}^{\ell-1} \frac{\ell!}{k!} \right)$$

For $\ell \geq 4$, and $p \geq 3$, we have

$$\begin{aligned} M_A &< 2\frac{\ell!}{p!} \left(1 + \frac{1}{(p+1)} + \frac{1}{(p+1)(p+2)} + \cdots + \frac{1}{(p+1)\cdots(\ell-1)} \right) \\ &< 2\frac{\ell!}{p!} \left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots \right) = \frac{2\ell!}{p!} \times \frac{4}{3} < \frac{3\ell!}{p!} \end{aligned}$$

Thus, the first claim follows. To compute the time taken by the algorithm $A(p, \ell, M)$, we note that the nodes executing the procedure $\text{Mult}(k, M)$ in the k -th iteration belong to different mutually disjoint substars and they execute concurrently. The time taken by procedure $\text{Mult}(k, M)$ is $\mathcal{O}(\log k)$ (Lemma 3). Thus the total time taken by the algorithm $A(p, \ell, M)$ is given by $\sum_{k=p+1}^{\ell} \mathcal{O}(\log k) = \mathcal{O}(\ell \log \ell)$. \square

Algorithm B

Consider an arbitrary node y in S_ℓ and an arbitrary integer p such that $1 \leq p \leq \ell$. Algorithm $B(p, \ell, M)$ carries the value M to all nodes in $S^p(x)$ that is $[* \cdots * x_{p+1} \cdots x_\ell]$. Like algorithm A , it uses an integer p as an input parameter to control the size of the set that receive the message. There are two messages used in this algorithm: $MessagePB(k, M)$ and $MessageSB(k, M)$. Since this algorithm stops at 1-dimensional star, there is no need to carry parameter p in the message. As in algorithm A , M is a value of any type and k , $1 < k \leq p$ is determined at run time at each node. The basic idea is the same as in designing the algorithm A . Algorithm $B(p, \ell, M)$ is to repeatedly call the procedure $Mult(k, M)$ with different parameters until $k = 2$. Only the nodes receive $MessageSB(k, M)$ will invoke the $Mult$ procedure subsequently with a lesser parameter. Nodes receiving the message $MessagePB(k, M)$ do not invoke $Mult$.

Procedure $B(p, \ell, M)$ at y

Initial Conditions:

Node y has message M to be delivered.

Invocation of the Algorithm

Node y sends message $MessagePB(p, M)$ and $MessageSB(p, M)$ along direction 1.

Action at any node:

- **Upon receiving $MessagePB(k, M)$ from the link (direction) dir :**
for ($i = \lceil \log_2 dir \rceil, i < \lceil \log_2 k \rceil, i++$)
{
 $d = dir + 2^i$;
 if $d < k$ then
 send $MessagePB(k, M)$ along direction d ;
 }
send message $MessageSB(k, M)$ along direction k .

- **Upon receiving $MessageSB(k, M)$:**
Copy M into local memory.
if ($k - 1 > 1$) then
 send $MessagePB(k - 1, M)$ and $MessageSB(k - 1, M)$ along direction 1

Remark 9

- *The difference between algorithm A and B is the start point and stop condition. Algorithm A starts at level ℓ and stops at level p , while algorithm B starts at level p and stops at level 1.*

- *Parameter ℓ is included in algorithm B in order for it to look consistent with algorithm A . In fact, the execution of*

algorithm B is independent of the value ℓ . All nodes involved in this algorithm belong to $S^p(y)$ (Remark 2), which is not relevant to the value of ℓ . Also, ℓ does not have to be the same as n which is the size of entire star.

Lemma 6 When algorithm $B(p, \ell, M)$, $1 < p \leq \ell$, is executed at node $y = y_1 y_2 \cdots y_\ell y_{\ell+1} \cdots y_n$, a total of $p!$ nodes in the p -dimensional substar $[* \cdots * y_{p+1} \cdots y_\ell y_{\ell+1} \cdots y_n]$ receives $MessageSB(k, M)$.

