Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Dynamic Leadership Protocol for S-nets

Gregory J. Barlow[1], Thomas C. Henderson[2], Andrew L. Nelson[1], and Edward Grant[1]

[1] Center for Robotics and Intelligent Machines, Department of Electrical and
Computer Engineering, North Carolina State University, Raleigh, NC 27695-7911
{gjbarlow, alnelso2, egrant}@ncsu.edu
[2] School of Computing, University of Utah, Salt Lake City, UT 84112
tch@cs.utah.edu

Abstract— **Smart Sensor Networks (S-nets) are groups of stationary agents (S-elements) which provide distributed sensing, computation, and communication in an environment. In order to integrate information from individual agents and to efficiently transmit this information to other agents, these devices must be able to create local groups (S-clusters). A leadership protocol that creates static clusters has been previously proposed. Here, we further develop this protocol to allow for dynamic cluster updating. This accommodates on-the-fly network re-organization in response to environmental disturbances or the gain or loss of S-elements. We outline an informal argument for the correctness of this revised protocol. We describe our embedded system implementation of the leadership protocol in simulation and using a colony of robots. Finally, we present results demonstrating both implementations.**

I. Introduction

While mobile robots may be provided with a large amount of on-board sensing [1] in order to acquire information about their environment, it is also possible to distribute sensing among agents in the environment. One such approach to distributed sensing is the Smart Sensor Network, or S-net. An S-net is composed of individual agents, called S-elements, which can compute, communicate, and sense the environment. S-nets have distributed computational capabilities and can provide processed data to mobile robots. Experiments have been performed in simulation to evaluate the utility of the S-net approach, and performance advantages compared to using only on-board sensors have been shown for certain cases [2]–[4].

Distributed sensing within an environment has a large number of potential applications. Because sensing is spatially distributed, S-nets are well suited to gradient calculation. For example, if robots were used to aid in fighting forest fires, and sensors could be distributed across the environment, an S-net could be used to identify hot spots and temperature gradients, aiding the deployment and movement of mobile agents [2].

S-nets can also be used as a distributed communications network. The use of digital pheromones has been proposed as an effective way to achieve distributed communication between agents in an environment [5]. S-nets are particularly well suited to this application; as a smart sensor, each S-element is capable of sensing, communication, and computation independent of other agents.

One of the distributed algorithms necessary for S-nets is an algorithm for S-cluster formation. For a very simple sensor network, mobile robots could communicate individually with all sensors in the environment. However, to enable distributed computation and to reduce the amount of communication a mobile agent must have with the S-net, grouping the individual S-elements into clusters with leaders is desirable. Mobile robots can then communicate with S-net leaders in order to acquire sensory information from the S-net. A solution to this leadership problem that forms static S-clusters has been presented in [6].

S-elements have the ability to communicate within a restricted range, have a fixed location, and have a unique integer identifier (ID number). The S-elements that make up the S-net should be grouped into S-clusters such that each leader has the lowest ID number in the cluster and is able to communicate with all the follower S-elements in the cluster. Followers are not necessarily able to communicate with all members of the S-cluster; it is only necessary for followers to be able to communicate with the S-cluster leader.

In this paper, we extend the static cluster leadership protocol to allow for dynamic cluster updating as S-elements are added and removed. This extension of the algorithm is designed to be asynchronous for use in embedded systems. We present the dynamic cluster leadership protocol, outline an argument for the correctness of the algorithm, and describe its implementation, both in simulation and using a colony of robots as S-elements. Results of both implementations are presented.

II. Leadership Protocol

An S-net system can be represented as an undirected graph where each node is an S-element and two connected nodes are within communication range of each other. Each node can be realized a number of ways. Previous simulations implemented each node as a unique Unix process. The current implementation uses a colony of small autonomous robots with communications abilities. Each S-element is implemented as an individual robot controller. Robot controllers can run either on real robots or within simulated robot agents.

In the static cluster case, the S-net leadership (SNL) algorithm [6] is run once in order to establish the clusters, and does not contain provisions for dynamically updating the clusters. After the clusters are resolved, clusters are used for local integration of information collected by the individual S-elements. This information integration is task dependent. Because the clusters are static, if any S-elements are added or removed no changes are made to the clusters. If a leader is removed, the performance of the S-net could be severely degraded, as the sensory data from all of its followers would be lost. The improved algorithm presented in this paper allows for dynamic updating of the clusters.

