An Implementation of Distributed Unconnected Graph Determination in Sensor Networks

Joshua Lifton, Michael Broxton, and Joseph Paradiso

MIT Media Lab, Responsive Environments Group, 1 Cambridge Center 5FL,
Cambridge, MA 02142 USA
{lifton, mbroxton, joep}@media.mit.edu
http://www.media.mit.edu/resenv/

Abstract. This paper details recent progress toward implementing and testing in hardware an unconnected graph determination algorithm for distributed sensor networks. Specifically, we address the question of enabling a distributed sensor network to determine if it was been fragmented into two disjoint networks due to a large scale event. We present the problem, propose a solution, and detail the progress toward realizing our proposed solution in hardware.

1 Introduction

The defining features of a distributed sensor network are:

- a large number of independent constituent nodes
- the ability of each node to sense and react to the world
- the ability of each node to communicate with some subset of other nodes in the network

Recent efforts to realize distributed sensor networks in hardware (as opposed to the numerous simulations already developed) typically make room in the hardware design for each of these three characteristics, but in practice rarely exhibit all three simultaneously. Our goal in this work is to demonstrate a real distributed sensor network comprised of approximately one-hundred nodes running simple sensing algorithms that require all three features listed to operate successfully. In addition, we will summarize a number of practical concerns when dealing with hardware realizations of distributed sensor networks.

1.1 Pushpin Computing

This work uses the Pushpin Computing hardware and software platform as its distributed sensor network [1]. The Pushpin platform consists of approximately 100 identical nodes (Pushpins). At the core of each Pushpin is a 22 MIPS 8051-based microcontroller unit (MCU) imbued with 32KB flash, 2.25 KB RAM, and a host of analog and digital peripherals. An infrared communication module permits communication with neighboring Pushpins within approximately six

inches at a data rate of up to about 300kbps, but currently set at 92kbps. Two pins of unequal length extrude from the bottom of each Pushpin in order to make contact with power and ground planes when inserted into a layered corkboard-like substrate which acts as the power source for all Pushpins. Each Pushpin draws up to 30mA at 3V, but typical operating current is less than half that amount and sub-milliampere operating modes are possible as well. A sensing and actuation module contains a five-element multi-colored LED display and a light intensity sensor. The MCU has a built-in temperature sensor as well. Eight other analog channels are reserved for future use.

A small operating system runs on each Pushpin and includes, among other things, a random number generator and seed, an interrupt-driven variable-length packet-based subsystem for communication with immediate neighbors, a simple memory manager, and the facilities for remotely updating the the entire OS and propagating those changes along to neighboring Pushpins. A single Pushpin tethered via a serial cable to a desktop computer allows the user to query, monitor, and update other Pushpins using a compiling, debugging, monitoring, and control software package designed specifically for the Pushpins.

Full details of the Pushpin hardware platform as well as a previous, but related version of the current software platform are available online [1, 2]. Butera's Paintable Computing [3] simulation work inspired the Pushpin hardware realization and provides much background information otherwise omitted here.

1.2 Prior Art

The majority of work within the realm of distributed sensor networks is simulation-based. However, hardware platforms are slowly proliferating. Most prominent among these platforms is the Intel Berkeley 'motes' hardware platform [4] and its associated TinyOS software platform [5]. Briefly, each mote contains a 4MHz MCU imbued 128KB of program memory and 4KB of RAM, a 512KB cache of EEPROM, a 917MHz radio capable of 50kbps, and expansion ports for numerous sensors and actuators. A recent demonstration of the motes included an impressive 800 nodes used to build up a network routing tree [6].

Madden, Franklin, Hellerstein, and Hong [7] put forth the Tiny AGgregation (TAG) service for using SQL-like queries in a network of motes running TinyOS. Although they clearly demonstrated a useful and much-needed concept (an efficient mechanism for making high-level queries of a distributed sensor network), their prototype implementation amounted to only 16 motes answering a "how many nodes are there in the network?" query.

Mainwaring, Polastre, Szewczyk, and Culler [8] have deployed small networks of motes amid environmentally harsh conditions at two nature reserves in order to monitor specific habitats. Sensor-rich real-time data from the sensor networks were made available on the Internet by routing compressed data to a satellite uplink. Their work implements a relatively simple data collection application in order to gain valuable practical knowledge on integrating complex systems operating in harsh environments with limited energy resources.

Liu, Cheung, Guibas, and Zhao [9] developed a centralized computational geometry approach in dual space to track a half-plane shadow moving across a random but fixed grid of 16 motes. As with the Pushpins, all nodes received power from a central source for the sake of convenience. All computation and actuation took place on a PC communicating with the network vis a tethered mote.