Proof : The proof follows directly from the observation that the algorithm $B(p, \ell, M)$ is essentially a broadcast algorithm where the source node y broadcasts the message within the p -dimensional substar $[* \cdots * y_{p+1} \cdots y_\ell y_{\ell+1} \cdots y_n]$ using the links (directions) 2 through p at each node. The algorithm has $p - 1$ stages. After the first stage, p nodes receive message $MessageSB(p, M)$, each node belonging to the substars $[* \cdots * t_i y_{p+1} \cdots y_\ell y_{\ell+1} \cdots y_n]$ where $t_i \in \{y_1, y_2, \dots, y_p\}$; after the second stage, $p(p - 1)$ nodes receive message $MessageSB(p - 1, M)$ each from the substars $[* \cdots * t_1 t_2 y_{p+1} \cdots y_n]$ where $t_1 t_2$ denotes all possible permutations of 2 symbols from the set $\{y_1, y_2, \dots, y_p\}$; and so on. □

Remark 10 The same observation, as in designing algorithm A , applies; each node maintains a queue to buffer the messages it receives in the middle of processing of other messages; it does not respond to the messages until the current processing is complete.

Corollary 2 An arbitrary node y can execute algorithm $B(p, \ell, M)$ to generate a set of size $p!$, i.e. the set of nodes which receive message $MessageSB(p, M)$. This set is denoted by $\mathcal{N}_B(y, p, \ell)$.

Lemma 7 Execution of the algorithm $B(p, \ell, M)$ for a given p , $p \geq 3$, at an arbitrary node y in S_ℓ , $\ell \geq 4$ requires at most $3 \times p!$ messages and takes at most $\mathcal{O}(p \log p)$ time.

Proof : Note that the algorithm has $p - 1$ stages, say for $1 < k \leq p$, where k denotes the index of the stage. In the k -th stage, $(1 < k \leq p)$, $\frac{p!}{k!}$ nodes concurrently execute the procedure $Mult(k, M)$ each generating $2k - 1$ messages and taking $\mathcal{O}(\log k)$ time. Thus, the total number of messages generated by algorithm $B(p, \ell, M)$ is given by

$$M_B = \sum_{k=2}^p \frac{p!}{k!} (2k - 1) < \left(\frac{3}{2} + \frac{5}{6} \right) \times p! + \sum_{k=4}^p \frac{p!}{k!} 2k$$

$$\begin{aligned}
&< \left(\frac{3}{2} + \frac{5}{6}\right) \times p! + \sum_{k=4}^p \frac{1}{(k-1)!} < \frac{7}{3}p! + \frac{p!}{3} \times \left(1 + \frac{1}{4} + \frac{1}{4^2} + \dots\right) \\
&= \left(\frac{7}{3} + \frac{4}{9}\right) \times p! < 3 \times p!
\end{aligned}$$

Also, since each of the loop takes $\mathcal{O}(\log k)$ time, the total algorithm takes $\mathcal{O}(\sum_{k=2}^p \log k) = \mathcal{O}(p \log p)$ time. \square

Remark 11

1. When any arbitrary node x in S_ℓ executes algorithm $A(p, \ell, M)$ for any given value of p , it generates the set $\mathcal{N}_A(x, p, \ell)$; the size of this set as well as the number of messages required to generate this set decreases as p increases.
2. When any arbitrary node y in S_ℓ executes algorithm $B(p, \ell, M)$ for any given value of p , it generates the set $\mathcal{N}_B(y, p, \ell)$; size of this set as well as the number of messages required to generate this set increases as p increases.

Theorem 1 For any two arbitrary nodes x and y in a star graph S_ℓ and an integer $p, 1 \leq p \leq \ell$, there is exactly one node in the intersection of $\mathcal{N}_A(x, p, \ell)$ and $\mathcal{N}_B(y, p, \ell)$.

