**Title**
Experiences with the Extensible Sensing System ESS



**Authors**
Richard Guy
Ben Greenstein
John Hicks
et al.

**Publication Date**
2006

# Experiences with the Extensible Sensing System ESS

Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan,
Tom Schoellhammer, Thanos Stathopoulos, Karen Weeks,
Kevin Chang, Lew Girod, Deborah Estrin
UCLA Center for Embedded Network Sensing
3563 Boelter Hall, UCLA
Westwood, CA 90095-1596
01-310-825-3127
rguy@cens.ucla.edu

## ABSTRACT
The Extensible Sensing System (ESS) has been in use for several years in a variety of sensor network deployments. It is a key component of a collection of tools that together are a nearly complete, end-to-end, sensor-to-user facility for deploying and managing a sensor network. This paper provides the context and architectural overview of ESS, along with selected deployment details and a series of lessons learned. Lesson areas include connectivity, interactivity, energy vs. robustness, vertical integration, and real-time visibility. The current version of ESS reflects changes from these lessons; further, new tools are in development that complement ESS.

## Keywords
Sensor networks; field deployment experiences; ESS; CENS.

## 1. INTRODUCTION
Environmental sensing applications commonly share a sampling model in which most nodes in the network are homogenous, in the sense that many nodes will share identical sensors that will all be tasked to sample with the same frequency and at the same time. The data from each sensor will be returned to a 'sink' node, where it is collected and then transferred to an external data repository for archival and analysis.

The Environmental Sensing System (ESS) is a key component of a system that a domain scientist (e.g., botanist, ecologist) would employ for a long-duration study of the environment at a fine-grained spatial and temporal density: a complete environmental data collection, archival, and analysis system. Figure 1 shows a detailed view of such an overall system.

<table>
<tr><td>Domain scientist</td><td rowspan="2">DAS, Excel, MATLAB, gnuplot</td></tr>
<tr><td>Data display/analysis tools</td></tr>
<tr><td>Data management tools</td><td rowspan="2">SensorBase.org</td></tr>
<tr><td>Database host</td></tr>
<tr><td>Internet</td><td rowspan="8">ESS</td></tr>
<tr><td>High-power wireless link</td></tr>
<tr><td>Microserver basestation</td></tr>
<tr><td>Low-power wireless network</td></tr>
<tr><td>Low-power platform</td></tr>
<tr><td>Sensor interface</td></tr>
<tr><td>Sensor</td></tr>
</table>

**Figure 1: Overall sensing system levels and the corresponding software application that manages each level.**

ESS is a software application that spans the lower tiers of this overall system architecture, from sensor to microserver base-station. It provides high-level interfaces for controlling data sampling, transformation, and collection from the sensor network; it also includes lower-level tools such as energy-efficient routing algorithms and sensor interface drivers. A primary goal of ESS is that it be suitable for multi-year deployments.

ESS leverages two primary software development and deployment environments: TinyOS at the low-power platform level and Emstar at the microserver base-station level. TinyOS [1] is an operating environment designed for resource-constrained platforms such as the Crossbow Mica mote family of low-power wireless sensor network processor boards. TinyOS includes a variety of useful tools, including a radio stack for the Mica2's CC1000 20Kbps radio. This stack contains a basic radio driver (ie, physical and link layer protocols) and a tunable energy-efficient medium access control layer (BMAC with low-power listen).

Emstar [2] is a sensor network development environment that primarily supports higher-power platforms that run the Linux operating system. One key Emstar tool is EmTOS, a real-time TinyOS simulator. EmTOS enables the TinyOS application code that normally executes on a mote platform to be simulated on a higher-powered Linux platform, and further allows for easy interfacing to other Emstar services. Emstar also provides easy-to-use hooks into EmTOS to allow for export of TinyOS application control and status interfaces.

In a simple sensor network with a collection of motes and a single micro-server (the common ESS deployment model to date), Emstar's main contributions are EmTOS and interface services to applications external to Emstar. However, in more complex environments, with multiple microservers (either redundantly in one network or as gateways for several networks), Emstar is leveraged much more extensively.

While control of the sensor network, and retrieval of data from the sensors for collection at the microserver base-station, is an essential foundation for a sensor network system, a domain scientist needs much more: she needs the data moved from a remote field-deployed microserver to a database environment that enjoys well-connected access, high-quality power, significant computation resources, backup services, and so on—services routinely available at most institutional computation centers. To this end, ESS is augmented by SensorBase.org, a recent experiment in providing a centralized repository that allows people to easily publish and share a specific domain of environmental sensor network data.