2 Statement of Problem

2.1 The DUGD Problem

The distributed unconnected graph determination (DUGD) problem can be stated as follows:

Definition 1. *Consider a distributed sensor network that can be modeled as a connected directed graph $G = (V, E)$ of vertices V and edges E, where each vertex represents a sensor node in the network and each edge represents a communication channel between two nodes. Given a subset of vertices $N \subset V$ and the subset of edges $M \subset E$ associated with N, find an algorithm for each sensor node to execute such that each sensor node can determine whether the graph $G = (V - N, E - M)$ is still connected.*

In other words, given a network in which all nodes can communicate with all other nodes either directly or through intermediary nodes, is it possible for the network to know if it has been split into two disjoint networks due to the removal of some of the nodes.

2.2 Relevance of the DUGD Problem

At first glance, the DUGD problem seems unsolvable in the sense that there is no algorithm that will correctly solve the problem in every instance. For example, if all but one node in a network are removed, the remaining node has no way of determining by means of processing and communication alone that all the other nodes were removed and not just the remaining node's immediate neighbors. The implicit assumption here, however, is that the subset of nodes to be removed are actually removed before the algorithm is run. However, this is not in the statement of the problem, a fact that makes the problem much more tractable.

The danger, of course, is that without the implicit assumption the nodes are actually removed before the algorithm is run, the DUGD problem may become either uninteresting or trivial or both. We argue that the problem is still interesting (useful) and non-trivial. First, consider an energy constrained distributed sensor network in which a subset of the nodes are low on energy reserves. Although these nodes will eventually shutdown and effectively be removed from the network, they will typically foresee their departure from the network and may have enough time and energy to coordinate with the rest of the network to

determine the post-shutdown network topology. This is a useful when, for example, information required by the entire network (and the subsequent disjoint networks if the network is in fact bisected) is expensive in terms of time, energy, and/or memory to distribute to every node in the network. In this case, knowing that the network will soon be split in two allows time for the critical information to be distributed such that both subsequent networks maintain copies and can continue functioning independently.

The DUGD problem is non-trivial in that, while many solutions exist, not all are both robust and resource efficient. For example, one solution is simply to establish a multi-hop communication link between two nodes on opposite sides of the network and then constantly passing ping messages back and forth. If the communication link is broken and cannot be rerouted then a bisection is presumed to have occurred. Although robust, this solution is not resource efficient. (Note however, that this is essentially the only recourse to even partially solving the DUGD problem if the nodes are removed immediately.)

3 Experimental Setup

We set out to solve the distributed unconnected graph determination problem on an ensemble of approximately 100 Pushpins. In our experimental setup, the Pushpins are randomly[1] inserted into the substrate and cover approximately a one square meter area. See Figure 1. Once powered (either as they are inserted or all at once by power cycling the substrate), a flashlight is used to pass a well-defined patch of light over the ensemble. Those nodes sensing above a certain threshold of light are defined to be those that will soon be removed from the network. The goal then is for each Pushpin in the network to both know if the network would be unconnected if the illuminated nodes were removed and, if so, which of the disjoint networks that particular Pushpin would belong to.

This setup was chosen explicitly so that any scalable, robust, and resource efficient solution to the problem stated above would necessarily utilize all three salient features of a distributed sensor network – large number of nodes, ability of the nodes to sense the world, and the ability of the nodes to communicate locally.

4 Progress & Results

Our solution to the DUGD problem is not yet complete as of this writing. However, we've made significant progress on several fronts toward realizing a general solution.

[1] Random in that a graduate student placed them by hand without any preconceived notion of a pattern. This essentially results in a space-filling distribution.

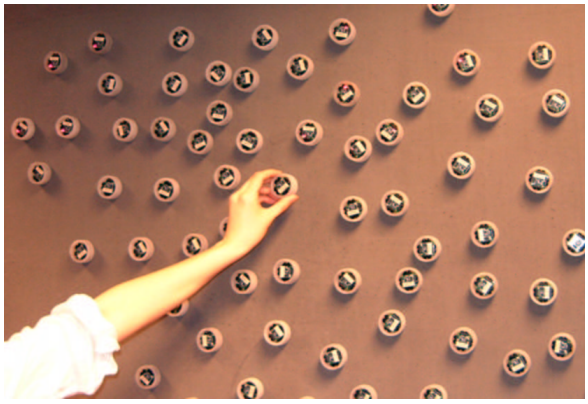

Fig. 1. An ensemble of Pushpins.

