# Sensor Andrew: Large-Scale Campus-Wide Sensing and Actuation

Anthony Rowe† Mario Berges‡ Gaurav Bhatia† Ethan Goldman‡
Ragunathan (Raj) Rajkumar† Lucio Soibelman‡ James Garrett‡ José M. F. Moura†

† Electrical and Computer Engineering Department
‡ Civil and Environmental Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213

{agr,mberges,gnb,ejgoldman,rajkumar,lucio,garrett,jm5u}@andrew.cmu.edu

***Abstract*—We present Sensor Andrew, a multi-disciplinary campus-wide scalable sensor network that is designed to host a wide range of sensor, actuator and low-power applications. The goals of Sensor Andrew are to support ubiquitous large-scale monitoring, operation and control of infrastructure in a way that is extensible, easy to use, and secure while maintaining privacy. Target applications currently being developed as part of Sensor Andrew include builing emergency, first-responder support, quality of life for the disabled, monitoring and optimization of water distribution systems, building power monitoring and control, social networking, and biometric sensors for campus security. Sensing devices that are used range from cameras and battery-operated sensor nodes to energy-monitoring devices wired into building power supplies. Some of these sensing devices may also be mobile and require hand-off between different networked regions. Supporting multiple applications and heterogeneous devices requires a standardized communication medium capable of scaling to tens of thousands of sources.**

**In this technical report, we present the architecture underlying Sensor Andrew for managing sensor data collection as well as server-side application interactions. Sensors and actuators are modeled as event nodes in a push-based publish-subscribe architecture. A data handler provides registration, discovery and data logging facilities for each device. The major elements of this architecture have been deployed in five buildings at Carnegie Mellon University, and are comprised of over 1000 sensing points reporting data from multiple communication interfaces. Finally, we describe two different case study applications currently using the infrastructure that benefit from shared information. Design choices, limitations and enhancements across various layers and protocols are also discussed.**

## I. Introduction

Sensor Andrew is a large-scale effort to widely deploy sensing devices across Carnegie Mellon University. We envision a broad set of applications ranging from: infrastructure monitoring, first-responder support, quality of life for the disabled, water distribution monitoring, building power monitoring and control, social networking and biometric systems for campus security. Researchers of other institutions have already successfully built other sensor networking applications, but they are typically isolated, small-scale and short-lived experiments. One of the primary goals of Sensor Andrew is to have a permanent living laboratory where applications can be rapidly prototyped at scale. Our architecture focuses on supporting practical deployments with direct community uses. Imagine an infrastructure monitoring system that could immediately alert the campus facilities personnel in the event of broken water pipes or power outages. We see great potential in development not only at the sensor networking level, but also with applications operating at a higher level of abstraction. Application developers should be able to directly utilize physical data from the environment without having to re-invent lower-level interfaces. A variety of social networking applications are possible given support for mobility. Students can carry low-powered mobile devices that communicate seamlessly with the the infrastructure and one another.

The target applications of Sensor Andrew will require numerous heterogeneous sensors, actuators and communication networks in order to interoperate. During the onset of this project we found that multiple groups across campus were either deploying redundant sensing systems or were unaware of existing systems that could benefit their applications. By combining resources from different applications, the system can become greater than the sum of its parts. In order to support this mixing of components, there needs to be a unifying architecture capable of facilitating communication between components while maintaining security and privacy.

In this paper we introduce the goals and requirements of an architecture that meets the needs of such a large-scale sensing and actuation system. We propose a three-tiered architecture that utilizes a distributed communication and addressing service capable of scaling to Internet class proportions. The communication component of the architecture provides a standard messaging interface with extensible packet types incorporating encryption and user access control. We propose an extensible schema for both communication as well as a database for historical information. We leverage the XMPP Internet messaging protocol to provide both point-to-point as well as publish-subscribe communication.

We evaluate the flexibility of this architecture by showcasing two early applications. The first application attempts to reduce building energy consumption through non-intrusive electrical load monitoring. Its goal is to understand if better decisions can be made towards saving energy given the operational schedule of major appliances. The second applications is a campus-wide wireless sensor network used for environmental monitoring and tracking mobile tags. Tags can be used to track targets ranging from technical equipment like portable projectors or to students themselves participating in a social networking project. In both cases, we show how easy access to information streamlines application development, and how multiple existing sensor systems can be used in collaboration.

