Presented at the IPSN/SPOTS '05 Conference in Los Angeles, CA on April 25-27, 2005

EXPERIENCES AND DIRECTIONS IN PUSHPIN COMPUTING

Joshua Lifton, Michael Broxton and Joseph A. Paradiso
Responsive Environments Group
MIT Media Lab
Cambridge, MA 02139

Abstract— **Over the last three years we have built and experimented with the Pushpin Computing wireless sensor network platform. The Pushpin platform is a tabletop multihop wireless sensor network testbed comprised of 100 nodes arbitrarily placed within a one-square-meter area. The Pushpin platform's concise form factor and extreme node density allow for fine-grained control of its environment and immediate user interaction, thereby uniquely situating it between simulated and real world sensor networks. This paper details our salient successes and lessons learned along the way. We also discuss how these experiences have shaped our vision of the future of wireless sensor networks and some concrete research directions to follow.**

I. INTRODUCTION

The wireless sensor network research community has grown tremendously in the last several years. Work in simulation accounts for much of this growth. Of the work done on actual wireless sensor network hardware platforms, much of it has been confined to only a few platforms, such as the University of California at Berkeley motes or the Compaq iPaqs. Use of other platforms typically remains confined to the platforms' creators and their associates. In many respects, this situation works well – most researchers can share a small set of common, well-supported platforms and don't need to worry about hardware implementation details, while a small number of researchers can experiment with their own custom systems that perhaps provide functionality not available elsewhere. One of the downsides of this situation, however, is that much of the knowledge gained from implementing a wireless sensor network hardware platform is never passed on to the rest of the research community, which more often than not only sees the finished product and does not benefit from the knowledge gained along the way. In the spirit of this workshop, this paper is a small step toward countering this trend.

The Pushpin platform uniquely combines many of the affordances of simulated sensor networks, such as precisely controlled experiments, ease of use, and user interaction, with the benefits of real-world sensor networks, such as real data collected from the physical world, realistic communication channels, and continuous real-time operation. These characteristics have allowed us to effectively straddle simulation and the real world.

In what follows, we give an overview of the process of building up and working with the Pushpin Computing wireless sensor network platform and attempt to distill out generally applicable lessons, both successes and failures. Along the way, we discuss how working with our own hardware platform has informed our vision of sensor networks and suggest several avenues of future research.

II. PUSHPIN OVERVIEW

Development on the Pushpin Computing platform began in earnest almost three years ago as an attempt to build a prototyping platform for extremely high density distributed wireless sensor networks. As such, it includes low-maintenance, easily reconfigurable hardware, a set of mature software libraries, and a suite of interactive programming and debugging tools. This section describes the Pushpin platform as it exists today. The lessons learned during its evolution will be discussed later.

A. Hardware

Each node, or Pushpin, consists of four easily swappable stacked modules, one each for power, communication, processing, and sensing/actuation [1]. See Figures 1 and 2.

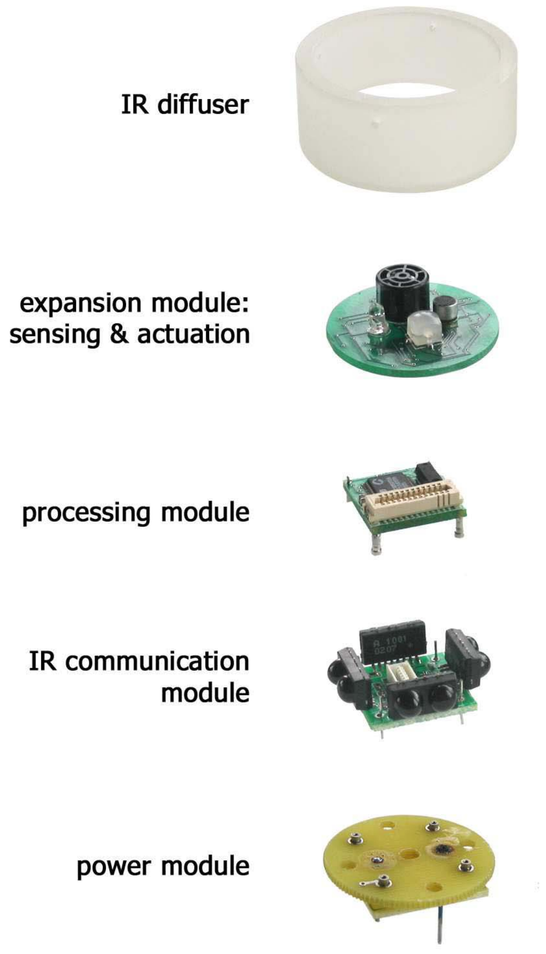


Fig. 1. This expanded view of a single Pushpin shows each of the four standard modules. When assembled with the modules shown here, a Pushpin is 3-cm in diameter by 3-cm high.

