MoteLab: A Wireless Sensor Network Testbed

Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh
Division of Engineering and Applied Sciences
Harvard University
{werner,swies,mdw}@eecs.harvard.edu

***Abstract*— As wireless sensor networks have emerged as a exciting new area of research in Computer Science, many of the logistical challenges facing those who wish to develop, deploy, and debug applications on realistic large-scale sensor networks have gone unmet. Manually reprogramming nodes, deploying them into the physical environment, and instrumenting them for data gathering is tedious and time-consuming.**

To address this need we have developed MoteLab, a Web-based sensor network testbed. MoteLab consists of a set of permanently-deployed sensor network nodes connected to a central server which handles reprogramming and data logging while providing a web interface for creating and scheduling jobs on the testbed. MoteLab accelerates application deployment by streamlining access to a large, fixed network of real sensor network devices; it accelerates debugging and development by automating data logging, allowing the performance of sensor network software to be evaluated offline. Additionally, by providing a web interface MoteLab allows both local and remote users access to the testbed, and its scheduling and quota system ensure fair sharing.

We have developed and deployed MoteLab at Harvard and found it invaluable for both research and teaching. The MoteLab source is freely available, easy to install, and already in use at several other research institutions. We expect that widespread use of MoteLab will accelerate and improve wireless sensor network research.

I. Introduction

Testing wireless sensor networks can be frustrating. Deploying a network into a realistic environment requires iteratively reprogramming dozens of nodes, locating them throughout an area large enough to produce an interesting radio topology, and instrumenting them to extract debugging and performance data. Although reasonable tools exist for evaluating large sensor networks in simulation [1], [2], only a real sensor network testbed can provide the realism exigent to understand resource limitations, communication loss, and energy constraints at scale. The advent of networked "backchannel" interfaces for sensor nodes, such as the Crossbow MIB-600, makes remote reprogramming and monitoring of permanently-powered sensor network nodes possible. However, there are few existing tools for managing such large-scale networked testbeds.

The MoteLab sensor network testbed platform addresses these challenges. Through a web interface, MoteLab allows users to create and schedule experiments. It automates testbed reprogramming and logs data generated by experiments to a persistent database. Users can retrieve data through the web interface or directly from the database. MoteLab also allows users to interact with individual nodes directly during experiments. By providing a web interface to the testbed MoteLab simplifies and accelerates deploying and evaluating wireless sensor network applications.

In situ power profiling of sensor applications is another difficult task faced by researchers, since rigging a node up to an appropriate measurement harness is often non-trivial and requires manual data capture [2]. We have instrumented a node in MoteLab with a network-connected digital multimeter, allowing the MoteLab backend to continuously monitor the energy usage of the node. Current consumption data is logged and returned with other data generated during the experiment. Gathering energy usage data simply requires checking a box in the Web interface.

MoteLab has been deployed on a network of 30 Ethernet-connected MicaZ "motes" distributed over three floors of Maxwell Dworkin, the Electrical Engineering and Computer Science building at Harvard University. MoteLab is freely available as open source, and several universities and research labs have chosen it to manage their own sensor network testbeds. In addition, because MoteLab exports functionality through a web interface, remote users can use the Harvard testbed for their own research. We believe that pervasive web-based access to such testbeds will soon become an essential tool for developing and evaluating sensor network applications, thereby supporting a broad range of research efforts in this field.

This paper describes the architecture and implementation of the MoteLab testbed framework and our experiences with the system over the last 16 months of operation. Section II discusses related testbed tools. Section III provides a technical overview of the MoteLab system, and in Section IV, we discuss the various usage models for application development and debugging. In Section V, we present several ongoing research projects that have benefitted from MoteLab. We close the paper with a discussion of areas for improvement and future development plans.