4.1 Additions & Improvements

Much of our effort focused on improving and adding to the Pushpin platform as a whole so as to facilitate building and testing applications in general. Most of these improvements and additions came about as a result of hard learned lessons from the first iteration of Pushpins. The most significant improvements and additions are detailed here.

IR-to-RS232 Tethered Pushpin: Previously, we had no way to communicate directly between a PC and the Pushpins. We developed and built a communication module to convert between serial and infrared to solve this problem. This converter allows the PC to passively monitor Pushpin communication and to issue user commands by means of the Pushpin integrated development environment (IDE).

OS Updater: Previously, there was no way to update the Pushpin operating system (Bertha) aside from physically connecting to the Pushpin with a MCU

programmer and re-burning the flash memory with new code. Given that reprogramming all the Pushpins takes on the order of 2 hours, not having any other way of updating the OS made debug cycles prohibitively long. We developed a remote updating system whereby the binary image of the new operating system can be propagated throughout the network through the usual Pushpin communication channels. Updating the OS is now a matter of simply pushing a button in the IDE and pointing the tethered Pushpin at one of the other Pushpins. It will then propagate to all neighbors. Version control is included so that multiple copies of the same operating system can be inserted into the network without redundantly overwriting themselves on a single Pushpin.

Scheduled Communication: A messaging layer was added on top of the packet layer of the communication subsystem. Multiple (currently ten) messages can be queued for transmission. The number times the message is to be transmitted and the interval between transmissions can be specified when the message is queued. This feature was added after realizing just how much lossy communication effects algorithm design. In essence, this type of message queuing helps the programmer to acknowledge and design around an imperfect communication channel.

Elimination of Process Fragments: The previous implementation of process fragments was not at all efficient with communication bandwidth and did not recover gracefully from communication failures. In particular, it relied on transferring relatively large pieces of binary code as well as regular exchanges of internal state between neighboring Pushpins. A cleaner code interface and the ability to more easily update the operating system allows the programmer to compile applications into a smaller operating system and quickly distribute the new operating system among the Pushpins.

4.2 Proposed DUGD Algorithm

We have designed and implemented a distributed unconnected graph determination algorithm. The implementation of the algorithm complete, but still being tested at the time of this writing.

Some colorful political terminology helps when explaining and understanding our algorithm. An `ENLIGHTENED` Pushpin is one which has sensed a light above its threshold. `ENLIGHTENED` Pushpins correspond to those Pushpins that will ostensibly be removed from the network and in doing so potentially cause the network to become two disjoint networks. The `ELECTORATE` is comprised of all Pushpins within a certain hop count (typically 2 or 3) of an `ENLIGHTENED` one. This usually corresponds to the Pushpins immediately surrounding, but not included within where the light passed. A `PLEBE` is a Pushpin that is neither `ENLIGHTENED` nor a member of the `ELECTORATE`. These are typically Pushpins distant from the path the light took. A high-level description of the algorithm is then:

1. Set affiliation to `PLEBE`.
2. Calibrate light sensor and set ADC to automatically sample and compare against calibration + noise.
3. If light > calibration + noise, set affiliation to `ENLIGHTENED`, turn on the green LED and initiate a n-hop count gradient.
4. All `PLEBE`s within n hops set affiliation to `ELECTORATE`, turn on the red LED and generate a random 16-bit number. Broadcast this random number ('nomination') to all neighboring Pushpins.
5. All members of the `ELECTORATE` replace their own nominations with any lower nomination received from neighboring Pushpins. Rebroadcast the new nomination.
6. After the nominating has died down, the `ENLIGHTENED` ones request votes from the `ELECTORATE`.
7. Upon receiving a request for a vote, members of the `ELECTORATE` broadcast their current nomination.
8. The `ENLIGHTENED` ones collect all nominations and share them amongst themselves. The lowest of all nominations they have collected is selected as the winner and broadcast to the `ELECTORATE`.
9. If a member of the `ELECTORATE`'s nomination is greater than the winner selected by the `ENLIGHTENED` ones, then that member of the `ELECTORATE`should turn off its red LED, turn on its yellow LED and broadcast a yellow message to the `PLEBE`s. Otherwise, it should broadcast a red message to the `PLEBE`s.
10. All `PLEBE`s should do as instructed and turn on the appropriate LED upon receiving either a yellow or red message.