Perhaps the most immediately salient feature of the typical Pushpin platform configuration is that each node receives power from two insulated prongs that are inserted into a polyurethane foam substrate

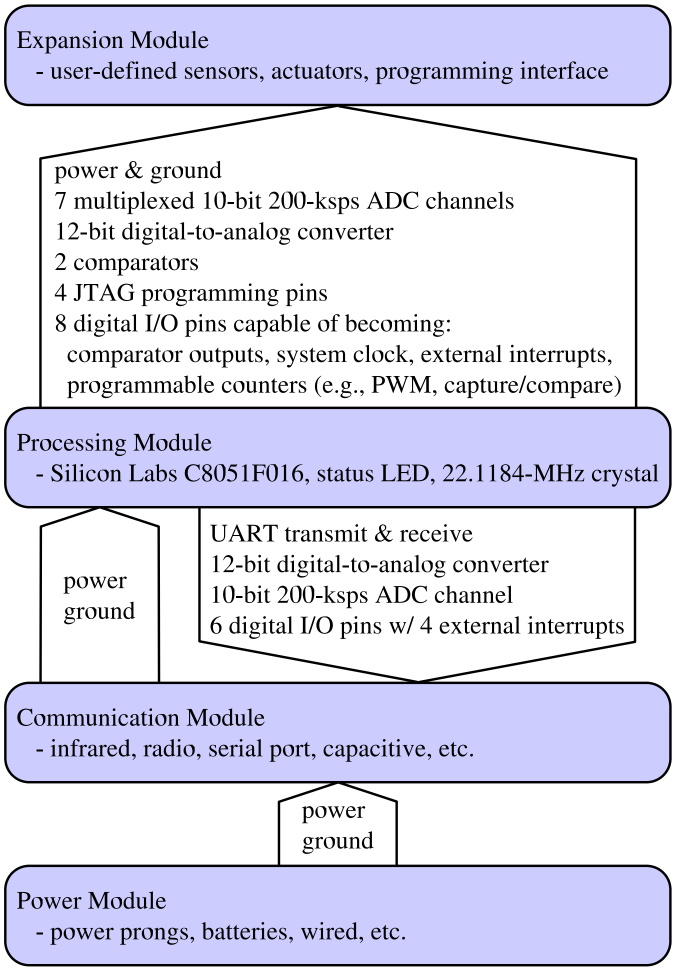

Expansion Module
- user-defined sensors, actuators, programming interface
power & ground
7 multiplexed 10-bit 200-ksps ADC channels
12-bit digital-to-analog converter
2 comparators
4 JTAG programming pins
8 digital I/O pins capable of becoming:
comparator outputs, system clock, external interrupts,
programmable counters (e.g., PWM, capture/compare)
Processing Module
- Silicon Labs C8051F016, status LED, 22.1184-MHz crystal
power
ground
UART transmit & receive
12-bit digital-to-analog converter
10-bit 200-ksps ADC channel
6 digital I/O pins w/ 4 external interrupts
Communication Module
- infrared, radio, serial port, capacitive, etc.
power
ground
Power Module
- power prongs, batteries, wired, etc.

Fig. 2. Each Pushpin node is comprised of four modules separated by function (shaded boxes). Each module makes available certain well-defined resources to other modules (arrows pointing from provider module to recipient module). Although various power, communication, and expansion modules have been built and used, the current standard Pushpin configuration includes a two-pronged power module, an IR communication module, and a sonar time-of-flight module.

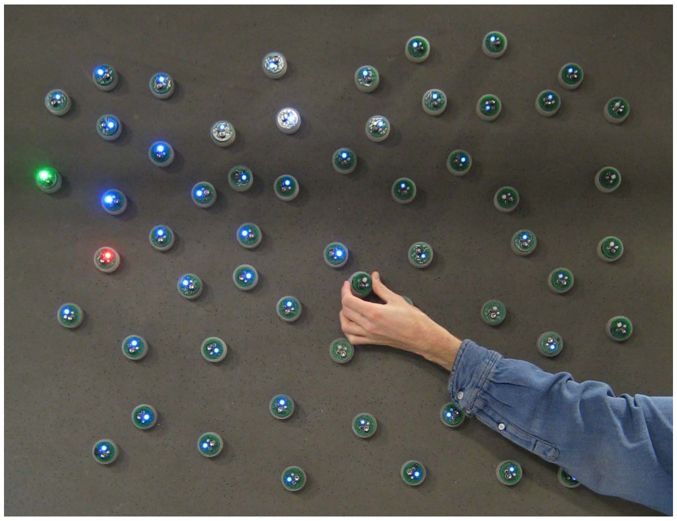


Fig. 3. The 1.2-m by 1.2-m foam substrate into which Pushpins are inserted provides power and ground connections, alleviating the need for batteries. As the name suggests, Pushpins can be inserted and extracted from the substrate as easily as a thumbtack from a corkboard.