Proof : The star graph S_ℓ can be decomposed into $\frac{\ell!}{p!}$ mutually disjoint substars of dimension p . Since $\mathcal{N}_A(x, p, \ell)$ includes exactly one node from each of these substars and $\mathcal{N}_B(y, p, \ell)$ includes all the nodes of one of these $\frac{\ell!}{p!}$ substars (see Lemmas 4 and 6), the intersection of $\mathcal{N}_A(x, p, \ell)$ and $\mathcal{N}_B(y, p, \ell)$, generated with the same integer p , has precisely one node. \square

Example 3 In this example, we show how match-making works in a S_5 ($n = 5$). We select two arbitrary nodes $x = 12345$ and $y = 54321$, and invoke algorithm $A(3, 5, M)$ at node x and $B(3, 5, M')$ at node y .

At the first stage of algorithm $A(3, 5, M)$, 5 nodes will receive $MessageSA(3, 5, M)$. These are 12345, 51324, 52143, 51342, and 52341. Each of them will propagate $MessageSA(3, 4, M)$ to 4 different nodes. So, after the second stage of algorithm $A(3, 5, M)$, there will be 20 nodes that receive $MessageSA(3, 4, M)$. These nodes are 12345, 42315, 42135, 41325 in 4-substar $[* \dots * 5]$, 51324, 21354, 21534, 25314 in 4-substar $[* \dots * 4]$, 52143, 42153, 42513, 45123 in 4-

substar [$* \cdots * 3$], 51342, 41352, 41532, 45312 in 4-*substar* [$* \cdots * 2$], 52341, 42351, 42531, 45321 in 4-*substar* [$* \cdots * 1$].

Algorithm $A(5, 3, M)$ stops after the second stage, so the set $\mathcal{N}_A(x, 3, 5)$ consists of those 20 nodes.

Algorithm $B(3, 5, M)$ starts at node $y = 54321$. At the first stage, $MessageSB(3, M)$ is sent to 3 nodes, 54321, 34521, and 35421. At the second stage, these three nodes send $MessageSB(2, M)$ to 6 nodes, including these three nodes 54321, 34521, 35421, and three additional nodes 45321, 43521, 53421. Algorithm $B(3, 5, M)$ then stops and these 6 nodes constitutes the set $\mathcal{N}_B(y, 3, 5)$.

Comparing $\mathcal{N}_A(x, 3, 5)$ and $\mathcal{N}_B(y, 3, 5)$, we see there is exactly one common node 45321 between $\mathcal{N}_A(x, 3, 5)$ and $\mathcal{N}_B(y, 3, 5)$.

3.2 The Election Algorithm

3.2.1 Tournament Scheme

We use the match making technique developed in the previous section to design our leader election algorithm that uses a tournament scheme to elect the leader. The match making sets $\mathcal{N}_A(x, p, \ell)$ and $\mathcal{N}_B(y, p, \ell)$ for two arbitrary nodes x and y in the star graph S_n and any given positive integer ℓ will be used to communicate messages between leaders during the tournament scheme. The tuning parameter ℓ determines the message complexity of both the algorithms A and B; we need to choose a specific value of ℓ to minimize the total number of messages in the election algorithm. We introduce the function δ_n in the following to compute this optimum value of ℓ for a given n .

Definition 3 For any integer n (dimension of star S_n), an integer δ_n is defined such that $2(\delta_n!) \leq \sqrt{6n!}$ and $2(\delta_n + 1)! > \sqrt{6n!}$.

Remark 12 It is quite easy to see that for any n , $n \geq 2$, the integer δ_n is uniquely defined. Given any integer n , δ_n can be computed very easily; compute $\sqrt{6n!}$, get the highest integer i such that $i! \leq \sqrt{6n!}/2$. Table 1 shows some typical values.