The typical steady-state behavior of this algorithm is that all Pushpins will have only one LED turned on. Those which have sensed light should have their green LED turned on. All others should have either their red or yellow LED turned on. If the light event did not bisect the network, then only red and green LEDs should be on. See Figure 3. If the network was bisected, then yellow LEDs will appear as well. See Figure 2. Furthermore, the union of red and green Pushpins forms a connected network disjoint from all yellow Pushpins. Likewise, the union of yellow and green Pushpins forms a connected network disjoint from all red Pushpins.

4.3 Current Status

As already noted, the algorithm just described has been implemented, but has not yet been tested enough to warrant the reporting of any results. If everything goes smoothly (as it appears to be), initial results may appear within a week or two. The OS update mechanism and other improvements and additions to infrastructure used by the algorithm have already been tested.

5 Discussion

The key design points of our algorithm are:

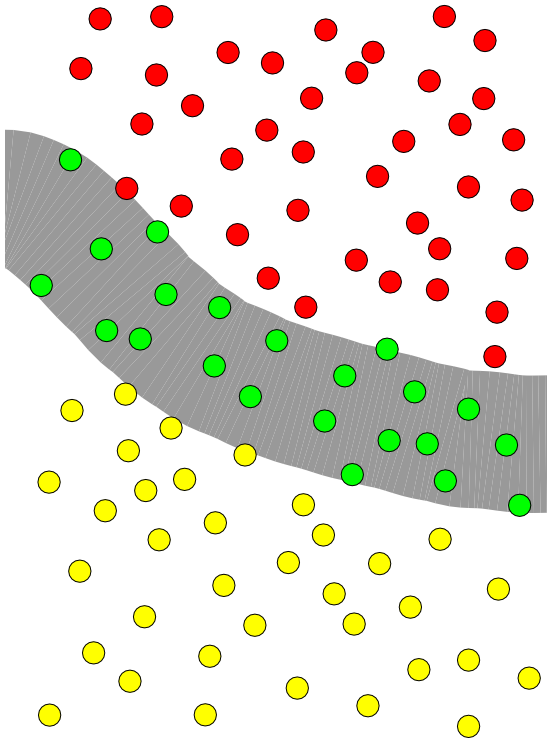

Fig. 2. Desired final state of the proposed DUGD algorithm in the case where the network is fragmented into two disjoint networks. The gray swath represents the path the light took. Green nodes have sensed the light. Red and yellow nodes are disjoint.

- Distributed: All Pushpins run the same application code. No Pushpin is singled out as different than the others except for a 16-bit random seed, its sensor readings, and a small amount of clock skew which is inconsequential for our purposes.
- Resource efficient: The default state of each Pushpin is to do nothing. An interrupt is generated when the light sensor's value is above a threshold determined by a calibration routine at startup. No other resources are required until this interrupt occurs or communication is initiated by another Pushpin.
- Scalable in Space: Only Pushpins in the immediate vicinity of the where the light passed over need to be active. All other Pushpins remain in the default state until a message informing them of the result of the algorithm is received. Thus, the algorithm is independent of the size of the network. This contributes appreciably to the resource efficiency mentioned above. Furthermore, it is independent of the size of the initiating event (patch of light) as well.

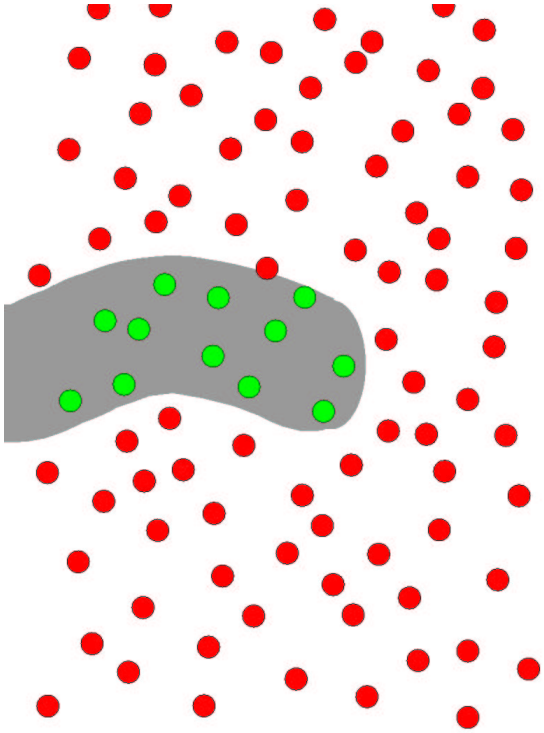

Fig. 3. Desired final state of the proposed DUGD algorithm in the case where the network remains connected despite the green nodes not participating. The gray swath represents the path the light took. Green nodes have sensed the light.