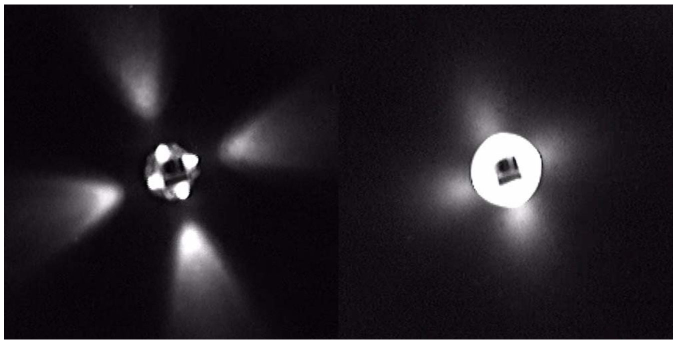


Fig. 4. Images taken from the output of the IR-sensitive video camera monitoring the Pushpin network. The left image is a top-down view of a Pushpin transmitting on all four IR channels without an IR diffusing ring. The right image is the same Pushpin with an IR diffusing ring added to create a more omnidirectional and confined communication zone.

and make electrical contact with two conductive planes (power and ground). See Figure 3. This setup requires very little maintenance (e.g., no batteries to change) and allows for up to 100 Pushpins to be arbitrarily placed on a planar surface and remain within arm's reach of the developers.

Pushpins communicate at 96-kbps using infrared communication hardware on the infrared (IR) communication module, where an IR transceiver points in each of the four cardinal directions, thereby enabling communication with neighboring nodes. The transmit lines of the four transceivers are tied together to a standard hardware UART transmit pin, whereas the receive lines are multiplexed to the same UART's receive pin, allowing each receiver to be independently accessed. Precise received signal strength information is provided by external interrupt pins attached to each of the transceivers' receive lines and relies on the fact the duration of a received IR pulse is a function of the distance between transmitter and receiver. In order to more evenly disperse the transmitted IR in all directions, a frosted polycarbonate ring is placed around each Pushpin. See Figure 4.

The short range of IR communications makes the Pushpins an exceptionally compact platform that realizes a wireless sensor network. RF communication would be difficult to constrain at this short range; any node broadcasting RF would likely be received by the entire network. Figure 5 illustrates a number of representative IR broadcast ranges, which clearly do not conform to the uniform discs so often assumed in simulation.

The Pushpin processing module contains an 8051-core, 8-bit, 22-MIPS microcontroller made by Silicon Labs (formerly Cygnal) [2]. This processor has 2.25-Kbytes of RAM and 32-Kbytes of non-volatile flash memory. It runs off a 22.1184-MHz external crystal, but can also run off a software-adjustable internal oscillator for low-power operation. The processing module provides a host of on-board digital and analog peripherals to the sensor/actuator expansion module through a 25-pin header. See Figure 2.

In addition to the original three hardware modules just described, the most recent sensor/actuator expansion module is comprised of:

- RGB LED with each color channel individually pulse-width-modulated.
- Electret audio microphone conditioned to provide both the raw signal and an 87-Hz cutoff low-pass filtered envelope of the signal to the ADC.
- 40-kHz ultrasound receiver conditioned to provide a digital software-determined threshold detection signal wired to an ex-

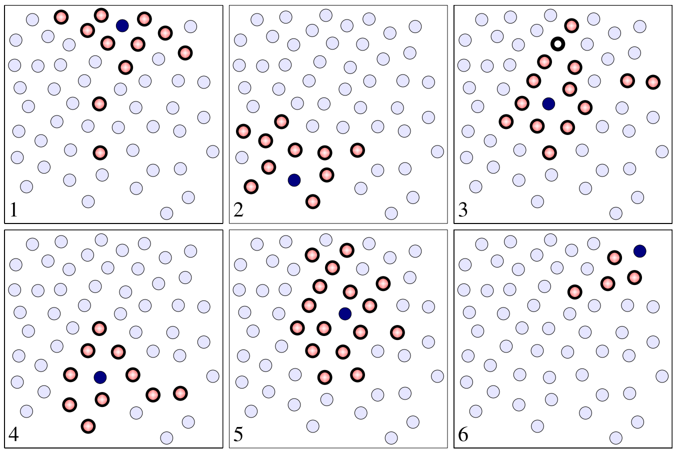


Fig. 5. A schematic representation of actual network data taken from the Pushpins. The neighborhood of nodes with which a Pushpin can communicate varies considerably from Pushpin to Pushpin. The dark, solid circle in each of the above frames indicates a Pushpin that is constantly transmitting packets via IR. The filled circles with bold outlines indicate which of the surrounding nodes reliably receives the packets directly from the originating node. The single unfilled circle with a bold outline in frame three indicates a node that sporadically enjoyed good reception. Aside from this one exception, all other nodes in the neighborhood received virtually 100% of the transmitted packets. Note that there was no other network traffic and the neighborhoods shown only depict one-way communication. In particular, the transmitter in frame four belongs to the neighborhood of the transmitter in frame one, but not the other way around; clearly, the neighborhoods shrink if two-way communication is required for membership. Each frame is approximately 1 meter on a side.

ternal interrupt. The raw sensor signal and a 400-Hz cutoff low-pass filtered envelope of the signal are wired to the ADC.

- Phototransistor conditioned to provide a high-pass filtered digital flash detection signal to an external interrupt and a raw sensor signal proportional to light intensity to the ADC.

