Leadership Protocol for S-Nets

Thomas C. Henderson
School of Computing
University of Utah
Salt Lake City, UT, 84112 USA

22 February 2001

Abstract

*Smart Sensor Networks are collections of non-mobile devices (*S-elements*) which can compute, communicate and sense the environment; they must be able to create local groups of devices (*S-cliques*). We propose here a protocol to solve the leadership problem for* S-Nets*. We sketch the correctness of this protocol in terms of an asynchronous network model.*

1. Introduction

At one extreme, mobile robots can be provided with a wealth of on-board sensing, communication and computational resources [1, 2]; at the other extreme, robots with fewer on-board resources can perform their tasks in the context of a large number of stationary devices distributed throughout the task environment [3]. We call the latter approach the *Smart Sensor Network*, or the *S-Net*. We have performed simulation experiments using software (C and Matlab), and the performance of robot tasks with and without the presence of an *S-Net* (i.e., a set of distributed sensor devices) has been evaluated in terms of various measures. See [4, 5] for a more detailed account.

This approach can be exploited widely and across several scales of application; e.g., fire fighting robots. If mobile robots are used to fight forest fires, there may be several hot spots to extinguish or control. If sensor devices can be distributed in the environment, then their values and gradients can be used to direct the behavior of fire fighting robots and to transport fire extinguishing materials from a depot to the closest fire source. During this movement to and from the fire, collision avoidance algorithms can be employed. Sometimes coordinated activities are necessary and communication models are also important.

In our previous work, we provided models for various components of the study: (1) mobile robots with on-board sensors, (2) communication, (3) the *S-Net* (includes computation, sensing and communication), and (4) the simulation environment. We have developed algorithms in the simulation environment for the *S-Net* which perform cooperative computation and provide global information about the environment. Local and global frames are defined and created. A method for the production of global patterns using reaction-diffusion equations has been described and its relation to multi-robot cooperation demonstrated. In addition, we have shown how to compute shortest paths in the *S-Net* using level set techniques [7].

The results of our simulation experiments help us better understand the benefits and drawbacks of the *S-Net*. We have shown that for behaviors of one mobile robot going to a temperature source, and multiple mobile robots surrounding a temperature source, in the ideal situation (which means no noise), the *S-Net* takes more time and distance. But when noise is added in, which is more realistic, the *S-Net* system is more robust than the non-*S-Net* system. For the task of multiple mobile robots going back and forth to a temperature source, there are thresholds above which the *S-Net* system out-performs the non-*S-Net* system.

In this paper, we begin our conversion of the uni-processor *S-Net* simulation code to embedded processor algorithms. The set of necessary distributed algorithms includes:

- *S-Clique* formation with leader
- Coordinate frame calculation
- Gradient calculation
- Reaction-Diffusion computation
- Shortest Paths using level sets.

The first necessary algorithm is a solution to the leadership problem.

The Leadership Problem: Each *S-element* has a unique integer ID (UID) and a fixed geographic location; *S-elements* have a restricted broadcast range which defines a connectivity graph. The *S-elements* are to be grouped into subgraphs, called *S-cliques*, such that each *S-clique* has a leader, and the leader of each *S-clique* has the lowest ID of all members of the *S-clique*.

In this paper, we describe an algorithm to solve the *S-clique* leadership problem. For a good introduction to distributed algorithms, including solutions to variations of the leadership problem and correctness proofs, see [6].

2. Leadership Protocol

An *S-net* system will be represented as an undirected graph where each node is an *S-element*. The graph is undirected because if one *S-element* is within range to receive communication from another, then it works both ways. Each node can be viewed as a distinct process; in our implementation, they are Unix processes, but ultimately, they will be distinct hardware devices.

Formal definitions can be given for the nodes, and this involves defining states, including start states, message generating functions, and state transitions. However, only an informal description is given here. Such a description will include *broadcast()* and *receive()* primitive functions with their associated messages. A *broadcast* sends a message to all *S-elements* within range. Proof methods typically involve either invariant assertions and a demonstration that they hold, or simulations.

A simple example of a leadership algorithm is the LCR algorithm which provides a basic solution to the leadership problem in a synchronous ring network; it involves each process sending its UID to its neighbors; when a process receives an UID, it will throw it away if it is lesser than its own, resend it to its neighbors if it is larger than its own, and declare itself the leader if it is equal to its own. Our solution is related to this idea, although not the same.

The *S-Net* leadership basic algorithm (**SNL**) is executed by each node, and is as follows:

Algorithm **SNL**:

Step 1. Broadcast own ID

Step 2. Receive from other nodes, create neighbors list

Step 3.

```
Create remaining nodes list (initially, neighbors)
while not done
   if node's own ID is lowest in remaining nodes list,
```

then node is leader
broadcast cliques (self and neighbors)
done
else receive broadcast list
if in list
node is not a leader
re-broadcast list
done
else remove list from remaining

Note that we assume that enough time is given to steps 1 and 2 so that each node can complete the step correctly. This will most likely be implemented as a fixed time delay in an embedded system. Also, we assume that there are communications protocols that are reliable enough to transmit the messages without loss of information.

3. Correctness

We outline an informal argument for the correctness of algorithm **SNL**. Let $\mathcal{U} = \{1, 2, \ldots, uid_{max}\}$. The message alphabet $\mathcal{M}$ is the power set of $\mathcal{U}$, i.e., $\mathcal{P}(\mathcal{U})$.