While the static SNL algorithm uses *broadcast* and *receive* functions for communication, the dynamic S-net leadership (DSNL) algorithm is designed to use directed node to node communications, with the ability to send messages to other agents, read messages sent by other agents from a queue, and query other nodes. This change was made primarily for ease of implementation, as the simulation environment and the colony of robots used as S-elements all have the capability to do node to node communication. This algorithm could also be implemented using broadcast communications.

The list of nodes in communication range is updated dynamically outside of the leadership algorithm. This is done using broadcast functions in a similar manner to SNL. Lists of nodes in the cluster, nodes not in the cluster, and unresolved nodes are kept by each node individually. Leaders are the only nodes that have full cluster lists; cluster lists for follower nodes include only the leader and the node itself. Cluster lists are updated at each iteration of the algorithm, so if a leader or a follower is removed, the affected nodes are aware. If a leader is removed, all of the followers in the cluster are reset and renegotiate their leadership status. In the case where followers are removed, the leader removes those nodes from its cluster list. There is no exit path from the leader state because clusters with only a single member are acceptable, so once a node becomes a leader, it can never become a follower.

Unlike SNL, DSNL has the added ability to control the process of claiming followers (Step 3.2.2 below). In the simplest case, followers join the cluster of the first leader to be asserted, with the lowest ID number winning in the case of a tie. While this is the default behavior in DSNL, this behavior could be altered to choose between multiple leaders asserting a claim based on factors such as preferring either large or small networks.

The dynamic S-net leadership algorithm (DSNL) is executed by each node. An outline of the algorithm is as follows:

DSNL Algorithm

```
1) Update the lists of nodes
   1.1) Update the list of all nodes
   1.2) Update the list of nodes in the cluster, the list of nodes not in the cluster, and the list of remaining nodes.
   1.3) If the node is a leader, check all messages to see if any nodes have changed from claimed to unclaimed.
2) If the node is not a leader and has no leader in its cluster list, reset the node to be unresolved.
   2.1) If the node has just become leaderless, notify all other nodes that this node is unclaimed.
   2.2) If the node's ID number is the lowest in the list of remaining nodes, set node to be a leader
3) If the list of remaining nodes isn't empty
   3.1) If the node is a leader, for all nodes in the list of remaining nodes, claim those nodes that are not yet claimed, and update the node's state. Once the list of remaining nodes is empty, the node is resolved.
   3.2) If the node is not a leader
      3.2.2) If the node is not resolved, check all messages to see if this node has been claimed by a leader. If so, add the leader to the cluster list and make the node resolved.
      3.2.2) If the node is not yet resolved, check each node in the list of remaining nodes and move all nodes that have been claimed to the noncluster list.
4) If the node is resolved
   4.1) If the node is a leader, execute the task specific code for leader nodes
   4.2) If the node is a follower, execute the task specific code for follower nodes
```

The state of each node includes:

id_num_i	node i's unique identity number.
leader	a Boolean, indicates whether or not the node is currently a leader.
resolved	a Boolean, indicates whether the node has resolved as either a leader or a follower.
nodelist	list of all nodes in communications range, it is initialized before the algorithm begins, and can change over time.
remaining	list of nodes whose status is still unresolved, initially equal to *nodelist*.
cluster	list of nodes in the cluster, initially null. For a leader, this list contains all nodes in the cluster, for a follower, this list only contains the leader and itself.
lastcluster	list of nodes in the cluster after the previous iteration of the algorithm. This is used to track when a follower becomes leaderless.
noncluster	list of nodes that are resolved, but not in the cluster, initially null. For a leader, this list contains all nodes within communications range that are not in the cluster, for a follower, this list contains all nodes except the leader and itself.