– Scalable in Time: Our algorithm adapts gracefully to changes over time in the initial conditions. Namely, it accommodates for Pushpins being activated by the light source over a period of time, be it milliseconds or minutes. This is an important trait in general for sensing applications since the time scales of the events to be sensed not necessarily fixed.

As mentioned earlier, the success of this algorithm relies crucially on the fact that the nodes to be removed from the network aren't actually removed until they have had a chance to coordinate with their neighbors. Without this property, the design points listed are no longer valid.

Rather than solving the DUGD problem, we had initially intended on demonstrating a distributed shape recognition algorithm capable of distinguishing a patch of light by its shape, either circle, square, or triangle. In collaboration with Bill Butera, we developed a corner counting algorithm to solve this problem. Bill implemented the algorithm in the Paintable Computing simulator with significant success at high densities of nodes. As expected, lower densities of nodes caused the algorithm to fail more often. The key distinction between the dis-

tributed unconnected graph determination algorithm and the distributed shape recognition algorithm is that the former tries to answer a question about network connectivity whereas the latter tries to answer a question about physical shapes. That is, one is grounded in the network and the other is grounded in the physical world. The reason the distributed shape recognition algorithm worked at high densities of nodes is that, given a physical scale length of interest and a communication radius that is linear with the inverse of node density (i.e., each node has a constant number of neighbors, regardless of density), network proximity becomes more tightly correlated with physical proximity as node density increases. Even with 100 Pushpins within a square meter, the correlation of network proximity and physical proximity is poor at best. This phenomenon comes across clearly in simulation and would only become more apparent when subjected to the non-ideal communication channels of the real world. The DUGD problem does not suffer from such hardships and posed itself as more tractable and arguably more useful.

6 Future Work & Conclusions

Future work is happening now. Final conclusions regarding our proposed DUGD algorithm will have to wait until concrete results have been realized.

Medium-term goals for the Pushpin work include designing an infrared communication module with improved spatial uniformity with regard to both transmission and reception, better characterizing average network behavior, and developing data aggregation and distributed feature extraction algorithms which adapt to the scales of interest of the phenomenon being sensed.

Long-term goals are centered on the idea of a 'sensate skin' composed of many nodes at millimeter^{-1} or centimeter^{-1} scale densities. Such densities have two important practical consequences. First, radio communication as we know it will not be applicable. That is, the near-field will dominate over the far-field. Inductive, capacitive, optical, and other methods of communication may perform better at such small scales. Second, the common assumption that all distributed sensor networks are energy constrained is no longer necessary. This is due in part to the decreased distance across which communication must take place, and in part to the potential for manufacturing processes to deliver power throughout the sensate skin to all nodes.

Finally, a general lesson learned from this project is that many of the details glossed over (by necessity) in many simulation environments are exactly those which dictate the most important algorithm design rules. Lossy communication, asynchrony, and events spread over long periods of time are all fundamental aspects of distributed sensor networks and should be treated as such. For sensor networks to succeed when the number of nodes becomes large, they must self-organize. That is, the global behavior of the system must emerge from many instances of local behavior. Engineering such self-organization requires getting the local behavior right first.

Acknowledgments

Special thanks to Bill Butera the members of the fall 2002 class of MIT's *6.978: Biologically Motivated Programming Technology for Robust Systems* for their support and constructive critique.

References

1. Pushpin Computing website: `http://www.media.mit.edu/~lifton/Pushpin/`
2. Lifton, J.; *Pushpin Computing: a Platform for Distributed Sensor Networks*, MIT Media Laboratory, master of science thesis, 2002.
3. Butera, W.: *Programming a Paintable Computer*, MIT Media Laboratory, doctoral dissertation, 2002.
4. motes website: `http://www.xbow.com/Products/Wireless_Sensor_Networks.htm`
5. TinyOS website: `http://webs.cs.berkeley.edu/tos/`
6. 800-mote demo summary: http://webs.cs.berkeley.edu/800demo/
7. Madden, S.; Franklin, M.; Hellerstein, J.; Hong, W.: *TAG: a Tiny AGgregation Service for AD-Hoc Sensor Networks*, Intel Research, IRB-TR-02-011, 01 August 2002.
8. Mainwaring, A.; Polastre, J.; Szewczyk, R.; Culler, D.: *Wireless Sensor Netorks for Habitat Monitoring*, Intel Research, IRB-TR-02-006, 19 June 2002.
9. Liu, L.; Patrick, C.; Guibas, L.; Zhao, F.: *A Dual-Space Approach to Tracking and Sensor Management in Wireless Sensor Networks*, PARC technical report P2002-10077, March 2002.