SensorBase.org [3] is a web site and a database designed for ESS-specific environmental sensor network data. It provides users a

uniform and consistent method for publishing sensor network data. It allows users to define data types, groups, and permission levels. It is also a sensor network search engine, which allows users to query for specific data sets based on geographic location, sensor type, date/time range, and other relevant fields. Figure 2 shows the entire schema of just 15 tables.

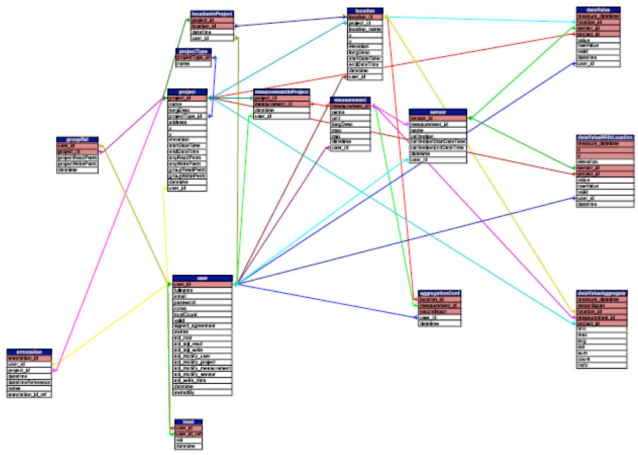

**Figure 2: The SensorBase.org relational database schema.**

SensorBase.org will be used by domain scientists, in particular, along with conventional data analysis tools such as MATLAB, Excel and gnuplot.

Our Deployment Analysis System (DAS) [4] is a web-based tool that interfaces to SensorBase.org and provides useful and concise displays of various aspects of a sensor network deployment. Summary charts and graphs provide easy access to current and historical data, including both domain data and system health data. The latter is extensively used to monitor and diagnose problems. Figure 3 shows a sample DAS display.

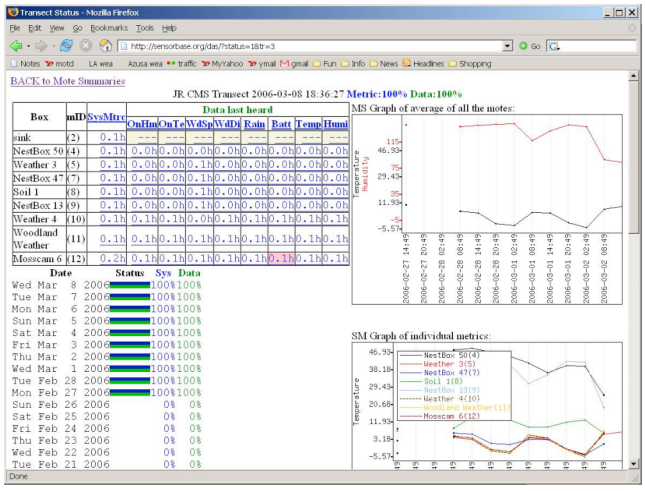


**Figure 3: Sample DAS display.**

In the sequel, we present an overview of the ESS architecture and key components in Section 2, followed in Section 3 by a brief summary of some typical ESS deployments. Section 4 lays out the most important lessons learned from our deployment experiences.

# 2. ESS ARCHITECTURE

ESS is composed of a small number of well-modularized components. These include: standard TinyOS scheduler and CC1000 radio stack (including BMAC/LPL low-power listen); standard Crossbow MDA300 sensor board driver; several experimental message routing services; time management service; persistent data buffer; Sympathy system status service; and the DSE Data Sampling Engine.

## 2.1 DSE

The DSE Data Sampling Engine provides the control and data export interfaces, and thus drives the rest of the application. The DSE sampling model uniquely labels each sensor type and sensor board channel combination. (The vast majority of existing sensors on the market are not self-identifying, so the burden is on the senor deployer to correctly attach the sensor wiring to the desired sensor board channel. DSE is pre-configured with a large number of existing sensors and possible channel mappings.)

DSE supports several classes of queries, including one-time queries (to be executed upon receipt by each node with the appropriate sensors) and periodic queries (executed on receipt, plus infinitely often at the specified interval). A query may be aggregate over several sensor types for concurrent sampling, or specify a single sensor type. DSE attaches a locally-generally timestamp for each data sample. A typical example ESS query specifies a 300 second periodic query interval, sampling a temperature thermistor on channel 0 (a 2.5V excitation reference) and concurrently sampling a relative humidity sensor on channel 1 (a 3.3V excitation reference) and also the (battery) supply voltage to the node. DSE is pre-programmed with the excitation duration and any delay to follow prior to the actual sample.