An infrared spotlight illuminates the entire Pushpin substrate to serve as a one-way global communication channel to all Pushpins from a personal computer. The spotlight can be used to simultaneously start or pause the entire network, change run-time parameters, request Pushpins meeting certain constraints to provide visual feedback for diagnostic testing, and upload new code to all Pushpins in parallel. Such commands can be received by a Pushpin via either the standard IR communication module or the phototransistor on the time-of-flight expansion module; the former takes advantage of a mature communication library whereas the latter allows for microsecond-level timing by means of the external interrupt attached to the phototransistor. Using the spotlight, all the Pushpins can be reprogrammed with a new operating system in less than a minute. In essence, the spotlight imparts the Pushpin platform with many of the ease of use characteristic of simulations.

A single gateway Pushpin acts as a two-way communication link between the network and a personal computer. The gateway Pushpin is communicates with neighboring nodes within a restricted local area like any other node in the network. Individual queries can be issued and answered through the gateway Pushpin.

At a very coarse, but extremely useful level, all infrared communication between nodes is monitored visually using a low-end black and white camera that views the entire Pushpin substrate. Figure 4 shows close-ups of freeze frames of the output of the camera.

See Figure 6 for a full view of the Pushpin experimental setup.

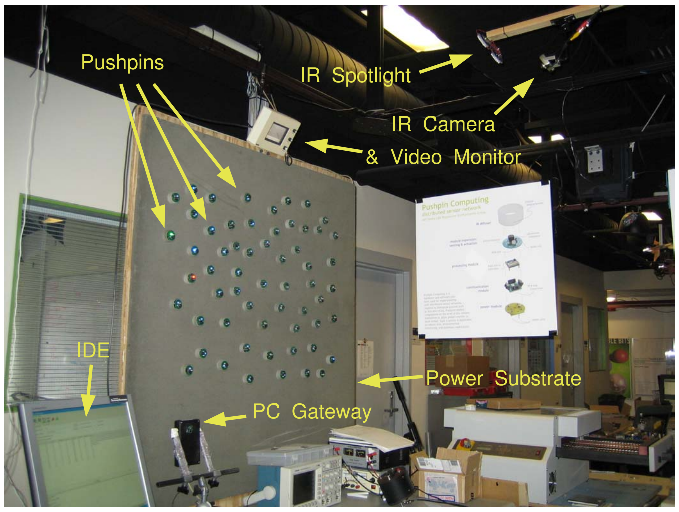


Fig. 6. The Pushpin experimental setup consists of the power substrate containing all the Pushpins, a gateway Pushpin connected to a PC running an integrated development environment (IDE), an IR spotlight for communicating with all Pushpins in parallel, and an infrared-sensitive camera and accompanying video screen to view the entire network's communication patterns.

B. Embedded Software

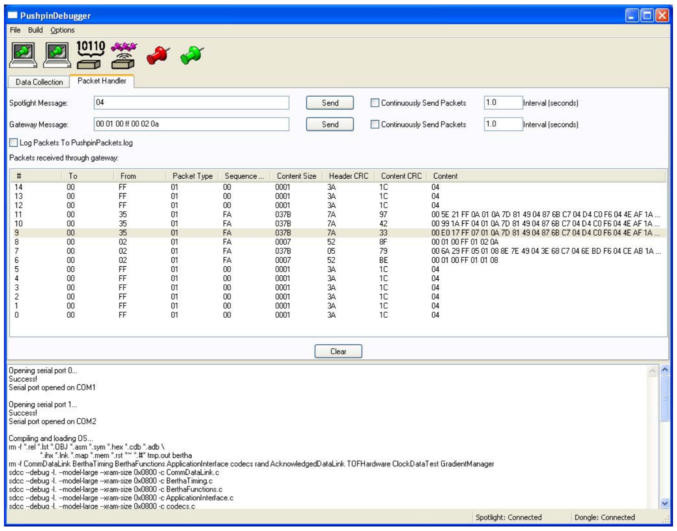
Each Pushpin has a minimal bootloader to receive, verify and load new machine code over the IR communication channel, usually from the IR spotlight. The bootloader can detect and correct communication errors and maintains a versioning system so as to write only new code to memory. On top of this, users have available to them a communications library with mechanisms for unacknowledged and acknowledged packets, gradient diffusion broadcasts, and splitting a payload into multiple packets. Other libraries exist for time-keeping, hardware initialization, sensing, actuation, and other standard microcontroller tasks.

C. Development Tools

The primary language for writing Pushpin programs is the variant of C specified by the Small Devices C Compiler (SDCC), which serves as the underlying toolset for compiling and linking code. Once all Pushpins are equipped with the bootloader, all further development takes place using a custom integrated development environment (IDE) written in Python. The IDE manages code compilation and uploading to the Pushpins over the IR spotlight as well as monitors and logs network traffic over the gateway Pushpin. An arbitrary number of application-specific graphical user interface panels (written in Python using wxPython) can be easily incorporated into the IDE. These tools are open source and cross-platform. Figure 7 shows a screen shot of the IDE.

