From Motes to Java Stamps: Smart Sensor Network Testbeds

Thomas C. Henderson, Jong-Chun Park, Nate Smith and Richard Wright

UUCS-03-003

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

March 3, 2003

Abstract

We have proposed Smart Sensor Networks (S-Nets) as an architecture and set of distributed algorithms to extract, interpret and exploit networked sensor devices. Heretofore, the development of this approach has been done in simulation. In this paper, we describe two complementary implementations of S-Nets: (1) on a set of Berkeley motes comprised of low-power 8-bit, 128Kb memory processors, communication devices and sensors, and (2) on a set of JStamps having 32-bit controllers, 2Mb of memory and native execution Java hardware.

1 Introduction

Sensor networks have received increasing attention over the last few years. For example, DARPA's SensIT program envisioned fields of cheap, long-lived, networked sensor devices. David Culler's work on sensor networks explores the rich design space of low-power processors, communication devices and sensors. NSF has recently funded an STC Center for Embedded Network Systems headed by Deborah Estrin that will develop algorithms for wireless and distributed sensing systems.

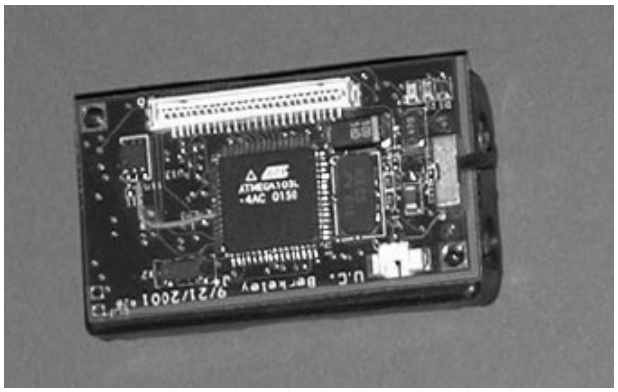

Figure 1: Berkeley Mote

Some examples of issues addressed by these various projects include: power minimization [14, 17], self-configuration [2, 10], data handling [1, 9, 11], systems issues [5, 13, 18], and fault tolerance [18]. In general, higher-level exploitation of sensor networks applies standard sequential or distributed algorithms to the data. Some work in this area includes calibration [16] and habitat monitoring [12].

Our own work started in the late 90's [7], and has mainly addressed the creation of an information layer on top of the sensor nodes. This includes distributed algorithms for leadership protocols, coordinate frame and gradient calculation, reaction-diffusion pattern formation, and level set methods to compute shortest paths through the net [3, 4, 6].

Exploiting sensor networks involves understanding algorithmic and engineering issues of real-world devices, and making both raw and processed data readily accessible to humans. In this paper we describe our first results in the implementation of the S-Net algorithms. We have chosen two complementary domains: Berkeley motes and JStamp embedded processors. We give the layout and results of running our distributed leadership protocol to establish clusters of devices, and a local coordinate frame algorithm which runs in each cluster.

2 Berkeley Motes

We have developed one implementation in a set of four Berkeley motes. Figure 1 shows one of the Mica nodes [8]. The device features an 8-bit Atmega 103 Microcontroller (4 MHz) with 4 Kb system RAM, 128 Kb flash program memory, 8 channel, 10-bit ADC and 3 hardware timers. For I/O it has one external UART, one SPI port and 48 general purpose

I/O lines. It has an AT90LS2343 microcontroller coprocessor for wireless communication, and a DS2401 unique ID device. It has RF range of up to tens of meters at rates up to 115Kb/s. A Maxim1678 DC-DC converter provides a solid 3V supply operated off a pair of AA batteries. There is an expansion connector I/O system interface which allows a variety of sensing boards. Finally, the mote runs the TinyOS multithreading event-based operating system, and applications are written in NesC; NesC is a C-like language that was developed by the Berkeley group just for the purpose of embedded system applications like sensor networks.