II. Related Work

While there are a number of existing sensor network testbeds, most of these have not addressed the user interface issues arising from a large and diverse user population. Existing testbeds often require users to use homebrew scripts to reprogram the network or debug network applications. Scheduling testbed access is performed manually, e.g. over a mailing list. We designed MoteLab because we beleive that a *web interface* is essential for simplifying access to the testbed. The web interface MoteLab provides facilitates testbed scheduling, node programming, data logging, and administrative functions such as adjusting user quotas.

Most similar to MoteLab in goals and realization is EmStar [3], a Linux-based approach to sensor network development and debugging. Emstar is a sensor network emulator allowing the boundary between simulated components and hardware to be shifted. For example, the same application can run using either a simulated radio model or the radios of a ceiling-mounted array of nodes. A related project, EmTOS [4], allows TinyOS applications to be recompiled to run on EmStar. MoteLab differs from these projects by running code directly on real hardware and facilitating fair sharing of the testbed among users.

SCALE [5] is a wireless communication assessment tool built on top of EmStar. In that way it is similar to MoteLab's Connectivity Daemon (Section IV-B), which collects statistics on radio connectivity between MoteLab nodes and presents this data to users through a web interface. MIT's Roofnet [6] has also been used to collect similar data.

More recently, several other large-scale testbeds have motivated groups to develop management software. Ohio State University is in the process of deploying a heterogenous testbed consisting of both