III. Research Space & History

The initial goal of the Pushpins was to implement and test in hardware many of the ideas put forth by Butera in his Paintable Computing simulation work [3], which revolved around the idea of realizing desired behaviours in sensor networks as the global result of local interaction among many autonomous "process fragments," similar to how global thermodynamic quantities such as pressure and temperature are the result of local interactions among particles governed by the laws of physics. This initial goal influenced many of the Pushpin design decisions. For example, the Pushpins use a microcontroller with a relatively large amount of RAM so as to accommodate many mobile process fragments within a single node. The



Spotlight Message: 04 — Continuously Send Packets [ ] — 1.0 Interval (seconds)

Gateway Message: 00 01 00 ff 00 02 0a — Continuously Send Packets [ ] — 1.0 Interval (seconds)

Log Packets To PushpinPackets.log [ ]

Packets received through gateway:

#	To	From	Packet Type	Sequence ...	Content Size	Header CRC	Content CRC	Content
14	00	FF	01	00	0001	3A	1C	04
13	00	FF	01	00	0001	3A	1C	04
12	00	FF	01	00	0001	3A	1C	04
11	00	35	01	FA	037B	7A	97	00 5E 21 FF 0A 01 0A 7D 81 49 04 87 6B C7 04 D4 C0 F6 04 4E AF 1A ...
10	00	35	01	FA	037B	7A	42	00 99 1A FF 04 01 0A 7D 81 49 04 87 6B C7 04 D4 C0 F6 04 4E AF 1A ...
9	00	35	01	FA	037B	7A	33	00 E0 17 FF 07 01 0A 7D 81 49 04 87 6B C7 04 D4 C0 F6 04 4E AF 1A ...
8	00	02	01	FA	0007	52	8F	00 01 00 FF 01 02 0A
7	00	02	01	FA	037B	05	79	00 6A 29 FF 05 01 08 8E 7E 49 04 3E 68 C7 04 6E BD F6 04 CE AB 1A ...
6	00	02	01	FA	0007	52	BE	00 01 00 FF 01 01 08
5	00	FF	01	00	0001	3A	1C	04
4	00	FF	01	00	0001	3A	1C	04
3	00	FF	01	00	0001	3A	1C	04
2	00	FF	01	00	0001	3A	1C	04
1	00	FF	01	00	0001	3A	1C	04
0	00	FF	01	00	0001	3A	1C	04

```
Opening serial port 0...
Success!
Serial port opened on COM1

Opening serial port 1...
Success!
Serial port opened on COM2

Compiling and loading OS...
rm -f *.rel *.lst *.OBJ *.asm *.sym *.hex *.cdb *.adb \
          *.ihx *.lnk *.map *.mem *.rst *~ *.#* tmp.out bertha
rm -f CommDataLink BerthaTiming BerthaFunctions ApplicationInterface codecs rand AcknowledgedDataLink TOFHardware ClockDataTest GradientManager
sdcc --debug -I. --model-large --xram-size 0x0800 -c CommDataLink.c
sdcc --debug -I. --model-large --xram-size 0x0800 -c BerthaTiming.c
sdcc --debug -I. --model-large --xram-size 0x0800 -c BerthaFunctions.c
sdcc --debug -I. --model-large --xram-size 0x0800 -c ApplicationInterface.c
sdcc --debug -I. --model-large --xram-size 0x0800 -c codecs.c
```

Spotlight: Connected   Dongle: Connected

Fig. 7. The application-specific panels can be easily added to the Pushpin IDE to compliment the standard packet handler, compilation, and data collection interfaces.

two-prong power delivery mechanism was also inspired by Paintable Computing's disregard of energy conservation concerns in favor of concentrating on algorithms and programming models. Finally, the original Pushpin operating system was a direct implementation of the Paintable programming model, complete with blackboard-like state mirroring across neighboring nodes and mobile process fragments. The details of this first instantiation can be found in [1].

Of course, one of the benefits of having an actual sensor network is that it can sense and interact directly with the real world. Accordingly, recent work with the Pushpins has focused on collecting sensor data for both in-network and off-line analysis. In particular, we've experimented with several approaches to localizing the Pushpins based on light, audio, and ultrasound data, using techniques such as lateration, spectral graph drawing, and mesh relaxation [4], [5].

The small physical scale of the Pushpins suggests another direction of sensor network research, namely scaling down sensor networks to the point where they resemble something like biological skin in terms of sensor density. Although nowhere near actual skin sensor density, the Tribble project made use of the Pushpins to realize a 32-patch electronic skin with each patch instrumented with vibration-sensitive whiskers, photodetectors, microphones, pressure sensors, and temperature sensors [6]. See Figure 8. The patches comprising the spherically shaped Tribble communicated neighbor-to-neighbor to route high-level representations of local sensor data and coordinate an appropriate reaction using each patch's local RGB LED, speaker, and vibrating motor. An overview of our vision of sensate skin is given in [7].

In addition to use within our own research group, Pushpins have been used by others in a variety of ways. For example, a battery power module, radio communications module, speaker actuation module, and magnetic field sensing module, among others, have been developed by the Viral Communication Group at the MIT Media Lab in order to implement, test or model a variety of ad-hoc wireless networks [8], [9]. A group in the MIT Department of Physics has used modified Pushpins as a distributed data collection network in an experimental particle detector [10]. Pushpins have also individually been used as a prototyping tool for a large variety of projects unrelated to sensor networks.