Lemma 8 For $n \geq 4$, (i) $\delta_n \geq 3$, (ii) $\delta_n + 1 \leq n$, and (iii) $\frac{1}{(\delta_n + 1)!} < \sqrt{\frac{2}{3n!}}$.

n	δ	$\sqrt{6n!}$	$2(\delta!)$	$2((\delta + 1)!)$
3	2	6	4	12
4	3	12	12	48
5	3	26.8	12	48

Table 1: Values of δ_n for different n

Proof: The claims readily follows from Definition 3. □

We start with a star graph S_1 of dimension 1. Since S_1 has only one node, leader election is easy; the only node becomes the leader. Now consider a star graph S_n . If we already have n leaders of all the component substars S_{n-1} , we can use a traditional tournament scheme to elect the unique leader for the entire S_n .

Definition 4 In a star graph S_n of dimension n , an ℓ -leader x is the leader of $\ell!$ nodes in the ℓ -substar (of dimension ℓ) $[* \cdots * x_{\ell+1} \cdots x_n]$.

Remark 13 An ℓ -leader knows its region (the ℓ -substar) by its links 2 to ℓ ; it does not need to know the node permutation to know the region (Definition 2 and Remark 3). For example, if node 1234 is a 3-leader in a star graph S_4 , it is the leader of 8 nodes: 1234, 1324, 2134, 2314, 3124, 3214. If the node 31245 is the 2-leader in S_5 , it is the leader of 2 nodes: 31245 and 13245.

Consider any $(\ell + 1)$ -substar $[* \cdots * x_{\ell+2} \cdots x_n]$, $0 \leq \ell \leq n - 1$. It consists of $(\ell + 1)$ mutually disjoint ℓ -substars $[* \cdots * x_i x_{\ell+2} \cdots x_n]$, where $x_i \in \{x_1, x_2, \dots, x_{\ell+1}\}$. If each of these ℓ -substars have already elected their respective ℓ -leaders, they will participate in the tournament to elect the $(\ell + 1)$ -leader of the $(\ell + 1)$ -substar $[* \cdots * x_{\ell+2} \cdots x_n]$. To set up the tournament, we need to define the image nodes of a leader.

Definition 5 For an ℓ -leader x in S_n , $image_set^\ell(x)$ is defined to be the set $\{image_k^\ell(x), 1 \leq k < \ell\}$ where the $image_k^\ell(x)$ is the node reached from node x by traversing the link k and then link $(\ell + 1)$. Note that when $k = 1$, traversing link 1 does not generate a new node. (Images are defined only for leader nodes).

Example 4 Let 123456 be a 3-leader in S_6 ($\ell = 3$). The set $image_set^3(123456)$ has 3 nodes $\{423156, 413256, 421356\}$. The first image ($k = 1$) is reached from 123456 by traversing link 4. The second image 413256 ($k = 2$) is reached from

123456 by traversing link 2 and then link 4 while the third image 421356 ($k = 3$) is reached from 123456 by traversing link 3 and then link 4.

Lemma 9 Consider any $(\ell + 1)$ -substar $[* \cdots * x_{\ell+2} \cdots x_n]$, $0 \leq \ell \leq n - 1$ corresponding to an arbitrary node $x = (x_1 \cdots x_n)$. It consists of $(\ell + 1)$ mutually disjoint ℓ -substars $[* \cdots * x_i * x_{\ell+2} \cdots x_n]$, where $x_i \in \{x_1, x_2, \dots, x_{\ell+1}\}$, each with a leader μ_i . For any i , $1 \leq i \leq \ell + 1$, the set $image_set^\ell(\mu_i)$ has exactly one node in each of the $(\ell + 1)$ mutually disjoint ℓ -substars, except the one it is leader of.