The full transition function for DSNL is shown below. The steps are identical to those in the algorithm outlined above, but all state updates are shown.

```
// Step 1
  // Step 1.1
  update nodelist

  // Step 1.2
  lastcluster = cluster
  cluster = intersect(nodelist, cluster)
  noncluster = intersect(nodelist, noncluster)
  remaining = nodelist - cluster - noncluster

  // Step 1.3
  if leader
    for each message in queue
      if sender is in noncluster
        if message is a claimed status change
          noncluster -= sender
          remaining += sender

// Step 2
if (!leader & (cluster == null))
  resolved = false
  noncluster = null
  remaining = nodelist

  // Step 2.1
  if (lastcluster != null)
    send_unclaimed(nodelist)

  // Step 2.2
  if (id_num(self) < min(id_num(remaining)))
    leader = true
    resolved = true

// Step 3
if (remaining != null)
  // Step 3.1
  if leader
    for each node in remaining
      if (current == self)
        cluster += current
        remaining -= current
      else
        if claimed(current)
          if (leader(current) == self)
            cluster += current
            remaining -= current
          else
            noncluster += current
            remaining -= current
        else
          claim(current)
  // Step 3.2
  else
    // Step 3.2.1
    if !resolved
      for each message in queue
        if sender is in nodelist
          if message is a claim
            cluster = leader
            resolved = true
            noncluster += remaining - leader
            remaining = null
    // Step 3.2.2
    if !resolved
      for each node in remaining
        if claimed(current)
          remaining -= current
          noncluster += current

// Step 4
if resolved
  // Step 4.1
  if leader
    perform the duties of a leader
  // Step 4.2
  else
    perform the duties of a follower
```

III. Correctness

The algorithm should achieve the following five objectives:

1) *leader = true*, *cluster = self and all followers*
   for any node that has the lowest ID number of all unresolved nodes in communication range.
2) *leader* = false, *cluster = leader*
   for any node that is within communication range of a leader.
3) *resolved = true*
   for every node.
4) $cluster \subset nodelist$
5) $cluster \neq \emptyset$
   if the node is resolved.

The first three objectives are similar to those defined for the original SNL, but the last two objectives are unique to DSNL.

Theorem 1: The algorithm will resolve with *leader = true*, *cluster = self and all followers* for any node that has the lowest ID number of all unresolved nodes in communication range at some point in time.

Proof: Suppose that node *i* has the lowest ID number of any of the nodes in communication range. The first time the DSNL algorithm is executed, the node will execute Step 2. In Step 2.2, $id\_num_i = min(remaining)$. Node *i* then sets $leader = true$ and begins to claim the other nodes in *remaining*.

Suppose that node *i* does not have the lowest ID number of any of the other nodes in communication range, but all nodes with lower ID numbers than *i* are within communication range of nodes that assert themselves as leaders. While those nodes with lower ID numbers than *i* are still unresolved, *i* will remain unresolved as well, since its ID number will not be the lowest in *remaining*. Once all nodes with lower ID numbers than i are resolved as followers, node i will set $leader = true$ and begin to claim any unresolved nodes. ■

Theorem 2: The algorithm will resolve with *leader* = false, *cluster* = *leader* for any node that is within communication range of a leader.

Proof: Suppose that node i is within communication range of a node which eventually resolves as a leader. As long as node i is unresolved, the test in Step 2 will be true, since *cluster* will be empty. Once a node within communication range of i becomes a leader, this leader node will try to claim i. When node i runs Step 3.2.1, there will be a message from this leader node claiming i, which will resolve i. If more than one node claims i, the node that claimed it first or the node with the lowest ID number, in that order, will become node i's leader.

Suppose a new node is added after other nodes have previously resolved. The same process applies as if the node had been added as the same time as the other nodes. Because all nodes add new nodes to *remaining* even after becoming resolved, and leaders continue to attempt to claim nodes after becoming resolved, a node that is added within range of a leader will eventually resolve as a follower.

Suppose a node's leader is removed. The node will reset its own state in Step 2, and then send a message to all nodes in *nodelist* to notify them of the change in Step 2.1. Then, the same process as above applies, where leader nodes within range, after adding the node to *remaining* in Step 1.3, will attempt to claim the node, unless the node now has the lowest ID number of unresolved nodes in range, in which case Theorem 1 applies. ■

Theorem 3: Within a finite number of iterations, *resolved* = *true* for every node.

Proof: Every node is either a leader or is within communication range of a leader. Hence, all nodes are subject to either Theorem 1 or 2. In either case, $resolved = true$. ■

Theorem 4: During every iteration, $cluster \subset nodelist$ for all nodes.