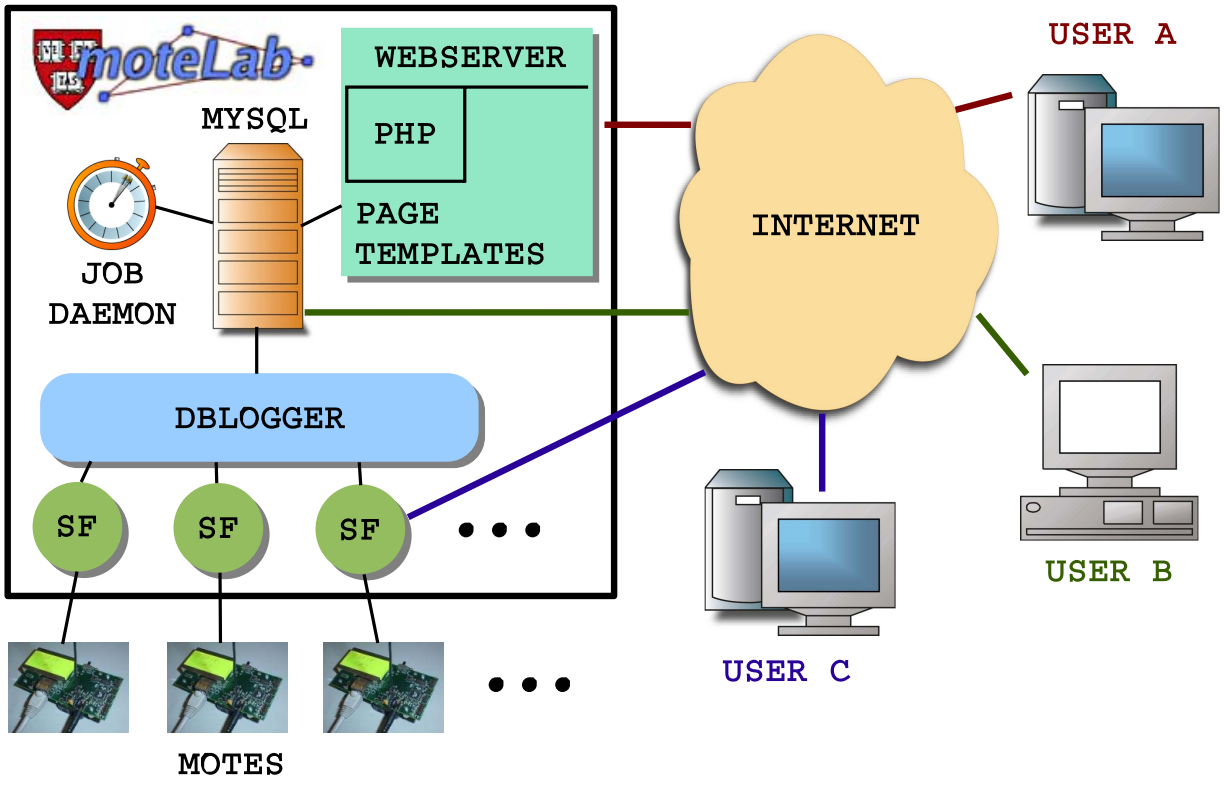

Fig. 1. **Component model showing the different software pieces that combine to form MoteLab. Connections between components illustrate data flow. Here, three external users are using the lab. User A is setting up a job to be run later (Section III-D). User B is accessing the MySQL tables directly to process data collected during a previous experiment (Section III-C). User C has a job running and has made a direct connection to a serial forwarder to receive or send messages to the attached node (Section III-G.2).**

XSM "motes" and more substantial Stargate nodes. Kansei [7] is under development to manage this testbed. It is unclear how similar it will become to MoteLab before the testbed opens in Spring, 2005, but due to the nature of the hardware supported there are some inherent differences. Data logging, for example, is initially done to local Stargate nodes rather than to a central server.

To address high levels of testbed contention Intel Research Berkeley has developed Mirage [8] which applies microeconomic approaches to arbitrate among competing users. Users request resources by submitting bids using a virtual currency and auctions run periodically to select the winners. Besides this different scheduling model, Mirage also does not provide the automated reprogramming and data collection that MoteLab does. Users access the subset of the testbed that they have gained access to by logging into a server and are responsible for directing any reprogramming or data collection that their experiments require. We view this usage model as extremely complementary to the one provided by MoteLab and plan to integrate more direct control over lab resources into future MoteLab releases.

III. Technical Details

MoteLab is a set of software tools for managing a testbed of ethernet-connected sensor network nodes. A central server handles scheduling, reprogamming nodes, logging data, and providing a web interface to users. Users access the testbed using a web browser to set up or schedule jobs and download data.

MoteLab consists of several different software components. The main pieces are:

- **MySQL Database Backend** : Stores data collected during experiments, information used to generate web content, and state driving testbed operation.
- **Web Interface** : PHP-generated pages present a user interface for job creation, scheduling, and data collection, as well as an administrative interface to certain testbed control functionality.
- **DBLogger** : Java data logger to collect and parse data generated by jobs running on the lab.
- **Job Daemon** : Perl script run as a cron job to setup and tear down jobs.

Figure 1 shows elements of the communication between these components and an illustration of typical testbed activity.

The rest of this section is structured as follows. We begin with an overview of MoteLab hardware. After defining a MoteLab "job", an important piece of terminology, we explain each software component listed above in more detail. Finally, we describe several useful MoteLab features not mentioned above. We postpone a description of the typical MoteLab user experience to Section IV.

A. MoteLab Hardware

MoteLab software manages an fixed array of wireless sensor network nodes fitted with Ethernet interface backchannel boards allowing remote reprogramming and data logging. At Harvard, our original testbed was 26 Mica2 motes. The Mica2 "mote" consists of a 7.3 MHz ATmega128L processor, 128KB of code memory, 4KB of data memory, and a Chipcon CC1000 radio operating at 433 MHz with a data rate of approximately 34 Kbps. These were attached to 26

Fig. 2. **Crossbow MIB600 "emote" interface board. Attached to a Mica2 or MicaZ "mote", provides remote reprogramming and data logging capabilities.**

Ethernet interface boards: 6 EPRB's [9] developed at Intel research by Phil Buonadonna and 20 Crossbow MIB-600 "emotes" (Figure 2). The EPRB was the original solution integrating the reprogramming and data logging hardware. The MIB-600 is a later, single-board solution. Both provide one TCP port for reprogramming and another for data logging. We recently upgraded MoteLab to use 30 CrossBow MicaZ motes, which upgrade the Mica2 with Chipcon CC2420 IEEE 802.15.4 compliant radios.