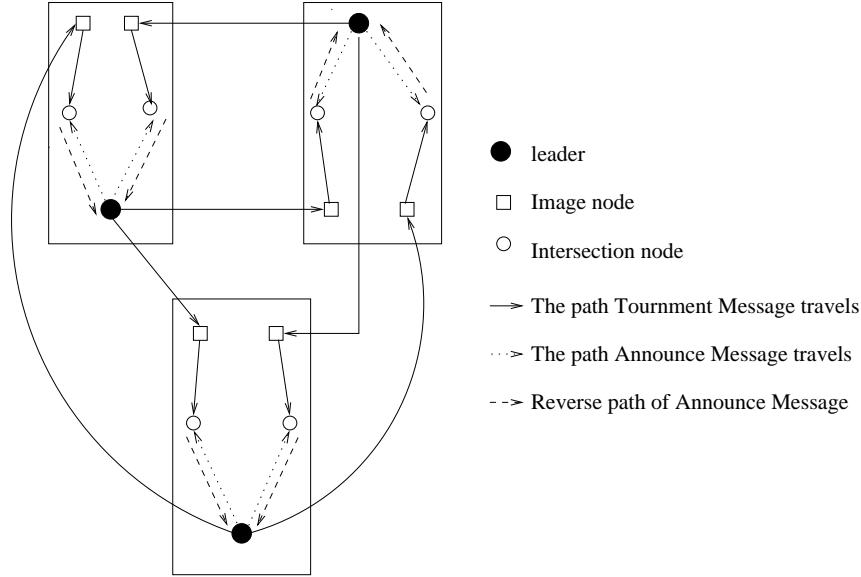


Figure 4: Message Forwarding in Leader Election Algorithm

Proof : Assume one of the ℓ -leaders is $x = x_1 \dots x_n$. From the definition of image nodes, $image_k^\ell(x)$ has x_k at the $(\ell + 1)$ th position in its label. Since $1 \leq k < \ell$, these symbols are all distinct and different from the one at the same position in x . Since all these nodes have the same permutation between position $(\ell + 2)$ and n , these nodes belong to different substars of dimension ℓ , but are in the same substar of dimension $\ell + 1$. We can also easily see that all ℓ -leaders in this substar of dimension $\ell + 1$ and their images are in the same substar of dimension $\ell + 1$. \square

We now present the algorithm *Tournament* (ℓ) that computes the $(\ell + 1)$ -leader of the $(\ell + 1)$ -substar from among the ℓ -leaders of the component ℓ -substars.

Algorithm *Tournament* (ℓ)

- **Step 1:** Each ℓ -leader μ_i , $1 \leq i \leq \ell + 1$, announces its ℓ -leadership by sending a message **Announce** (μ_i, ℓ) to all nodes in $\mathcal{N}_A(\mu_i, \delta_\ell, \ell)$ (each ℓ -leader μ_i invokes algorithm $A(\delta_\ell, \ell, \text{Announce}(\mu_i, \ell))$ from itself).
- **Step 2:** Each ℓ -leader μ_i , $1 \leq i \leq \ell + 1$ sends a message **Tournament** (μ_i, ℓ) to all its image nodes $image_j^\ell(\mu_i)$, $1 \leq j \leq \ell$
- **Step 3:** After receiving message **Tournament** (μ_i, ℓ) , the image node $image_j^\ell(\mu_i)$, $1 \leq i \leq \ell + 1$, $1 \leq j \leq \ell$, broadcasts the message to all nodes in $\mathcal{N}_B(image_j^\ell(\mu_i), \delta_\ell, \ell)$ (each image node invokes algorithm $B(\delta_\ell, \ell, \text{Tournament}(\mu_i, \ell))$ from itself).
- **Step 4:** Consider the i th ℓ -substar, $1 \leq i \leq \ell + 1$: this substar contains its own leader μ_i and ℓ image nodes of ℓ -leaders from ℓ other ℓ -substars. Consider any pair $(\mu_i, image_i^\ell(\mu_j))$, where $1 \leq j \leq \ell + 1$ excluding $j = i$. For each pair, there is exactly one node (Theorem 1), say p_k , that receives both **Tournament** (μ_k, ℓ) and **Announce** (μ_i, ℓ) , where μ_i is the ℓ -leader of the i -th substar and μ_j , $1 \leq k \leq \ell$, $i \neq j$ is the leader of the j -th substar. After receiving both message, p_j sends the message **Tournament** (μ_j, ℓ) to node μ_i (using a path obtained by reversing the path that message **Announce** (μ_i, ℓ) uses; information about the path can be easily maintained by providing a parent pointer at nodes).
- **Step 5:** Upon receiving the tournament messages from all other ℓ -leaders, each node μ_i , $1 \leq i \leq \ell + 1$, will know all other ℓ -leaders; one of them will decide to be the $(\ell + 1)$ -leader and others will decide not to be the $(\ell + 1)$ -leader.