Proof: Suppose one of the followers in a cluster is removed. In this case, the leader of that cluster should remove the follower from the cluster list. This occurs in Step 1.2, where the lists of nodes are updated. Because $cluster = intersect(cluster, nodelist)$ the cluster is always set to be a subset of the nodelist every time the DSNL algorithm is executed, and *cluster* is always a subset of *nodelist*. ■

Theorem 5: If a node is resolved, then $cluster \neq \emptyset$.

Proof: Suppose a leader node is removed. When this occurs, $cluster = \emptyset$ for all of the followers in the cluster. The next time the DSNL algorithm is executed, Step 2 is true, because for all the followers, *cluster* is reduced to the null set in Step 1.2. In Step 2, these nodes would set $resolved = false$, which would restart the DSNL algorithm and subject these nodes to either Theorem 1 or 2. ■

TABLE I
NODE POSITIONS FOR EXAMPLE 1

Node	X	Y
1	20.25	119.25
2	58.5	21.375
3	110.25	128.25
4	83.25	72

IV. IMPLEMENTATION

We have implemented this algorithm both in simulation and on a colony of robots, where each robot is used as an S-element. Both implementations use identical communication functions. Communication occurs by transferring files containing messages between nodes.

In the implementation of the DSNL algorithm in simulation, all S-elements run on a single computer using Matlab for the simulation. Each S-element has distinct state information. During each iteration, an S-element updates its sensor information, updates its communications, and then runs a controller, which in this case is the DSNL algorithm. Information is exchanged between nodes in two ways, queries and messages. Every node may make certain information available to other nodes, for example, the *claimed* status of the node. This information may be queried at any time during the controller cycle by the other nodes. Messages may be sent at any point when the controller is running, but the message queue on each robot is only re-indexed at the beginning of each iteration. While iterations are synchronized in simulation for convenience, the communications model was designed to be asynchronous.

The same agent architecture is used for the implementation of the DSNL algorithm on a colony of robots. Each robot is completely autonomous, with on-board computation, sensing, and communications abilities [7]. The robots use PC/104 hardware, with USB CCD cameras for sensing, and wireless Ethernet for communication. Infinite Atom Linux 2.0, based on Red Hat Linux 8.0, is installed on all the robots, and Matlab 5.3 is used to run the controller. Unlike in simulation, the embedded system communications are asynchronous.

Experiments with both the simulated and embedded implementations verify the performance of the DSNL algorithm. These experiments were designed to test the ability of the algorithm to achieve the goals outlined in Section III. In all cases the DSNL algorithm was able to fulfill the specified goals.

A. Example 1

As an example system, suppose we have the arrangement of S-elements shown in Figure 1. The nodes are located at the positions given in Table I, are active for the times shown in Figure 2, and have a maximum communication range of 90. In the simulated system, the status of each node for selected iterations is shown in Table II. Figure 1 shows the status of all nodes once clusters resolve for the first time. Node 1 is then removed at time step 10, and Figure 3 shows the status of the remaining three nodes once they re-resolve.