The control interface also includes features to delete a previously established query, add new sensor configuration parameters, and inspect current query and sensor configuration parameters.

DSE assumes an unreliable broadcast flooding model for query dissemination, and neglible network forwarding delay. Periodic queries are flooded from the microserver at intervals usually much larger than the query's period; nodes that fail to receive the initial query (or have lost the query due to a node reboot after a battery swap, for example) are likely to receive it at some future point.

One important exception to this query dissemination model arises in cases where *no* microserver is within range (directly or indirectly) of a sensor node. For example, subterranean deployment of wireless sensor nodes in a set of caves presents very difficult RF connectivity issues. ESS can be used in this setting by pre-configuring each sensor node with a default query which takes effect immediately at boot time. Another method is to carry a portable microserver for use in initiating queries at deployment, and then 'disconnecting' simply by leaving the area. So-called "lonely motes" have been deployed in the former fashion, using default queries to collect data for several weeks; they were then physically retrieved and placed near a microserver for data upload.

## 2.2 Routing

ESS has several compile-time-selectable routing services, including a basic beacon-based multihop service (multihop), a centralized routing service (centroute), and an advanced distributed routing service (hyper). ESS assumes that the

dominant communication pattern in a sensor network is node-to-sink (carrying data in response to queries), and therefore optimizing routing for that pattern is paramount. All of the ESS routing services create a routing tree used primarily for moving data from sensor nodes to sink node. No point-to-point mechanism exists for sink-to-node communication, under the assumption that the dominant communication paradigm for messages originating at the sink is one-to-all. Although messages from node to sink are point-to-point in that a rooted acyclic path is followed, no other notion of arbitrary node-to-node communication is provided.

The beacon-based multihop routing service uses periodic beacons to assess link quality between nodes and passes that data to the designated sink node, where a routing tree is constructed and disseminated via flooding to the sensor network. Newly booted nodes send out beacons at an accelerated rate, to spur any neighbors to push their (by definition, new) link quality data to the sink node, where a new energy-efficient routing tree will be constructed and disseminated. This service has been in use for nearly two years on a number of deployments, ranging from 2 to 27 nodes each; as with many first-effort routing algorithm implementations, it works well with stable links and less so with very weak critical links.

More recent routing algorithm implementations include centroute and hyper.

Centroute is a *centralized* tree-based routing protocol, in that all control decisions are made in a single point, the microserver (which is also the root of the tree). The protocol uses source routing in addition to a centralized decision point in order to avoid loops. In addition, only constant state is kept on the motes themselves (information about their parent) so the protocol scales well with increasing network density (in contrast, protocols that maintain neighbor lists on each mote have scalability issues when density increases). Centroute is able to maintain higher than 99% connectivity in medium or high-density networks while incurring a low overhead.

The Hyper routing protocol creates routing trees in response to a 'tree formation' message flooded from the sink (root) node. Every node waits a short period to collect path cost estimates from neighbors, selects the lowest cost path as the preferred path to the root, and in turn floods out that estimate. Experimental results suggest that a stable, high-quality tree can be formed in under a second. In addition to building quality routes quickly, Hyper also includes several features to support fast convergence. When a node boots it can quickly assess its neighborhood, graft onto an existing tree, and get time synchronization information from neighbors. These features make deployment and maintenance easy, reducing the amount of time spent per node.

## 2.3 Time Management

Ideally, data samples are timestamped at the instant the sample is taken, with the time provided by a very high-quality time reference. In practice, timestamps suffer from both delay in receiving a reference value and internal clock drift. Sensor network platforms can suffer from an additional malady: the absence of battery-backed clocks in the presence of power interruptions (e.g., primary battery failure) or manual reset. ESS periodically floods a time reference value from sink to nodes, both to limit clock drift on the mote hardware and to ensure that power-cycled nodes receive a valid time value relatively soon. Because most data sample intervals are on the order of tens or hundreds of seconds, accumulated time error resulting from varying hop latencies and drift (on the order of milliseconds, or even tenths of seconds) are insignificant.