Remark 14 *Figure 4 shows a schematic view of the Tournament algorithm. each smaller substar has a leader and one image each of all other leaders. There is precisely one intersection node for each pair of leader and image node. An Announce message is sent from the leader to its local substar, hence passes the intersection node. The Tournament message is sent from a leader to its images in other substars. The image nodes relay this message to intersection nodes. After intersection nodes get the Tournament message, it sends this message to the leader via the reverse path that Announce message travels. Eventually, each leader receives the Tournament message from all other leaders at the same level. Then, a next level leader is elected which will participate in the leader election at the next higher level until the leader of the entire S_n is elected.*

3.2.2 Message Complexity of Leader Election

We assume that $n \geq 4$ without any loss of generality, since for star graphs S_n of dimensions 1, 2 and 3 leader election can be done by the simple broadcast scheme using constant number of messages. Let $M(\ell)$ denote the number of messages generated by the algorithm to elect a ℓ -leader in a ℓ -substar and $T(\ell)$ denotes the number of messages needed to elect the $(\ell + 1)$ -leader from among the ℓ -leaders using the $Tournament(\ell)$ algorithm of the previous section. Note that the number of nodes in S_n is $N = n!$ and the $Tournament(\ell)$ algorithm uses the characteristic integer δ_ℓ such that $2(\delta_\ell!) \leq \sqrt{6\ell!}$ and $2(\delta_\ell + 1)! > \sqrt{6\ell!}$ (Definition 3).

Lemma 10 For any ℓ , $3 \leq \ell \leq n$, $T(\ell) < 7.5 \times \ell(\ell + 1)\sqrt{\ell!}$.

Proof: We add up the messages needed in each of the 5 steps in the $Tournament(\ell)$ algorithm.

- Step 1 needs less than $(\ell + 1) \times 3 \frac{\ell!}{\delta_\ell!}$ ($= T^1$) messages since there are $\ell + 1$ leaders execute algorithm A in an ℓ -substar and each needs less than $3 \frac{\ell!}{\delta_\ell!}$ messages (Lemma 5).
- Step 2 takes less than $2\ell(\ell + 1)$ ($= T^2$) messages, since there are $\ell(\ell + 1)$ image nodes and the distance from leaders to their image nodes is at most 2 (Definition 5).
- Step 3 requires less than $2\ell(\ell + 1) \times \delta_\ell!$ ($= T^3$) messages since there are $\ell(\ell + 1)$ image nodes and each of them executes algorithm B in a δ_ℓ -substar each requiring $3\delta_\ell!$ messages (Lemma 7).
- In step 4, $\ell(\ell + 1)$ intersection nodes are generated (from the $(\ell + 1)$ leaders and $\ell(\ell + 1)$ image nodes). Each of these intersection nodes relay the message along reverse path from algorithm A . The maximum length of each of these paths is less than $2(\ell - \delta_\ell)$; so, the total number of messages needed in step 4 is less than $2\ell(\ell + 1)(\ell - \delta_\ell)$ ($= T^4$).
- Step 5 does not generate any message.