Although the Pushpin platform is a research facility, it has also inspired other groups to build applications in ubiquitous computing around networked pushpin interfaces, e.g., [11].

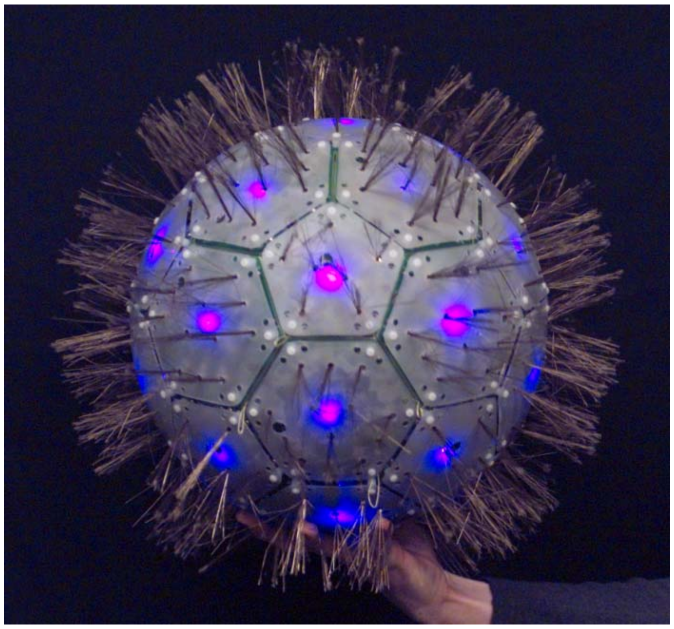


Fig. 8. The Tribble, a sphere tiled with a peer-to-peer wired sensor network, is a coarse manifestation of how sensor networks might be scaled down to form 'electronic skin.'

IV. Lessons Learned

This section chronicles, in no particular order, some of the many lessons we learned in building and using the Pushpin platform.

A. *Connecting Simulation & Reality*

Having initially undertaken the Pushpin project as an implementation of the Paintable simulation, the differences between simulation and reality very quickly became apparent. While some simulations are better than others, no matter how detailed the simulation, there will always be differences with reality. From a practical standpoint, there are three approximately distinct categories of real-world limitations from which such differences arise: energy (e.g., finite absolute quantity or finite consumption rate), communication (e.g., finite capacity), and sensor data (e.g., non-zero noise and finite resolution).

When building a hardware platform from scratch, it is surprisingly easy to get caught up in attempting to simultaneously and optimally overcome all these limitations. While this may result in a more robust platform suited for deployment in the real world, the benefits of overcoming all these limitations at once generally require the kind of time and effort antithetical to quick prototyping principles. With this in mind, the Pushpin platform was designed to overcome real-world limitations only so far as doing so makes for a more realistic simulation-like environment nimble enough to use for quickly prototyping hardware and algorithms. The design decisions and trade-offs are discussed below.

1) Energy Limitations: A large body of wireless sensor network research revolves around energy efficiency – routing schemes, MAC protocols, network maintenance algorithms, and tracking applications are among the many areas that have been optimized for energy efficiency. All of this is important work and essential to creating viable real-world sensor networks. That said, the Pushpins were designed to never have to confront this difficulty. Not having to change or charge batteries or design around strict energy constraints has allowed us to concentrate our efforts elsewhere. This decision has played out well in that the problems associated with limited energy resources can be easily isolated or simulated. For example, rather than determine the energy load by the frequency of battery charges, we can accumulate statistics of node performance to infer the Pushpins' power drain. In general, unless the main thrust of a project is energy efficiency, we've found it best to abstract the problem away and worry about more central aspects of the project.

2) Communication Limitations: Unlike energy limitations, the details and effects of communication within a wireless sensor network are not so easily decoupled from other problems, and therefore do not so readily lend themselves to accurate simulation and analytic methods.

Whereas most wireless sensor network platforms take wireless to mean radio, the Pushpins use infrared. Infrared is more easily confined to short distances and therefore lends itself to building multi-hop networks on a physical scale that would otherwise be a one-hop network if radio were used. This small scale facilitates benchtop experiments in which the network can be stimulated in a controlled manner, such as projecting an image on top of it or precisely placing acoustic sources around it. Fading as seen in RF communication could even be crudely emulated by introducing smoke or fog. Infrared is also more readily debugged – the infrared-sensitive black and white camera monitoring the Pushpins has been invaluable to quickly identifying gross communication patterns and malfunctioning hardware.

The specifics of RF undeniably affect the outcome of certain experiments, and may themselves be the focus of those experiments. However, there is also significant overlap in the challenges associated with RF and IR communication, as evidenced by the large body of research concerning network protocol layers above the physical layer. For example, medium access control must be addressed regardless of the choice of IR or RF.

More generally, a fundamental characteristic of sensor networks is that the nodes communicate with a small (relative to total network size) set of nearby nodes. The exact communication channel used is important, but secondary nonetheless. Even the term ”wireless” is somewhat misleading; a wired sensor network can also be constrained to nearest neighbors, as in Tribble. Although the wireless case is clearly more difficult, many of the data collection and routing algorithms developed for wireless sensor networks apply equally well in the wired case.