2.1 Leadership Protocol in the Berkeley Motes

The S-Net leadership protocol has been described in [6]. Basically, it consists of two phases:

1. **Phase I**: Broadcast ID and receive other broadcast ID's
2. **Phase II**: determine if leader (or not) and broadcast cluster (or re-broadcast cluster)

The protocol was developed in NesC and the configuration file is:

```
configuration SandR {}
implementation {
  components Main, SandRM, RadioCRCPacket as Comm, UARTNoCRC-
Packet,
  ClockC, LedsC;

  Main.StdControl -> SandRM;

SandRM.UARTControl-> UARTNoCRCPacket;
SandRM.UARTSend-> UARTNoCRCPacket;
SandRM.UARTReceive-> UARTNoCRCPacket;

  SandRM.RadioControl -> Comm;
  SandRM.RadioSend -> Comm;
  SandRM.RadioReceive -> Comm;

  SandRM.Clock -> ClockC;
```

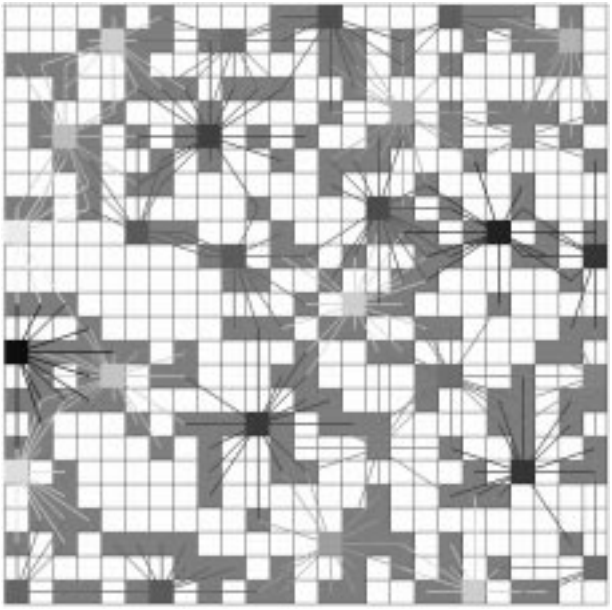

Figure 2: 250 Mote Leadership Solution (from Mote Simulator)

```
  SandRM.Leds -> LedsC;
}
```

The code was developed first in the Mote simulator, and Figure 2 shows a 250-node leadership solution. The gray squares have devices and the variable gray level squares are leaders. The edges show communication connectivity.

In the mote implementation, the leadership code takes 14.3Kb memory. A delay of 2 seconds is set for Phase I to allow neighbors lists to be built. Figure 3 show four motes which have run the protocol; leader motes have the red LED illuminated. (The leader motes are the left and right motes which are not in each others broadcast range;they both can communicate with the middle two motes.)

We have also developed algorithms to compute a coordinate frame for a cluster. Figure 4 shows a 250-mote simulation result with the local frames shown. We have run the code on the 4 motes and produced correct frames for them as well; the coordinate frame executable takes 21Kb.

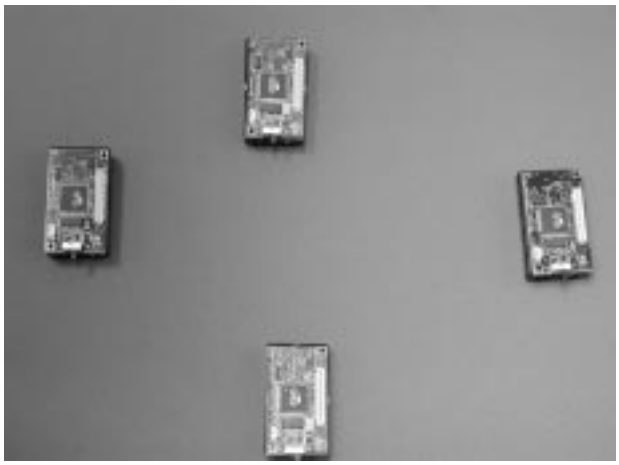

Figure 3: 4-Mote Leadership Solution (red LED means leader)