## 2.4 Persistent storage

Experience with early versions of ESS in real deployments quickly showed that connectivity in the wild is much worse than in-lab experiences had suggested. In particular, nodes frequently could not maintain sufficient link quality with neighbors to reliably transport data packets across a multi-hop network to the sink. This was exacerbated by limited availability of mote RAM for buffering in-transit data packets. The ESS solution is to use the on-board EEPROM as a persistent data store for packets that aren't immediately acknowledged by an upstream neighbor. This store can hold about 40,000 individual data samples, which is sufficient for nearly two node-weeks of 120 samples/hour.

## 2.5 Sympathy

Sympathy is a prototype tool for detecting and debugging failures in pre- and post-deployment sensor networks. Sympathy has selected metrics that enable efficient failure detection; nodes periodically transmit a subset of these metrics back to a sink, which combines this information with passively-gathered metrics to detect failures and determine their causes. Sympathy also includes a fault-tree algorithm that root-causes failures and localizes their sources in order to reduce overall failure notifications and point the user to a small number of probable causes.

Sympathy gathers and analyzes general system metrics such as nodes' next hops and neighbors. Based on these metrics, it detects which nodes or components have not delivered sufficient data to the sink and infers the causes of these failures.

# 3. DEPLOYMENTS

ESS has been deployed in a range of settings, from forest to farm and botanical garden to Bangladesh. Our longest and largest deployments have been at the University of California's James Reserve [5] in the San Jacinto National Forest near Palm Springs (1.5 years, 20-27 nodes); smaller or younger deployments are present in a farm in the high desert of Palmdale, California (1.5 years, 2 nodes) and at the UCLA Botanical Garden [6] (3 mos, 24 nodes); our most recent and novel deployment was for two weeks in a rice paddy near Dhaka, Bangladesh.

A typical ESS microserver consists of an Intel Stargate, equipped with either a GPRS Sony Ericsson Edge modem card or a 200mW SMC 802.11b card; a solar panel, charge controller and 100Ahr battery; and a 'transceiver' gateway Mica2 mote to bridge to the sensor network array of motes.

A typical ESS node comprises a Crossbow [7] Mica2 mote with 433MHz CC1000 radio; Crossbow MDA300 sensor interface board; and 2-8 sensors. One James Reserve transect of 27 nodes uses simple temperature and relative humidity sensors jointly housed in a solar shielded enclosure at each node; another 10-node transect includes additional sensors for wind, wind gust, rainfall, and soil moisture on many nodes. The Bangladesh experiment involved 12 motes with up to four nitrate, calcium, or phosphate sensors each. The motes are commonly equipped with

a single LiSO2 D-cell at 3.6V or 3.0V; occasionally a pair of 1.5V alkaline cells (AA, C, or D) is used instead. Figure 4 shows a typical micro-climate monitoring node.


**Figure 4: A typical ESS micro-climate monitoring node.**

# 4. LESSONS LEARNED

ESS is now in its third year and second major version. Imperfect assumptions about how sensor networks would be used, and technology change, have promoted improvements.

## 4.1 Connectivity

**Assumption:** A sensor network is dense, implying easy ongoing RF connectivity.

**Reality:** Sensor installation is labor-intensive, encouraging carefully chosen sites; science-driven placement rarely yields quality RF paths; installation is usually done in daylight and good weather and minimal foliage, when RF conditions are best. However, nightfall, precipitation and plant growth happens!

**Impact:** Software at all levels (mote, microserver, DB host) must assume disconnection is normal—for all control, monitoring, and data collection activities; this includes disconnection between microserver and outside world. The three most important impacted issues are tasking, time dissemination, and data buffering. (Neither Mica2 mote nor Stargate have battery-backed clocks; power cycling in isolation yields 'timeless' data, which is often forever useless!)

Another effect is seen in our growing use of "relay motes" which do not service sensors, but simply act to plug network connectivity gaps. These motes are often placed at a higher elevation plane, where fewer obstructions might be found. These additional motes may stress network routing algorithms that have a dependency on neighbor density and overall scale.

## 4.2 Interactivity

**Assumption:** Hardware and software would work in the wild, much as in the lab; so, one could physically deploy a complete network rapidly in a convenient order and later look at the data stream (at microserver or even DB host) to identify problems.

**Reality:** Hardware breaks or is carelessly manufactured or installed; it is very hard to identify the nature of failure at a distance (either physical or time).

**Impact:** Sensor network deployment is by necessity an incremental, interactive, time-consuming activity. The software architecture must support interactive control and monitoring of an individual node while the node is being assembled and installed in the field. This further implies the need for a portable pseudo-microserver to interact temporarily with a node during installation.