While we expect that MoteLab will be most useful for experimenting with nodes in the Mica lineage running TinyOS [10], the MoteLab software can manage any lab of nodes providing remote reprogramming and data logging capabilities. For example, we intend to explore the use of Ethernet-USB interfaces for interfacing to the newer Telos [11] motes.

At Harvard, all of the MoteLab software described below runs on a central server running Linux with Apache, MySQL, and PHP.

B. What Is a MoteLab Job?

Throughout this section we refer to a "job" running on MoteLab. A MoteLab job consists of some number of executables and testbed nodes, a description mapping each node used to an executable, several Java class files used for data logging, and other configuration parameters, such as whether or not to perform power profiling during the experiment. To create a job a user uploads the required executables and class files and describes how the lab should be programmed. Once a job is created, MoteLab stores the configuration information allowing the same job to be run multiple times, for different amounts of time or at different times of day.

C. MySQL Database Backend

MoteLab uses a MySQL database to store all information needed for testbed operation. This information divides into two categories: job-generated data and testbed state.

When a user account is created, a MySQL database is created for that user that will hold all of their job-generated data. A new set of tables is created for each instance of a job run, one table for each message type associated with the job. The user is given access rights to their database, allowing them to leverage the MySQL query language for post-processing.

A seperate database holds all lab state information, including user information and access rights; node state; information about uploaded executables and class files; job properties; and a representation of the lab schedule. This state is provided to and modified by all of the other main MoteLab components.

D. Web Interface

MoteLab uses PHP to generate dynamic web content, and Javascript to provide an interactive user experience. This allows users to access the lab in a platform-independent way. After logging in, a normal user has access to the following functionality:

- **Home Page** : provides a summary of pending, running, and completed jobs, and the ability to download data logged by the lab during past experiments.
- **User Info** : instructions for database access, serial forwarder access, and the ability to change lab passwords.
- **Create Job** : a rich interface for composing jobs. Users can upload executables; choose which executables will run on which testbed nodes; upload class files for message parsing; and choose from among various options, including whether to run power profiling during the experiment. Administrators can also choose to run a job as a daemon for a given period at specified intervals, and choose programs to run on MoteLab during and after job execution.
- **Edit Job** : provides the same capabilities as the job creation page, but reloads it with information from a stored job.
- **Schedule** : presents a view of the current state of the lab, including finished, running, and pending jobs, and the ability to schedule a job at various degrees of granularity. Users can delete their own pending jobs; administrators can delete any.

Two additional pages are provided for administrators. The first allows new user accounts to be created and modified, the second allows lab partitioning to be configured.

E. DBLogger

DBLogger is a Java program started at the beginning of every job. It connects to each node (via the each node's interface board's data logging TCP port described in Section III-A) and uses class introspection to parse messages sent over its serial port and insert them into the appropriate MySQL database. The individual fields of each message sent are parsed and their values extracted into the database. The resulting table structure is identical to the message structure, with the addition of fields identifying which testbed node originated the message, the time the message was inserted into the database, and a global sequence number.

F. Job Daemon

The Job Daemon is a Perl script run as as cron job. The Job Daemon responsible sets up experiments, which involves reprogramming nodes and starting other necessary system components (including the DBLogger and serial forwarders), and tears them down when finished, which involves stopping node activity, killing processes necessary during the job, and dumping the data from the MySQL database into a format suitable for download.

G. User Quotas, Real-time Access, Power Measurement

Three additional features of MoteLab not mentioned above merit attention.

1) User Quotas: We provide a user quota system that facilitates sharing the lab between multiple users. The quota does not control how much total access the user can have to the lab. Rather, it limits the number of outstanding jobs that the user can post to the lab at once. For example, with a 30 minute quota, a single user could still use many hours of lab time during a given day. But they could only have 30 minutes of pending jobs at once, limiting the rate at which they can schedule jobs and allowing other users to better compete for access to the lab.