### A. Related Work

Multiple complete sensing applications have been deployed showing the technical feasibility of sensor networks [1][2]. These applications are successful at their specific tasks, but tend to be relatively small-scale and somewhat narrowly defined systems, usually with environmental concerns or with a military purpose. Also, many of these deployments tend to be short-lived, and although they show great potential as components for other applications, the technology is often not reused. Each application in Sensor Andrew is designed from the start with components that promote integration. In order for this to be successful, it must be easy to interface new components or subsystem with the existing infrastructure while at the same time provide a benefit to the application.

In [3], the authors outline that much of the work done in sensor networking has resulted in vertically built designs where individual components are only compatible with each other and not with other systems. The authors put forth the challenge of making a protocol for sensor networks that can unify communication in the same way that IP was able to support the Internet. Due to the resource-constrained and hence the tightly coupled and highly optimized nature of sensor networks, this problem continues to exist. Our objective in this work is not to standardize low-level sensor networking communication, but instead to enable unified tools and interoperability across multiple deployments. Work has also been done trying to integrate sensor networks with IPv6 [4][5]. IPv6 solves many of the problem of addressing and sending / receiving data packets to and from individual sensor devices. It allows standard tools such as *traceroute* and *ping* to be used for network diagnostics. However, it does not solve many of the higher-level challenges associated with managing and providing applications with the data. In this paper we provide a framework that would run on top of existing network protocols like IPv6 and IPv4 that addresses access control, registration, discovery, event logging and management of transducer devices beyond a single subnet.

Multiple research groups have worked on collaborative sensing services like SenseWeb [6] from Microsoft research and SensorWeb [7]. These systems are targeted towards visualizing and sharing data with end-users. They currently show little support for managing actuators. Sensor Andrew aims not only to collect sensor data, but to support control of environments and to enable interaction between software agents. These projects complement Sensor Andrew and could be integrated to aid in navigation and visualizing events.

There have been several research efforts in the field of pervasive computing and social networking [8][9][10]. These projects could greatly benefit from a low-power ubiquitous sensing and communication infrastructure. Solutions like TinyDB [11] exist that allow for querying and collection of data. MoteTrack [2] demonstrates a system capable of indoor tracking. Unfortunately, the operation of these systems typically requires an application domain expert and ends up making it nearly impossible for further system integration.

The building industry has been working for a number of years towards standard communication protocols and data formats to simplify the exchange of information between monitoring and control equipment in commercial and residential buildings [12][13]. Multiple attempts [14][15] have been made to translate communications between different systems in order to facilitate cross-domain interactions. These systems are narrowly focus on supporting control and automation interactions and were never intended to operate with the number of users or scale of Sensor Andrew.

The IRISnet [16] project shares many of the same goals as Sensor Andrew by enabling transducer reuse and collaborative Internet-scale sensing. It uses a distributed database-centric architecture that facilitates the storage, processing and retrieval of transducer information. The IRISnet architecture was intended primarily for Internet connected desktop PCs and inexpensive commodity off-the-shelf sensors such as Webcams. In contrast, Sensor Andrew focuses on management of a wide range of devices including resource-constrained transducers which may not have direct Internet connectivity. It also provides presence notification essential for supporting mobile devices.

### B. Paper Organization

The rest of this paper is organized as follows. Section II describes the architecture requirements, design goals and trade-offs associated with Sensor Andrew. Section III describes the architecture we chose for Sensor Andrew and where it is located in the design trade-off space. Section IV discusses the implementation of various system components. Section V presents early experiences with Sensor Andrew illustrated through two different case studies. Finally, Section VI summarizes our contributions and discusses our future plans.

## II. DESIGN GOALS AND TRADEOFFS

We adopt the following design goals for Sensor Andrew.