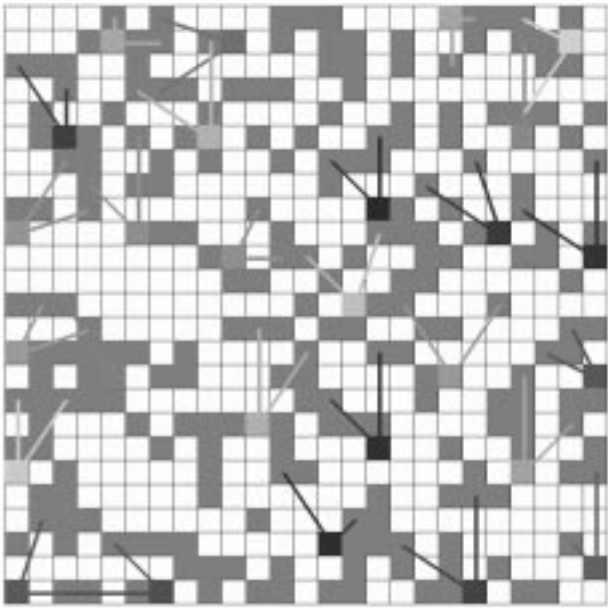

Figure 4: 250-Mote Coordinate Frames Calculation (mote simulation)

Figure 5: Systronix JStamp Processor

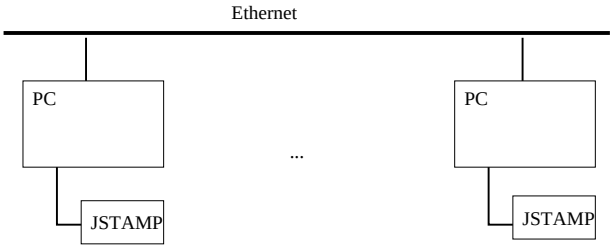

Figure 6: JStamp Testbed Layout

3 JStamp Processors

We have also implemented the S-Net algorithms in Systronix JStamps (see Figure 5). There are many benefits to using Java as the programming language, and the JStamp or JStik as the controller hardware. JStamp and JStik are physically small (JStamp is only 1x2 inches), yet contain a 32-bit controller, 2 Mbytes of memory, and the rich constructs of Java. Software can be developed in Java on PCs and then easily loaded onto the nodes. Another huge benefit of Java is the robust and proven security models designed into the Java language and JXTA. Native execution Java hardware is physically small, very power efficient, and computationally powerful. For example, the 1x2 inch JStamp can run off a standard 9V transistor battery for up to 40 hours, and execute three million Java byte codes per second. Systronix is currently the world leader in the commercial development of such modules.

Of course, sensor networks do not always require wireless connectivity, and our current JStamp testbed is set up as shown in Figure 6. Each JStamp in the testbed has an RS232 connection to a PC, and the PCs are connected through Ethernet. (If we use JStiks instead of

JStamps, they have their own Ethernet ports and eliminate the need for PCs. RF capability for JStamps/JStiks is also under development by Systronix.)

Independent processes are run on each PS which handle the communication between JStamps; these processes connect to each other through sockets. The S-Net leadership protocol and coordinate frame algorithm have been implemented in the JStamp testbed with no problems encountered. There is an effect in setting timer values in the leadership protocol which is a critical issue in energy awareness in S-Nets.