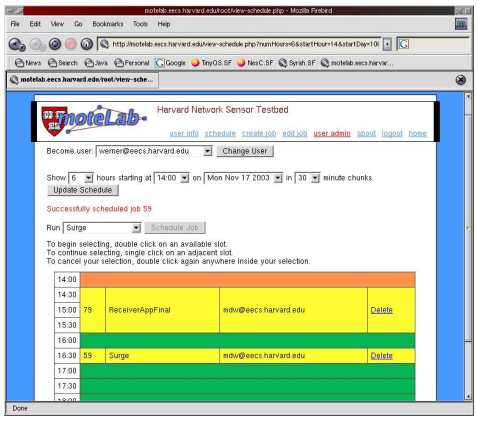

Fig. 3. **MoteLab screenshot showing the Schedule Page. Two pending jobs are shown.**

2) Direct Node Access: In addition to logging data to a database through DBLogger, we provide users with direct access to each node's serial port over a TCP/IP connection. This permits the use of custom programs for monitoring and injecting data into the running application. Because our interface boards allow a single TCP connection to the node, we use the TinyOS *SerialForwarder* program, which acts as a TCP multiplexer.

3) In Situ *Power Measurement:* We have connected one node on our network to a networked Keithley Digital Multimeter (Keithley 2701) and we expose the use of this device on the Create Job page. The Keithley can sample continuously at 250Hz, and it bursts at 3000Hz; currently we only suppor continuous operation. Time-stamped current data is included in the download archive if the user has selected this option. As sensor network developers become more aware of power as a design constraint we expect that use of this feature will become more common.

IV. Use Models

Broadly speaking, there are two different ways to use MoteLab. Users can schedule a large number of jobs to be run unattended in a batch fashion, or they can interact directly with their running job by attaching to the exposed per-node serial forwarders or exploiting real-time access to the MySQL database. Below we briefly describe both usage models.

A. Batch Use

Sensor network experimentation begins on the desktop. After local testing to verify that their application produces data, a user is ready to use MoteLab. After logging on they proceed to the Create Job page. After uploading the necessary files and specifying other job parameters they proceed to the Schedule page and schedule their job some time in the future.

When the job is ready to run, the Job Daemon reprograms the network and starts the DBLogger with the user-uploaded class files. The job is now live, and data sent to the serial port of any node will be parsed and inserted into the appropriate MySQL tables created for this job. When the job completes, the Job Daemon removes the executable from the lab, and archives job data.

After the job completes post-processing can be done by parsing the data dump files in the job download or by directly accessing the MySQL database.

B. Real-time Access

MoteLab allows researchers to connect directly to the serial forwarder running during their job via a set of dedicated ports on the MoteLab machine. This facilitates a wide variety of different ways of interacting with a running job. For example, a researcher may have a data set that they want to inject for simulation, either because the data collected is not of the type that could be collected on MoteLab, or to make experiments reproducable.

Another use of the direct serial forwarder access is to do real-time data analysis. Real-time data processing is possible either by connecting to the serial forwarders providing a data stream for each node, or by accessing the MySQL database during the job.

As an example of an application that uses real-time access, as well as almost every other feature available on MoteLab, we discuss here the Connectivity Daemon. The Connectivity Daemon is job like any other on MoteLab. At Harvard it is used to collect information eventually used to graphically illustrate connectivity between lab nodes on the Maps page.

The Connectivity Daemon uses one executable and two class files. In addition, it uses three features available only to administrators: the abilities to run periodically when the lab is available, to execute a program locally on MoteLab during the job, and to run a program on MoteLab when the job completes. A Java program that runs during the job connects directly to active nodes and uses this connection to tell them when to send messages and when to listen. After the job completes, a Perl script accesses the database tables created during the job and calculates packet loss rates between each pair of nodes in the network. Before finishing, it updates the connectivity information stored in the mote info table in the database. The next time that the Maps page is viewed, PHP generates a graphical representation of link quality between nodes by using the new information. Figure 4 shows a screenshot of the Maps Page illustrating how the data collected by the Connectivity Daemon is used.