Using Lemma 8 and the fact that $n \geq 4$, it is easy to see that $T^1 < \ell(\ell + 1)\sqrt{6\ell!}$, $T^2 + T^4 < \ell(\ell + 1)\sqrt{\ell!}$, and $T^3 < 1.5 \times \ell(\ell + 1)\sqrt{6\ell!}$. Adding up, we get

$$T(\ell) < (1 + 2.45 + 2.45 + 1.23)\ell(\ell + 1)\sqrt{\ell!} < 7.5\ell(\ell + 1)\sqrt{\ell!}$$

□

Theorem 2 For a star graph S_n , $n \geq 4$, the total number of messages needed to elect a leader $M(n) < 32N$, where $N = n!$ denotes the number of nodes in S_n .

Proof: To elect an ℓ -leader in a ℓ -substar, we need $M(\ell)$ messages; to elect the $(\ell + 1)$ -leader from among the ℓ -leaders ($(\ell + 1)$ many of them), we need $T(\ell)$ messages. Thus,

$$M(\ell + 1) = (\ell + 1)M(\ell) + T(\ell)$$

Using Lemma 10, we have

$$M(\ell + 1) < (\ell + 1)M(\ell) + 7.5\ell(\ell + 1)\sqrt{\ell!}$$

Introducing $F(\ell) = \frac{M(\ell)}{\ell!}$, we get,

$$F(\ell + 1) < F(\ell) + \frac{7.5\ell(\ell + 1)\sqrt{\ell!}}{(\ell + 1)!} = F(\ell) + \frac{7.5\ell}{\sqrt{\ell!}}$$

Consider the initial conditions for the above recurrence relation. For S_1 , $M(1) = 0$ and $F(1) = 0$; for S_2 and S_3 , if we use simple broadcasting from all leaders, we easily see that $M(2) = 2$ and $F(2) = 1$, and $M(3) = 21$ and $F(3) < 4$.

Thus,

$$\begin{aligned} F(n) &< 5 + \sum_{\ell=4}^n \frac{7.5(\ell - 1)}{\sqrt{(\ell - 1)!}} < 5 + 7.5 \times \left\{ \frac{3}{\sqrt{6}} + \frac{4}{\sqrt{24}} + \sum_{\ell=5}^{\infty} \frac{\ell}{\sqrt{\ell!}} \right\} \\ &< 5 + 7.5 \times \left\{ 1.22 + 0.82 + \sum_{i=1}^{\infty} \frac{1}{2^i} \right\} < 5 + 7.5 \times 3.5 < 32 \end{aligned}$$

So, it follows that $M(n) < 32n!$.

□

Remark 15 It should be noted that the above theorem provides a very relaxed upper bound on the number of messages needed to elect a leader in S_n ; the actual number of messages would be much smaller.

3.2.3 Time Complexity of Leader Election

We assume, as in [6], that all the processes initiate the algorithm independently and that the last process starts at time say t_x . We want to find an upper bound on the time t_n when the leader is elected in the entire star S_n .

Theorem 3 *The algorithm elects a leader in S_n , $n \geq 4$, within $\mathcal{O}(n^2 \log n)$ time.*

Proof : Assume that all ℓ -leaders have been elected at time t_ℓ . At time $t_\ell + \ell \log \ell$ the announcement of leadership is completed among the nodes in all the \mathcal{N}_A sets (Step 1). At time $t_\ell + 2$ the image nodes have received the tournament message (Step 2) and the broadcast of the tournament is completed in time $t_\ell + 2 + \delta_\ell \log \delta_\ell$ (Step 3). Hence the forwarding of the tournament messages (Step 4) starts latest at time $t_\ell + \max\{\ell \log \ell, 2 + \delta_\ell \log \delta_\ell\}$ and the forwarding takes at most $2\delta_\ell$ time. Hence, the ℓ -leaders receives the message latest by $t_\ell + n \log n + 2 + 2\delta_\ell$. Assuming that the last process starts at time t_x , the time for leader election in S_n is bounded by

$$t_n = t_x + \sum_{\ell=1}^{n-1} (\ell \log \ell + 2 + 2\delta_\ell) < n^2 \log n + 2n + n(n-1) = \mathcal{O}(n^2 \log n)$$

□

Remark 16 *The number of nodes in S_n is given by $N = n!$. Since $\log N \approx n \log n$, the time for election is less than $\mathcal{O}(\log^2 N)$; thus the algorithm performs better than the leader election algorithm [6] for an oriented hypercube.*

4 Conclusion

We have proposed an election algorithm in oriented star graph networks that has a message complexity linear in the number of nodes in the network. It would be interesting to study election algorithms in unoriented star graphs and to find if orientation of links helps in star graphs helps or we can come up with a linear election algorithm in star graphs as was reported for hypercubes in [9].