1) **Ubiquitous Large-Scale Monitoring and Control:** The sensing infrastructure should exist at a significantly large scale to entice the development of new and innovative applications. The infrastructure should support both sensing and actuation.
2) **Ease of Management, Configuration and Use:** The system's ease of use needs to be considered both for managing the infrastructure as well as providing simple

ways for application developers to interface with their own and other subsystems. This should also include a process for registering and discovering transducers relevant to the user's project.
3) **Scalability and Extensibility:** The ability to support a large number of devices and users is of paramount importance. Extensions made by a particular project to satisfy a new requirement should ideally benefit the whole community.
4) **Built-In Security and Privacy:** The system should support security and privacy considerations including encryption, key management, access control and account / user management. Social aspects of privacy should be governed through policy.
5) **True Infrastructure Sharing:** One of the most unique contributions of Sensor Andrew is the notion of multiple heterogeneous applications and devices that can utilize each other's services. The architecture must easily integrate information from multiple applications, creating additional value to all as new types of applications are envisioned.
6) **Evolvability:** The architecture must be capable of evaluating different computation paradigms. It must also allow for rapid prototyping at scale to demonstrate practical usage and utility. It also needs to be able to change over time. Being able to evolve with and support these changes will be important for incorporating unforeseen and innovative applications in the future.
7) **Robustness:** The system should be robust and be able to reconfigure itself.

While each of the design goals may be worthy by itself, collectively they may conflict with one another and many design trade offs must therefore be considered. For example, the system could be push-oriented or pull-oriented when it comes to accessing sensors. Devices could broadcast data whenever they are ready, but this could waste energy if no agent is available to consume the data. If devices only respond to polling requests this would no longer be a problem except when many devices request the same data. Queues of requests could build-up for redundant data that would have been better handled by the push-based broadcast approach. We also need to address how these architectures will support actuators. A pull approach might simply send the actuator a message to enable it. For requirements like this, there needs to be mechanisms in place for resource management. In these examples, each scheme has an inherent bias and will likely better suit a certain set of applications.

In a system like Sensor Andrew, there will be different kinds of users: (a) end-users who only want to avail themselves of the features supported by various applications in the system (b) application developers who want to test and deploy applications quickly, and (c) system developers who may want to change the underlying protocols and node system software. We strive to create a secure backbone that takes care of authentication, encryption and access control. Thus, we have at least strived to provide a good base for applications to build upon. Scalability and extensibility need to be addressed in a way that balances system modularity with size and complexity. Self-healing mechanisms should aid in robustness, but without an unacceptable loss of performance. Since many aspects of Sensor Andrew are still open research topics, we wanted to design an architecture that would aid in fusing pieces together yet require as little intervention as possible.

### A. Challenges and Approach

Given these lofty goals, it would be infeasible to design every component of Sensor Andrew from scratch. We utilize existing technologies whenever possible and innovate whenever necessary. There are five core challenges required to meet our goals: (1) uniform access to heterogeneous devices; (2) sharing of transducers across applications; (3) scaling to many devices; (4) integration of subsystems and (5) security and privacy. Uniform access to devices is achieved using self-describing data objects. In our implementation, this takes the form of a transducer schema. Sharing transducer information across applications is achieved through a publish-subscribe mechanism. Scalability is achieved through use of encapsulated addressing. Each devices in the system is addressed with a unique name, server address and namespace attribute. Integration of subsystems is possible because of standardized communication mechanism with adapters providing the last-link of translation. Finally, security and privacy are achieved through encryption, key management, access control and policy. Our overall contributions are defining an architecture for Sensor Andrew, selecting the best currently available technologies to meet our requirements, and building the required tools to make the system cohesive.

## III. Sensor Andrew Architecture

In this section, we present the first version of our architecture designed to satisfy the goals of Sensor Andrew. Figure 1 illustrates the classical three-tiered architecture we adopted with a front-end server layer, a gateway layer and a transducer layer. The servers and gateways operate as part of the campus network, while the transducer layer may communicate over a variety of different bus or network protocols. Elements of the transducer layer are end-point sensors or actuator devices with little or no processing power. For example, a light sensor or energy meter that provides a voltage output or serial data would be considered part of the transducer layer. The gateway layer is comprised of medium to desktop-class computing devices that have Internet access. As described later in this section, gateways are responsible for running *adapters* that properly format the transducer layer information for the server layer. At its core, the server layer has a set of high-performance systems with extensive storage capabilities. This layer acts as the administrative front-end to the entire system for configuration, control, data aggregation and storage. The server layer also consists of *agents* which can subscribe to sensor data to provide high-level services or new *meta-sensors*.