V. Research Using MoteLab

Although a relatively new tool, MoteLab is already at work facilitating sensor network research at Harvard and elsewhere. Here we describe several research projects at Harvard that have used MoteLab, as well as instructional lab use. Because MoteLab provides a web interface, we at Harvard have a number of external users and we briefly describe their activities. And with MoteLab now deployed at several other universities we attempt to provide information about their experiences with the MoteLab software.

A. MoteTrack and CodeBlue

MoteTrack is an RF-based location tracking system developed for TinyOS-based motes [12]. While RF-based location tracking is a well-studied problem, most existing approaches make use of 802.11 and a *central server* to map RF signals to physical location. In contrast, MoteTrack is entirely *decentralized* and is designed to be extremely robust to failures of the beacon node infrastructure. In our building, MoteTrack achieves an 80th-percentile tracking accuracy of 2 meters, which rivals that of existing systems based on 802.11. In addition, MoteTrack continues to operate well in the face of beacon node failure or signal distortion.

MoteTrack represents a case where MoteLab is used not just to develop a complete sensor network application in an arbitrary environment, but as a valuable infrastructure in its own right. The distribution of MoteLab nodes around our building allowed us to achieve good coverage and develop a building-wide location tracking system, which would not have been possible in a single, smaller lab.

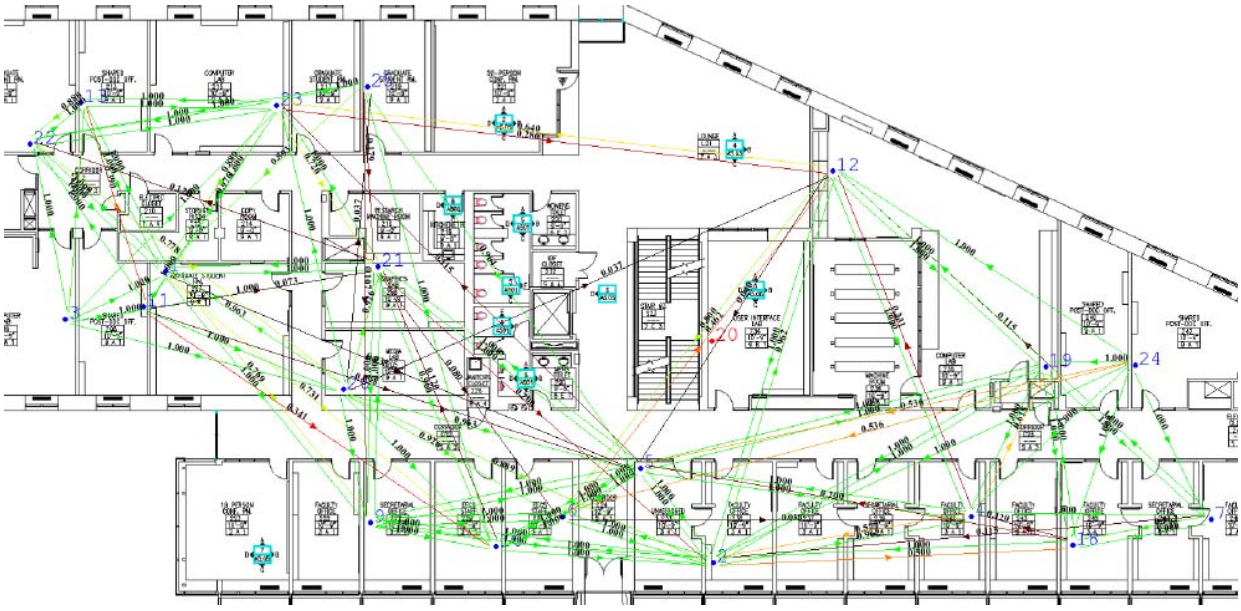

Fig. 4. **Screenshot of the Maps Page showing connectivity on the most densely-populated floor of our MoteLab deployment. The Maps Page is generated dynamically by PHP using data collected periodically by the Connectivity Daemon (Section IV-B).**