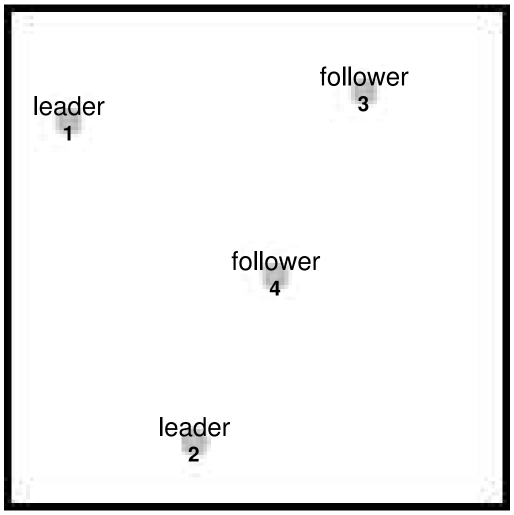

Fig. 1. Initial resolution of S-elements in Example 1

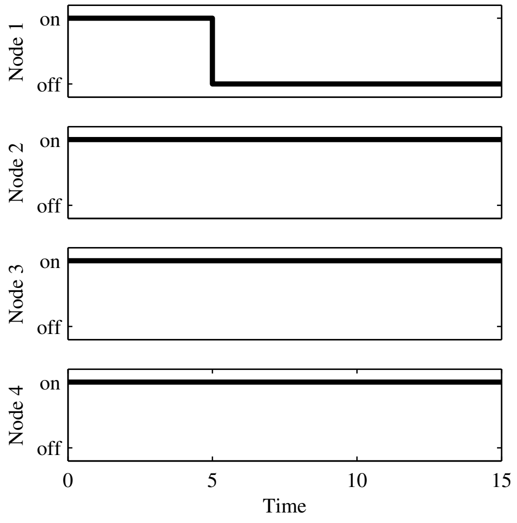

Fig. 2. Times of operation for Example 1

TABLE II
STATE INFORMATION FOR EXAMPLE 1

Time	Node	Status	*cluster*	*noncluster*	*remaining*
0	1				1 3 4
	2				2 4
	3				1 3 4
	4				1 2 3 4
1	1	leader	1		3 4
	2	leader	2		4
	3				1 3 4
	4				1 2 3 4
4	1	leader	1 3 4		
	2	leader	2	4	
	3	follower	1 3	4	
	4	follower	1 4	3	
10	2	leader	2		4
	3	leader	3		4
	4				2 3 4
12	2	leader	2 4		
	3	leader	3	4	
	4	follower	2 4	3	

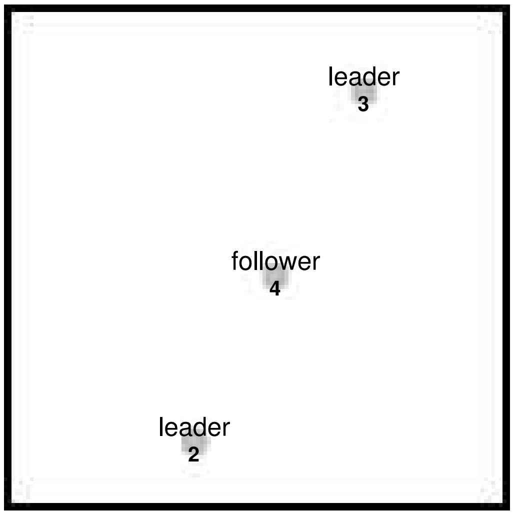

Fig. 3. Final resolution of S-elements in Example 1 after Node 1 has been removed

To test the transference of the algorithm to embedded systems, the same configuration of nodes was implemented using a colony of robots, shown in Figure 4. The clusters that were resolved proved to be identical in the simulated and embedded cases. In both the simulated and embedded cases, the DSNL algorithm determined the correct leaders and S-clusters for the S-net.

B. Example 2

For a more complex example, suppose we have the arrangement of S-elements shown in Figure 5. The nodes are located at the positions given in Table III, are active for the times shown in Figure 6, and have a maximum communication range of 50. In the simulated system, the status of resolved clusters for selected iterations is shown in Table IV. After the clusters are initially resolved, Node 1 is removed. After the clusters resolve again, Node 20 is added. Figure 5 shows the status of all nodes once clusters resolve for the final time. Example 2 serves as further verification of the performance of the DSNL algorithm. This simulation demonstrates the scalability of the algorithm as the number of nodes increases.

V. CONCLUSIONS

We have developed an improved leadership protocol for S-nets that allows for dynamic updating of clusters. An implementation of this protocol for embedded systems has also been developed. This protocol has been demonstrated both in simulation and using a colony of mobile robots as S-elements.