The leadership and coordinate frame executable takes 133.4Kb memory. Here is a partial trace of an execution of the coordiante frame calculation.

```
< On PC >

Test For Global Frame:

Given three points and distances to each known point,
we can calculate the coordinates of an unknown point.

Enter <x0, y0> for P0: 0 0
Enter <x1, y1> for P1: 3 0
Enter <x2, y2> for P2: 6 8
Enter a distance to point P0: 5
Enter a distance to point P1: 4
Enter a distance to point P2: 5

The unknown point is <3.0, 4.0>.
  << Data from J-Stamp

Test For Local Frame:

Given two known points and three dis-
tances from an unknown point, we can
calculate the coordinates of the
unknown point. And then we can also
calculate the coordinates of an unknown
point in this local frame with three
distances to three known points.
```

```
Enter a distance d12: 3
Enter a distance d23: 4
Enter a distance d13: 5
Enter a distance to a point
  P0 from an unknown point: 10
Enter a distance to a point
  P1 from an unknown point:
     8.544003745317531167 87
Enter a distance to a point
  P2 from an unknown point: 5

The third point is <3.0, 4.0>.
       << Data from J-Stamp
The unknown point is <
  6.000000000000003,
  7.999999999999997>.
 << Data from J-Stamp

Test For Converting Frame

Given two points from the old and new
frame, we can calculate the displace-
ment and angle rotated between two
frames. Thus, we can convert the point
of the new frame to the corresponding
one of the old frame.

Enter <x0, y0> : 0 0
Enter <x0', y0'> : 0 0
Enter <x1, y1> : 1 0
Enter <x1', y1'> : 0 -1
Enter <x1', y1'> : 5 6

The origin of the new frame has moved
to <0.0, 0.0> with angle
1.5707963267948966 radian rotated.
The point corresponds to
<-5.999999999999999,
4.999999999999999> of the old frame.
 << Data from J-Stamp
```

```
< On J-Stamp >

[TEXTIO.0]->TEST FOR GLOBAL FRAMES:
[TEXTIO.0]->  Given three points and
 distances to each from a known point,
[TEXTIO.0]->we can calculate the
  coordinates of the unknown point.
[TEXTIO.0]->...processing...
[TEXTIO.0]->...done...
[TEXTIO.0]->The calculated coordinates
  of the unknown points are sent to
  the requester.
[TEXTIO.0]->TEST FOR LOCAL FRAMES:
[TEXTIO.0]->  Given two points and
  distances among them, we can calculate
[TEXTIO.0]->the third point. And then
  we can caculate the coordinates of a
[TEXTIO.0]->unknown point to this local
  frame provided the distance to each.
[TEXTIO.0]->...processing...
[TEXTIO.0]->...done...
[TEXTIO.0]->The calculated coordinates
  of the unknown points are sent to
  the requester.
[TEXTIO.0]->TEST FOR CONVERT FRAMES:
[TEXTIO.0]->we can convert the
  coordinates of any point between
  any two frames.
[TEXTIO.0]->...processing...
[TEXTIO.0]->...done...
[TEXTIO.0]->The calculated coordinates
  of the unknown points are sent to
  the requester.
```

4 Conclusions and Future Work

These initial results of actual implementations of the S-Net algorithms are very encouraging. The leadership protocol and coordinate frame algorithm are the basis for most of the

other algorithms we are implementing; e.g., gradient calculation, reaction-diffusion, and level set calculations. We hope to be able to report on these other algorithms soon.

As far as comparing the two implementation testbeds, they have very complementary features. First, the Berkeley motes offer:

- small size
- low cost
- low power
- RF
- simulation environment

Mote cons include:

- small memory
- new programming language (NesC)
- differences between simulator and mote codes
- difficult to debug motes

The major issue in learning NesC is getting the communications aspects correct. In addition, there are some problems with shoehorning codes into the simulator (specified node connections may not occur in the simulator). In the actual motes, new batteries need to be used for benchmarking and testing to get consistent results. Moreover, the clock setting influences the correctness of the leadership protocol: set to 32 ticks/sec is really good; 64 ticks/sec results in failure about half the time, and 100 ticks/sec leads to high failure rates. In addition, delay timings are crucial for Phase I of the leadership protocol. Finally, simple acknowledgements in the frame algorithm led to more accurate results (angles between devices, etc.).

The JStamp testbed offers:

- Java programming

- off-stamp debugging
- small size
- low power
- large memory
- permits large memory sensors (e.g., CMUCam).

JStamp cons are:

- no RF
- no simulator for testbed

We hope to develop a large, heterogeneous mote and JStamp/JStik system (100 or so nodes) to be used with actual sensors. In addition, we are looking at providing the sensed data through interfaces to the leaf nodes in the Utah Emulab. (Emulab [15] is a universally-available time- and space-shared network emulator. Several hundred PCs in racks, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts of a Java GUI, allow you to remotely configure and control machines and links down to the hardware level. Packet loss, latency, bandwidth, queue sizes can be user-defined.) Data synthesis and analysis can be readily written in Java, as well, and then run on Utah's Emulab nodes. Other possible data sources include a wearable human terminal, or other monitoring nodes such as notebook PCs with wireless adapters.