3) Sensing Limitations: Real-world sensing presents a host of problems difficult to emulate in simulation, such as sensor calibration and noisy signal sources. For this reason, we've found that even simply treating the Pushpins as a collection of sensors that individually log data for off-line analysis can be of great value for building intuition about sensing problems and formulating possible solutions. The ability to trivially synchronize all Pushpins according to the IR spotlight is especially useful in this respect.

B. Mobile Processes & Virtual Machines

As mentioned in Section III, the first fully functioning version of the Pushpins supported mobile processes, where we loosely define a process as a piece of executable code and a set of persistent state variables. A single copy of a mobile process was confined to a single Pushpin, but could copy itself to other Pushpins and interact with other processes, both locally and remotely via mirrored blackboards, in order to form more complex processes spread over many nodes and composed of many mobile processes. The original implementation of mobile processes relied on an underlying operating system to transfer machine code between nodes. Mobile processes were created by compiling C source code into native machine code, injecting the process into the Pushpin network through a gateway node, and letting it autonomously propagate through the network.

This scheme worked, but suffered from two intimately connected inefficiencies. Firstly, the semantic and syntactic mismatch between the C programming language and mobile processes made development somewhat cumbersome and prone to error. Similarly, mismatch between microcontroller machine operations and the high-level operations that form the basis of mobile processes made for mobile processes containing a large amount of machine code. Given that all this code must be transferred between nodes and that communication is typically the largest energy sink for energy constrained nodes, this scheme for mobile processes would not generalize well beyond the Pushpins.

We believe the solution to both these problems is to develop a programming language and associated compiler-virtual machine pair more suitable for sensor network applications. Some progress has already been made in this area [12], [13]. Virtual machine primitives should be able to easily and concisely represent expressions such as “propagate to all neighbors with vibration sensor reading greater than 23” or “route a message back to a query source with priority 3.” The syntax employed by the Python language for list comprehensions may be well suited for such tasks [14].

C. Inexpensive Sensor Networks

The cost of parts alone for a single Pushpin is approximately $50 in quantities of 100. This high per-node cost reflects the considerable onboard resources and flexibility of the Pushpins, but it also limits the number of nodes in the network. Wireless sensor network hardware research in general seems to favor feature-rich nodes over quantity of nodes, yet many of the algorithms and simulations being researched rely on asymptotic behavior as the number of nodes increases. Therefore, we believe there is a strong case for developing wireless sensor networks composed of up to hundreds of thousands of inexpensive nodes. Our research group has already developed “dumb” sensors capable of transmitting a single bit over a 300-MHz radio when activated by very slight vibration [15]. In large quantities, we project the cost per node to fall under a dollar. Augmenting these nodes with simple radio receivers and a reprogrammable microcontroller would make for a full-fledged and inexpensive wireless sensor network. Not only would this allow for larger sensor networks, but it would also put real sensor networks within the financial grasp of more researchers.

D. Development Tool Set

In the beginning of the development of the Pushpins, we used a commercial IDE and commercial compiler, linker, and assembler suite. As stated in Section II-C, we have since migrated to entirely open source software for primary Pushpin development. This move has granted us a great deal more flexibility in what we can do and how we can do it. We have also taken care to maintain platform neutrality in the sense that primary Pushpin development can be carried out on the major operating systems (GNU/Linux, Mac OS X, and Microsoft Windows). Demand for such an integrated development environment for sensor networks and microprocessors in general appears high enough that we are considering releasing a generic version of our tool set.

E. Absolute Scale & Tiered Sensor Networks

There is a certain ideological appeal to envisioning sensor networks as composed of n identical nodes, where the application under consideration only improves as n increases, allowing for arbitrarily scalable networks. However, in working with a real sensor network, we were constantly reminded of the physical world's strong scale dependence. At the most fundamental level, the physical world must conform to the absolute scales imposed by constants such as the speed of light, an electron's charge, and Planck's constant. These constants in turn determine macroscopic physical characteristcs, such as propagation speeds and attenuation rates, of the physical phenomena of interest to a sensor network. In short, many, if not most, sensing tasks have well-defined spatial and temporal scales, thus dispelling the need or desire for arbitrarily scalable sensor networks. In place of arbitrary scalability, we found tiered hierarchy with scalability within each tier to be more fitting a design point. This is not to say scalability isn't of concern, but rather that scalability within certain bounds and arranged in a hierarchy may be more appropriate. Many aspects of biology, especially vertabrate nervous systems, seem to follow this principle.

From a practical standpoint, this means the absolute scales and tiers of the hierarchy should be made explicit when designing distributed sensor network algorithms (e.g., leader election). It further suggests a sensor network hardware design that explicitly incorporates nodes of varying abilities and resources from the very beginning instead of as an afterthought. We intend to pursue these lines in future work.