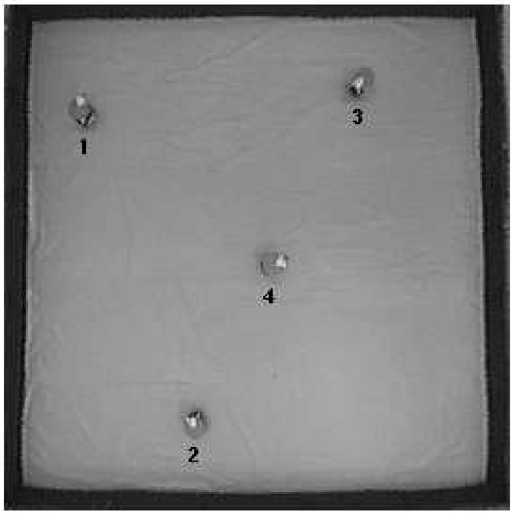

Fig. 4. Mobile robots as S-elements for Example 1

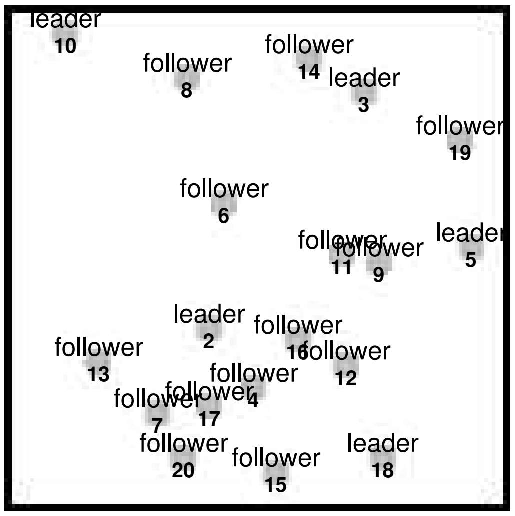

Fig. 5. Final resolution of S-elements in Example 2 after Node 1 has been removed and Node 20 has been added

TABLE III
NODE POSITIONS FOR EXAMPLE 2

Node	X	Y	Node	X	Y
1	18	106	11	92	70
2	56	50	12	93	40
3	98	114	13	26	41
4	68	34	14	83	123
5	127	72	15	74	11
6	60	84	16	80	47
7	42	27	17	56	29
8	50	118	18	103	15
9	102	68	19	124	101
10	17	130	20	49	15

TABLE IV
CLUSTER LISTS FOR EXAMPLE 2

Time	Leader	*cluster*
5	1	1 6 8 10
	2	2 4 7 9 11 12 13 15 16 17
	3	3 14 19
	5	5
	18	18
15	2	2 4 6 7 9 11 12 13 15 16 17
	3	3 8 14 19
	5	5
	10	10
	18	18
25	2	2 4 6 7 9 11 12 13 15 16 17 20
	3	3 8 14 19
	5	5
	10	10
	18	18

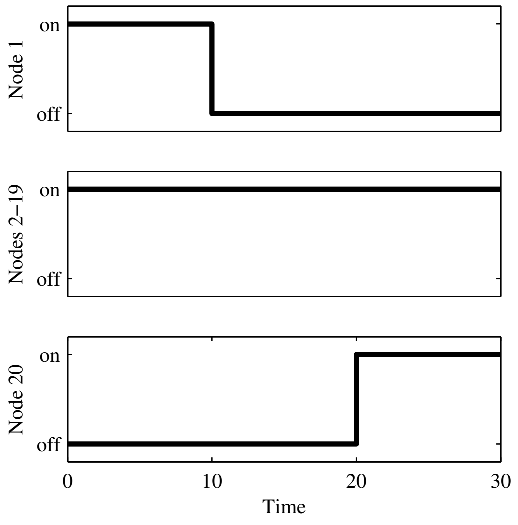

Fig. 6. Times of operation for Example 2

REFERENCES

[1] J. E. Bares and D. S. Wettergreen, “Dante II: Technical description, results, and lessons learned,” *International Journal of Robotics Research*, vol. 18, no. 7, pp. 621–649, July 1999.
[2] Y. Chen and T. C. Henderson, “S-Nets: Smart sensor networks,” in *Symposium on Experimental Robotics*, Honolulu, December 2000, pp. 85–94.
[3] Y. Chen, “S-Nets: Smart sensor networks,” Master’s thesis, University of Utah, 2000.
[4] T. C. Henderson, M. Dekhil, S. Morris, and W. B. Thompson, “Smart sensor snow,” in *IEEE Conference on Intelligent Robots and Intelligent Systems*, October 1998.
[5] H. V. D. Parunak, M. Purcell, and R. O’Connell, “Digital pheromones for autonomous coordination of swarming UAV’s,” in *AIAA Conference on Unmanned Aerospace Vehicles*, 2002.
[6] T. C. Henderson, “Leadership protocol for s-Nets,” in *Multisensor Fusion and Integration*, Baden-Baden, Germany, August 2001, pp. 289–292.
[7] J. M. Galeotti, “The EvBot: A small autonomous mobile robot for the study of evolutionary algorithms in distributed robotics,” Master’s thesis, North Carolina State University, May 2002.