We hope to explore human – S-Net interfaces where each node in the sensor network could have sensors capable of detecting motion of people; (e.g., a camera, microphones, pyroelectric motion detectors, ultrasound, or other such sensors. Camera and sound sensors would benefit from the JStik HSIO bus. Other sensor options include Dallas 1-Wire, I2C and SPI devices. Many such off the shelf devices exist and JStamp products can already accommodate them. The sensor nodes may be wireless. At least two possibilities exist: 916 MHz RF modems, and Bluetooth. Both RF devices could use the JXTA protocols, and there is already a Java implementation of JXTA. JXTA is media agnostic and could accommodate a mixed network of wired ethernet, 916 MHz RF modems and Bluetooth.

For emulation experiments the S-Net nodes may be attached to Emulab nodes. With a serial-to-1Wire adapter and some 1Wire devices, the Emulab node could synthesize signals which would appear to the sensor nodes to be actual sensor data. This would allow some

very interesting and rapid testing. The 1Wire Emulab test signal adapter would be limited to low bandwidth signals (a few tens of Hz typically). The advantage is the low cost of each such adapter. The adapters could be programmed via a Java application running on the Emulab node in a standard JVM.

Another option is a portable, wearable, human interface and display. This would consist of a small glasses-mounted virtual display terminal (e.g., this kind of device is available from Micro Optical Displays). This display creates a virtual screen that appears to float in the air about a meter in front of the wearer. The display has a serial interface and can be driven from JStamp or JStik. It can display simple monochrome block graphics and text. Models with color and gray scale are also available. The wearer could input data into the portable terminal by using a small handheld trackball/mouse, or a data glove. This display, driven by a battery-powered JStamp node, would communicate with the sensor network by means of the same RF transceivers used in the sensor nodes. The total system including batteries weighs as little as one pound (depending on the duration of the batteries). A user wearing the system could in fact be an interactive part of the sensor network and could be guided safely through the sensor network.

References

[1] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc of the Second Intntl Conf on Mobile Data Management*, Hong Kong, January 2001.

[2] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In *Proc. Sixth International Symposium on Communication Theory and Applications (ISCTA '01)*, Ambleside, Lake District, UK, July 2001.

[3] Y. Chen. Snets: Smart sensor networks. Master's thesis, University of Utah, Salt Lake City, Utah, December 2000.

[4] Y. Chen and T. C. Henderson. S-nets: Smart sensor networks. In *Proc International Symposium on Experimental Robotics*, pages 85–94, Hawaii, December 2000.

[5] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. *Mobile Computing and Communications Review*, 1(2), 2002.

[6] T. C. Henderson. Leadership protocol for s-nets. In *Proc Multisensor Fusion and Integration*, pages 289–292, Baden-Baden, Germany, August 2001.

[7] T. C. Henderson, M. Dekhil, S. Morris, Y. Chen, and W. B. Thompson. Smart sensor snow. *IEEE Conference on Intelligent Robots and Intelligent Systems*, October 1998.

[8] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization. Ece, UC Berkeley, October 2002.

[9] T. Imielinski and S. Goel. Dataspace - querying and monitoring deeply networked collections in physical space. In *Proc. of International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'99)*, Seattle, WA, August 1999.

[10] A. Lim. Support for reliability in self-organizing sensor network. In *Proc of the Intnl Conf on Information Fusion*, Annapolis, Maryland, July 2002.

[11] S. Madden, M. Franklin, and J. Hellerstein. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002. USENIX Association.

[12] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor netwroks for habitat monitoring. In *WSNA 2002*, Atlanta, GA, September 2002.

[13] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept 2002.

[14] V. Swaminathan, K. Chakrabarty, and S. Iyengar. Dynamic i/o power management for hard real-time system. In *Proc. Intl. Symposium on Hardware/Software Co-Design (CODES*, pages pp. 237–242, Ambleside, Lake District, UK, 2001.

[15] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[16] K. Whitehouse and D. Culler. Calibration as parameter estimation in sensor networks. In *Proc. WSNA 2002*, Atlanta, September 2002.

[17] Y. Yemini, S. da Silva, D. Florissi, and H. Huang. The network flow language: A mark-based approach to active networks. Computer Science XXX, Columbia University, July 1999.

[18] L. Zhang. Simple protocols, complex behavior. In *Proc. IPAM Large-Scale Communication Networks Workshop*, March 2002.