5 Acknowledgements

The authors are grateful to the detailed comments from the anonymous reviewers that helped to greatly improve the presentation.

References

- [1] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5:67–77, 1983.
- [2] N. Santoro. Sense of direction, topological awareness and communication. *ACM SIGACT News*, 16:50–56, 1984.
- [3] H. L. Bodlaender. New lower bound techniques for distributed leader finding and other problems on rings of processors. *Theoretical Computer Science*, 81:237–256, 1991.
- [4] M. C. Loui, T. A. Matsuhita, and D. B. West. Election in a complete network with a sense of direction. *Information Processing Letters*, 22:185–187, 1986.
- [5] E. Korach, S. Moran, and S. Zaks. Tight upper and lower bounds for some distributed algorithms for a complete network of processors. In *Proceedings of Symposium on Principles of Distributed Computing*, pages 199–207, 1984.
- [6] G. Tel. Linear election in oriented hypercubes. Technical Report RUU-CS-93-39, Computer Science, Utrecht University, 1993.
- [7] G. Tel. Linear election in hypercubes. *Parallel Processing Letters*, 5:357–366, 1995.
- [8] B. Mans. Optimal distributed algorithms in unlabeled tori and chordal rings. Technical Report TR 96-04, James Cook University, Queensland, Australia, 1996.
- [9] K. Diks, S. Dobrev, E. Kranakis, A. Pelc, and P. Ruzicka. Broadcasting in unlabeled hypercubes with linear number of messages. *Information Processing Letters*, 66:181–186, 1998.
- [10] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Trans. Comput.*, 10(9):878–888, September 1999.

- [11] S. B. Akers and B. Krishnamurthy. The star graph: an attractive alternative to n-cube. In *Proceedings of International Conference on Parallel Processing (ICPP-87)*, pages 393–400, St. Charles, Illinois, August 1987.
- [12] S. B. Akers and B. Krishnamurthy. The fault tolerance of star graphs. In *Proceedings of 2nd International Conference on Supercomputing*, volume III, pages 270–276, San Francisco, May 1987.
- [13] K. Day and A. Tripathi. A comparative study of topological properties of hypercubes and star graphs. Technical Report TR 91–10, Computer Science Department, University of Minnesota, Minneapolis, MN, May 1991.
- [14] A. Kavianpour. System level diagnosis strategies for n-star multiprocessor systems. In *Proceedings of International Conference on Parallel Processing*, pages 297–300, 1993.
- [15] S. Lakshmivarahan, J. S. Jwo, and S. K. Dhall. Symmetry in interconnection networks based on Cayley graphs of permutation groups: a survey. *Parallel Computing*, 19:361–407, 1993.
- [16] S. Latifi. On fault diameter of star graphs. *Information Processing Letters*, 46:143–150, June 1993.
- [17] S. Sur and P. K. Srimani. Topological properties of star graphs. *Computers & Mathematics with Applications*, 25(12):87–98, 1992.
- [18] S. Sur and P. K. Srimani. Super star: a new optimally fault tolerant network architecture. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS-11)*, pages 590–597, Texas, 1991.
- [19] D. A. Holton and J. Sheehan. *The Petersen Graph*. Cambridge University Press, Cambridge, England, 1993.
- [20] S. J. Mullender and P. M. B. Vitanyi. Distributed match making. *Algorithmica*, 3:367–391, 1988.