## 4.3 Energy vs Robustness

**Assumption:** Energy consumption is always the preeminent concern, therefore any feature should have little or no increased energy impact.

**Reality:** Robustness is more important than energy usage; energy usage focus is easily misapplied. Domain scientists do not want to trade data away for less frequent battery changes—complete data sets over weeks are important.

**Impact:** Perhaps the most potent example of misapplied energy conservation in ESS occurred in the early developers' extensive focus on minimizing packet header and payload overhead, with attendant increases in code complexity or decreases in robustness or functionality. This effort was rendered moot by our later adoption of BMAC/LPL, which in our usage prepends every 40ish byte application packet with a multi-hundred-byte preamble—in this mode, what's another byte or two… or even ten or twenty?

Another energy example is our initial attempt to treat node and network startup phases the same as the long-life phases: long intervals between data samples, slow reconstruction of network connectivity graphs, avoidance of energy-draining LEDs on motes, etc. We have since learned from a plethora of painful initial deployments (and re-deployments) that expending a bounded amount of energy at the outset to flash LEDs as an indication of healthy mote start-up, followed by short-interval sensor sampling and reporting, and rapid connection to neighboring nodes—all are indispensable tools to assure a degree of node and network health while the human team is onsite and able to diagnose and repair problems relatively easily.

## 4.4 Vertical integration

**Assumption:** Providing tools that dealt effectively with the sensor-to-microserver levels (e.g, ESS) was a sufficient contribution to the state-of-the-art to enable successful use by domain scientists; users could easily provided back-end database and display tools.

**Reality:** The pathway from microserver to database is non-trivial; even in our test deployments, ESS developers needed data display tools to reduce the effort in analyzing sensor network system health.

**Impact:** Now positioned between ESS and Sensorbase are a collection of tools that establish and maintain network connections between the two environments. ESS commonly uses either GPRS or 802.11b links between the microserver base-station and a wired network infrastructure. GPRS links in particular are often broken and re-established at lower levels, which often results in non-transparent new IP address assignment. Firewalls outside the control of either 'end' (sensor network or DB facility) also frequently limit flexibility in creating connections from DB end towards sensor network. The result is that tools that robustly push data from microserver to DB host despite frequent connection disruption are essential.

## 4.5 Real-time visibility

**Assumption:** The domain scientist would know, in advance, the desired placement of sensor nodes.

**Reality:** A domain scientist is often working in uncharted experimental territory, and doesn't know initially where best to place sensor nodes. Further, even with data in hand from an initial deployment, she might need to interactively experiment with sensor re-alignment or additional sensor placement.

**Impact:** The scientist needs not merely interactive tools at a node level, but also needs to see node-level data in the context of current and historical data from the extant network as well as other external data sources (e.g., satellite imagery, data from similar experiments at other locations, etc.) Our team is currently prototyping a new tool, Emissary, which leverages ESS, SensorBase.org, DAS, real-time status, and other integrated tools to provide this class of capability to domain scientists.

# 5. SUMMARY

ESS is a maturing tool useful in deploying and maintaining sensor networks for environmental science applications. The current version benefits from a number of important lessons learned over several years of deployments and use in a variety of settings. Lesson areas include connectivity, interactivity, energy and robustness, vertical integration, and real-time visibility.

# 6. ACKNOWLEDGMENTS

ESS is the result of a series of joint efforts over several years; early developers included Eric Osterweil and Ning Xu; routing algorithm developers were Ben Greenstein, Tom Schoellhammer, and Thanos Stathapoulos; DSE was written by Tom Schoellhammer; Sympathy was written by Nithya Ramanathan; the low-level data-sampler was created by Mohammad Rahimi; Rahul Kapur built the persistent data buffer; Kevin Chang fathered SensorBase.org; John Hicks and Karen Weeks provide ongoing feature integration and quality assurance support. Thanks also to our intrepid early adopters, Tom Harmon and John Ewart at UC Merced, Eric Graham and Mike Taggart at James Reserve, and Jenny Jay and Sara Rothenberg in CENS.

This material is based upon work supported by the National Science Foundation under Grant No. CCF-0120778. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# 7. REFERENCES

[1] http://tinyos.net

[2] http://cvs.cens.ucla.edu/emstar

[3] http://sensorbase.org

[4] http://sensorbase.org/~kchang/emissary/das

[5] http://www.jamesreserve.edu

[6] http://www.botgard.ucla.edu/bg-home.htm

[7] http://xbow.com