F. Sensor Network User Interface (SNUI)

In working with an actual hardware platform, the problem of usability becomes apparent. Without a simple user interface, wireless sensor networks will remain relegated to use only by experts, very much like computers were until the graphical user interface (GUI) opened the field to non-experts and led the way to widespread use of personal computers. Indeed, a sensor network user interface (SNUI) is needed to allow non-expert users to retrieve data from and enter information into sensor networks. One clear path toward a SNUI is to bridge the gap between wearable computers (some of which could themselves be considered sensor networks) and wireless sensor networks.

Rather than consider SNUIs as an eventuality, it may behoove the research community to consider them as a priority, if for no other reason than SNUIs will allow non-experts to help explore the application space of wireless sensor networks. As with so many other technologies, there is no guarantee that wireless sensor networks' killer applications will be discovered solely by researchers.

G. Actuation

Sensing is only half the game. The other half is actuation. Transitioning from wireless sensor networks to wireless sensor actuator networks is a natural step toward creating distributed closed-loop control systems capable of regulating themselves and responding to their environment. In addition, actuation will likely play a role in developing the SNUIs discussed in Section IV-F. In the context of the Pushpins, we've found the RGB LED on each node to be the single most useful feedback mechanism we have to judge the state of the network. In the context of the Tribble, sound plays an equally important role. In addition, our research group has also implemented a separate hardware platform for experimenting with *parasitic mobility*, the idea that mobile nodes in a sensor network can conserve energy by opportunistically attaching to and releasing from people or animals as they pass through the network [16].

The seeming inevitability of enabling sensor networks with actuators also introduces the concept of *self-calibration* (as opposed to *auto-calibration*, which refers to calibration without reference to a ground truth). In self-calibration, a node acts upon itself and its neighborhood to provide a ground truth and then calibrates accordingly. This method of calibration assumes matched sensor/actuator modalities and pre-calibration describing how a node's actuator affects the matched sensor. Where possible, self-calibration is perhaps the easiest method of re-calibration.

V. Conclusions

We have given a broad overview of the Pushpin Computing wireless sensor network platform and recounted some of the lessons learned, including possible future research directions such virtual machines, low-cost sensor network platforms, open development tool sets, hierarchical sensor networks, SNUIs, and sensor actuator networks.

VI. Acknowledgements

We would like to thank the Things That Think consortium and other sponsors of the MIT Media Lab for their generous support. Portions of this work are supported by National Science Foundation grant #ECS-0225492.

References

[1] J. Lifton, D. Seetharam, M. Broxton, and J. A. Paradiso, "Pushpin Computing System Overview: a Platform for Distributed, Embedded, Ubiquitous Sensor Networks," in *Proceedings of the International Conference on Pervasive Computing*, 2002.

[2] Silicon Laboratories, "Silicon Laboratories Homepage," `http://www.silabs.com`, 2003.

[3] W. Butera, "Programming a Paintable Computer," Ph.D. dissertation, Massachusetts Institute of Technology, 2002.

[4] M. Broxton, J. Lifton, and J. A. Paradiso, "Localizing a Sensor Network via Collaborative Processing of Global Stimuli," in *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, January 2005, pp. 321 – 332.

[5] M. Broxton, "Localization and Sensing Applications in the Pushpin Computing Network," Master's thesis, Massachusetts Institute of Technology, 2005.

[6] J. Lifton, M. Broxton, and J. A. Paradiso, "Distributed sensor networks as sensate skin," in *Proceedings of IEEE Sensors 2003*, vol. 2, October 2003, pp. 743 – 747.

[7] J. A. Paradiso, J. Lifton, and M. Broxton, "Sensate Media – multimodal electronic skins as dense sensor networks," *BT Technology Journal*, vol. 22, no. 4, October 2004.

[8] J. I. Silber, "Cooperative Communication Protocol for Wireless Ad-hoc Networks," Master's thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 2002.

[9] A. Bletsas and A. Lippman, "Natural Spontaneous Order in Wireless Sensor Networks: Time Synchronization Based on Entrainment," Viral Communication Group, MIT Media Lab, Tech. Rep., December 2003.

[10] A. Werner, "A self-triggered readout for a time projection chamber," B.S. thesis, Massachusetts Institute of Technology, February 2004.

[11] K. V. Laerhoven, A. Schmidt, and H.-W. Gellersen, "Pin&Play: Networking Objects through Pins," in *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*. Springer-Verlag, 2002, pp. 219–228.

[12] D. Seetharam, "c@t: A language for programming massively distributed embedded systems," Master's thesis, Media Arts & Sciences, Massachusetts Institute of Technology, September 2002.

[13] P. Levis and D. Culler, "Mate: A tiny virtual machine for sensor networks," in *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, October 2002.

[14] G. van Rossum and F. L. D. Jr., "Python tutorial, release 2.3.4," `http://www.python.org/doc/2.3.4/tut/`, May 2004.

[15] M. Feldmeier and J. A. Paradiso, "Giveaway wireless sensors for large-group interaction," in *Proceedings of the ACM Conference on Human Factors and Computing Systems (CHI 2004), Extended Abstracts*, April 2004, pp. 1291–1292.

[16] M. Laibowitz and J. A. Paradiso, "Parasitic mobility for pervasive sensor networks," in *Proceedings of the Third International Conference on Pervasive Computing (Pervasive 2005)*. Springer-Verlag, May 2005.