The Harvard CodeBlue project [13] is developing robust protocols and services for integrating wireless devices into a range of medical care settings. This project requires that we develop a robust infrastructure for naming, discovery, data delivery, and security that runs on a range of devices, including motes, PDAs, and laptops. We are experimenting with *ad hoc* multicast routing algorithms, such as ADMR [14], which permit multiple devices to transmit data to multiple recipients over multihop wireless links. MoteLab is an essential resource for developing the CodeBlue system, allowing us to test our approach in a realistic setting on real motes.

B. Instructional Use

MoteLab is also a valuable tool for teaching sensor network concepts, allowing students to experiment with a real testbed. The Web-based interface simplifies the mechanical aspects of programming and debugging the network. MoteLab has been used to teach two graduate courses at Harvard involving sensor network projects. In each class (32 and 28 students, respectively), groups of 2-3 students were issued "mote kits" comprised of several Mica2 or Telos motes, programming boards, and cables. Students used the kits to develop code for a TinyOS-based programming assignment, then deployed their code on MoteLab for a full-scale analysis. In the first year, students used MoteLab to study radio connectivity and RSSI as it varies with distance. In the second year, students developed a robust multihop routing protocol. Students also undertake an independent research project related to the theme of the course, and many groups used MoteLab for this purpose. The MoteTrack system described in Section V-A was originally a project for this course.

C. External Users and External MoteLabs

We have made MoteLab accounts available to external researchers upon request. To date, 20 external users have MoteLab accounts. One external user used it to analyze signal strength observed at different nodes and create a location estimation technique based on cluster analysis. Another used MoteLab to evaluate an order statistic service. MoteLab's web interface and quota system facilitate easy lab sharing. We view our MoteLab as a shared resource for the entire sensor network research community.

The MoteLab software is available for download and several external research groups have installed the system for managing their own testbeds. External use by these groups has resulted in valuable feedback and led to feature additions. Examples include the direct node access discussed in Section III-G.2, division of the lab into multiple "zones" scheduled independently, and simple scripting support.

D. Discussion

Our experiences with MoteLab, the experience of the user community, and recent related work (e.g. Mirage) have made evident the salient features of the MoteLab model and brought into clearer focus the strengths and weaknesses of our design.

For testbeds with a high degree of contention our simplistic scheduling model may not be the best approach. At Harvard, our MoteLab is typically lightly loaded. When Intel Research, Berkeley, experimented with MoteLab they found the scheduling model unable to cope with their level of testbed contention. Out of that need came Mirage [8] which uses an economic, bidding-based model to schedule testbed resources.

Another MoteLab design decision that Mirage brings into focus is tying testbed access with job scheduling and data collection. MoteLab automates everything, when a researcher might actually want more fine grained control during their assigned time slot. Mirage solves this problem by allowing users to log into a central server and use tools their to manage the testbed during their time window, restricting their access to the nodes that they have access to and kicking them off the server when their time window expires.

We are also exploring ways to allow researchers direct access to nodes, both through a web interface available only during their experiment, and through command-line scripts that would integrate seamlessly into a build environment. Our eventual goal is to provide a "`make motelab`" command that would collect new binaries locally, consult a local configuration file, and then communicate with

the MoteLab server to upload binaries and properly reprogram testbed nodes (assuming the user has access to those testbed nodes at that time).

In January, 2005, a Testbeds Working Group was created to explore testbed-related issues and to continue the development of testbeds to assist in wireless sensor networks research. The group consists of representatives from Harvard, UC Berkeley, Intel Research Berkeley, MIT, Ohio State, Kent State, Harvard as well as others with large testbeds to manage. It is our hope that this working group will be able to integrate some of the existing solutions into a more global architecture, allowing testbed management code to be collectively developed and maintained.