The state of each node includes:

- my_UID_i: node i's unique UID (i.e., $my_UID_i = i$)
- *broadcast*: a message in $\mathcal{M}$ or *null*, initially *null*
- *leader*: a Boolean, indicating whether the node is a leader, intially *false*
- *resolved*: a Boolean, indicating whether the node has resolved as either a leader or not, initially *false*

Data structures used include:

- *neighbors*: list of *S-element* neighbors, initially *null*
- *remaining*: list of *S-element* neighbors still unresolved, initially *null*

The start state for each node i is that initial set of values indicated above. For each node, the following messages are possible:

- *self*: consists of my_UID_i
- *clique*: list of UID's that form a clique

The transition function for **SNL** is defined as:

```
% Step 1 of SNL
while (timer1 > 0)
  broadcast self;
endwhile

% Step 2 of SNL
while (timer2 > 0)
```

```
  add_to_neighbors(receive)
endwhile

remaining = neighbors;
% Step 3 of SNL
while (not resolved)
  % Step 3.1
  if (my_UID(i) < min(remaining))
    leader = true;
    resolved = true;
    broadcast(my_UID(i), remaining);
  endif

  list = receive;
  % Step 3.2
  if (my_UID(i) in list)
    leader = false;
    resolved = true;
    broadcast(list);
  endif
  remaining = remaining - list;
endwhile
```

Note that the broadcast in (3.2) has to take place so that a node i not in the clique, but neighboring a node j in the clique, can know that node j is resolved; this is necessary since the leader will not reach the non-clique nodes that neighbor clique nodes (i.e., the broadcast from the leader node will not reach node i).

The algorithm is supposed to achieve:

- (i) $leader = true$

  for any node that has the lowest UID of it and its unresolved neighbors.

- (ii) $leader = false$

  for any node that neighbors a leader.

- (iii) $resolved = true$

  for every node.

Case (i)

Suppose that node i has the lowest UID of it and any of its neighbors. Then when it finishes Step (2),

$$remaining = (nei_{i_1}_UID, \ldots, nei_{i_k}_UID)$$

Thus, in Step (3),

$$\forall j \;\; my_UID_i < nei_{i_j}_UID$$

Node i then asserts itself as a leader.

Case (ii)

Suppose node i has a neighbor which eventually asserts itself a leader, say $nei_{i_m}_UID$. Then,

$$remaining = (nei_{i_1}_UID, \ldots, nei_{i_m}_UID, \ldots)$$

and (3.1) is always *false* as long as node i does not assert itself as a leader. This is true because $nei_{i_m}_UID$ will not be removed from *remaining* unless an *S-element* is declared with node i_m as a member. Eventually, node i_m will assert itself as a leader, and will broadcast a list with node i as a member. Thus, (3.2) will be true, and node i will declare itself not a leader.

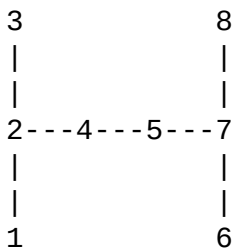
Case (iii)

Every node is a leader or neighbors a leader. Thus, eventually one of cases (i) or (ii) will occur, and in each case, node i is resolved.

4. Implementation

We have implemented this algorithm and tested it as a set of unique Unix processes that send and receive messages through files. The receive function is written so that a limited broadcast range is respected.

As an example system, suppose that we have the following layout:



with nodes located at:

Node	*X-Location*	*X-Location*
1	1	1
2	1	2
3	1	3
4	2	2
5	3	2
6	4	1
7	4	2
8	4	3

We assume a broadcast range of 2.

The neighbors found for the 8-node system is described in the trace as:

```
Neighbors of Node 5:  2  4  6  7  8
Neighbors of Node 3:  2  1  2  4
Neighbors of Node 4:  2  1  2  3  5  7
Neighbors of Node 8:  2  5  6  7
Neighbors of Node 1:  2  2  3  4
Neighbors of Node 6:  2  5  7  8
Neighbors of Node 2:  2  1  3  4  5
Neighbors of Node 7:  2  4  5  6  8
```

The resulting leaders and cliques are described as:

```
Device 1 leader?: Yes
Node 4 Clique 0:  1  2  3  4
Device 4 leader?: No
Node 3 Clique 0:  1  2  3  4
Device 3 leader?: No
Node 2 Clique 0:  1  2  3  4
Device 2 leader?: No
Node 5 Clique 0:  1  2  3  4
Device 5 leader?: Yes
Node 6 Clique 0:  5  6  7  8
Device 6 leader?: No
Node 8 Clique 0:  5  6  7  8
Device 8 leader?: No
```

Thus, the correct *S-cliques* were found, and the correct leaders were determined.

5. Summary and Conclusions

We have developed a leadership protocol for *S-Nets*. This protocol has been demonstrated as a set of Unix processes communicating through files.

The next set of issues include:

- Develop a formal correctness proof for the algorithm.
- Develop embedded system version of protocol.
- Develop other distributed versions of the *S-Net* algorithms.
- Implement complete set of *S-device* algorithms in physical testbed.

References

[1] Bares J E, Wettergreen D S 1999 Dante II: Technical description, results, and lessons learned. *Int J Rob Res.* 18(7):621-649 July

[2] Smith R, Frost A, Probert 1999 P A Sensor System for the navigation of an underwater vehicle. *Int J Rob Res.* 18(7):697-710 July

[3] Henderson, T C, Dekhil M, Morris S, Chen Y, Thompson W B 1998 Smart Sensor Snow. *IEEE Conf IROS.* Oct, pp 1377-1382

[4] Chen Y 2000 S-Nets: Smart Sensor Networks. MS Thesis, University of Utah

[5] Henderson, T C, Chen Y, 2000 Smart Sensor Networks. *IEEE Conf ISER.* Dec, pp 85-94

[6] Lynch N 1996 *Distributed Algorithms.* Morgan Kaufman Pub, San Francisco

[7] Sethian J A 1999 *Level Set Methods and Fast Marching Methods.* Cambridge University Press, Cambridge UK