VI. Future Work and Conclusions

Wireless sensor networks are becoming a nexus of activity within the computer science community. Unfortunately, few good tools for experimenting with wireless sensor networks exist. Simulators fail to capture significant details of node operation or wireless communication, and while real testbeds capture the realism that simulations miss, traditionally they have been difficult to use or limited to a small group of reseachers.

Our goal in designing MoteLab was to develop a interface to a networked testbed allowing a large community of users to share access. We believe that web-based access is essential since UNIX accounts or scripts are hard to manage and can restrict access to a local community. In addition, to arbitrate access among a large group of researchers a preemptive scheduler is necessary.

MoteLab meets these design goals. By running sensor network software on real hardware, MoteLab provides an experimentation environment similar to most deployments. MoteLab's web interface and preemptive scheduler allow a large community to share access to the lab and eliminates the difficulties inherent in cooperative scheduling. Finally, because MoteLab is freely distributed and built on top of readily-available software tools, other organizations can easily set up their own MoteLab testbeds. As our MoteLab testbed at Harvard increases to several hundred nodes, its size will drive future MoteLab development. Such a large lab will demand partitioning, allowing different experiments to run side by side on different portions of the lab. We have already developed a MoteLab feature allowing different lab "zones" to be created and scheduled seperately on the scheduling page, and we will continue work allowing MoteLab to manage larger labs. Other areas of future work include decoupling scheduling and job execution, allowing users more control over the testbed during their allocated time slots. Increasing the modularity of the codebase is also crucial since it allows a larger variety of hardware architectures to be supported. Finally, scripting support has been requested by several adopters and, while not replacing the web interface, would provide a powerful tool for more serious researchers interested in running multiple experiments with different parameters, for example.

In summary, we have described MoteLab, a wireless sensor network testbed. MoteLab is unique in its ability to manage a network of real, network-attached wireless sensor network nodes. MoteLab is a powerful tool for sensor network research, and we hope to see it widely deployed. We will continue MoteLab development as directed by the needs of our user community.

References

[1] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[2] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proc. the 2nd international conference on Embedded networked sensor systems (SenSys 2004)*, November 2004.

[3] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "Emstar: a software environment for developing and deploying wireless sensor networks," in *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, 2004.

[4] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, "A system for simulation, emulation, and deployment of heterogeneous sensor networks," in *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004.

[5] A. Cerpa, N. Busek, and D. Estrin, "Scale: A tool for Simple Connectivity Assessment in Lossy Environments," Center for Embedded Networked Sensing, University of California, Los Angeles, Tech. Rep. CENS Technical Report 0021, September 2003.

[6] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 121–132, 2004.

[7] Ohio State University, "Kansei: Sensor Testbed for At-Scale Experiments," Poster, 2nd International TinyOS Technology Exchange, Berkeley, CA, 11 Feb 2005.

[8] Intel Research Berkeley, "Mirage: Microeconomic Resource Allocation for SensorNet Testbeds," http://mirage.berkeley.intel-research.net/.

[9] P. Buonodanna, "EPRB : Ethernet PRogramming Board," http://berkeley.intel-research.net/pbuonado/EPRB/.

[10] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, USA, Nov. 2000, pp. 93–104.

[11] Moteiv Corporation, "Telos Sensor Network Module," http://www.moteiv.com.

[12] K. Lorincz and M. Welsh, "MoteTrack: A robust, decentralized location tracking system for disaster response," Harvard University, Tech. Rep. TR-19-04, March 2004.

[13] K. Lorincz, D. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh, "Sensor Networks for Emergency Response: Challenges and Opportunities," *IEEE Pervasive Computing*, Oct-Dec 2004.

[14] J. G. Jetcheva and D. B. Johnson, "Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks," in *Proc. ACM MobiHoc'01*, October 2001.