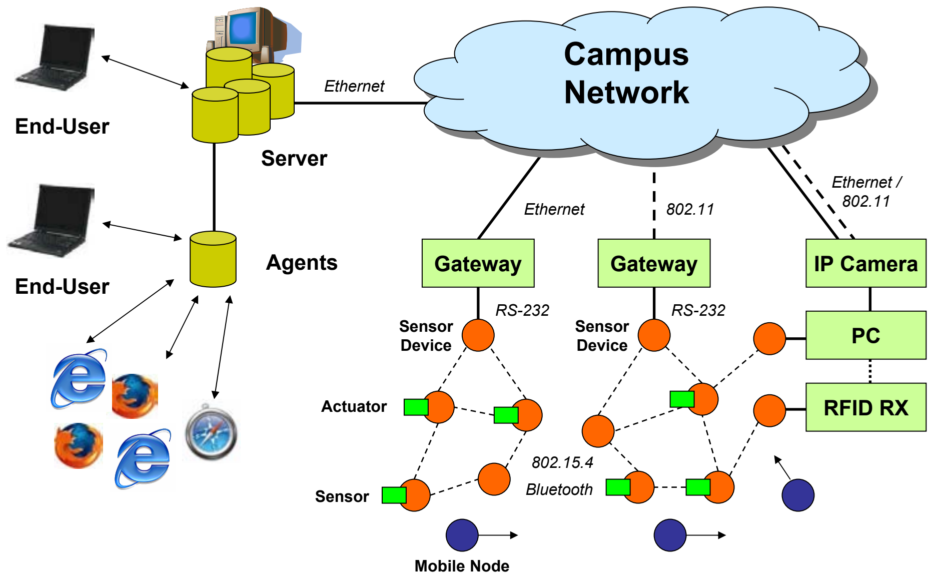

Fig. 1. The three-tiered Sensor Andrew Architecture.

## A. Communication Requirements

We now outline the requirements of a communication protocol that could achieve the goals outlined in Section II.

1) **Standard messaging format:** Applications frequently contain roll-your-own solutions for communicating between components. These solutions are often difficult to interface with one another either due to architectural incompatibility or even simple lack of documentation. Even if the body of a message is unique to each application, a standard messaging protocol will simplify data payload delivery.
2) **Extensible Message Types:** Different applications can be expected to require dramatically different message types. These could range from simple differences like one type of sensor over another or more significant changes such as streaming versus packetized data.
3) **Point-to-Point and Multicast Messaging:** The communication protocol should be able to provide point-to-point as well as, broadcast support in order to interface with multiple applications. Multicasts should ideally be implicit and not require any changes to the original application if more or fewer listeners exist in the system.
4) **Support for Data Tracking and/or Event Logging:** For maintenance purposes, the system must have the ability to track where data are being moved and the volume of data that different applications are generating. This can be later used for reconfiguration, fault analysis and/or to optimize system parameters. There also needs to be a mechanism for storing information about transducers, such as part number, units, location, etc.
5) **Security, Privacy, Access Control:** Providing security and access control at an applications communication interface is essential for protecting sensitive information. The communication layer should allow for access control and the ability to share privileges without always having to go back to a single administrator.
6) **Internet-Scale Performance:** The communication protocol needs to be able to support a large number of devices ideally without unduly impacting applications.

## B. Communication

In order to provide communication between gateways and user applications we chose to leverage the eXtensible Messaging and Presence Protocol (XMPP)[17]. XMPP is an open XML-inspired Internet protocol traditionally used for online chat communications. Originally based on the Jabber protocol, XMPP has evolved to incorporate features well beyond simple instant messaging such as: event publishing, voice streaming, file transfer and profile information management. The notion of presence, which is central to its operation, refers to the ability for groups of clients to detect other clients connecting and disconnecting from the system. This is critical both to identify when a service becomes available and to direct client-to-client communication given potentially new locations in the network. Figure 3 shows how XMPP uses decentralized addressing making it highly scalable. Much in the same way a domain can run its own email server, addressing in XMPP is defined first with a client identification (referred to as a JID) followed by a domain name and then a namespace. Entities using XMPP are classified as clients and servers. For example `gw_x@sensor.andrew.cmu.edu/water-pipes` identifies a particular gateway node's address, `gw_x` , at an XMPP server address, `sensor.andrew.cmu.edu` with its "namespace" specified as `water-pipes`. An XMPP server with the correct access permissions can pass a local client's requests to another XMPP server which, in turn, can pass the request to the

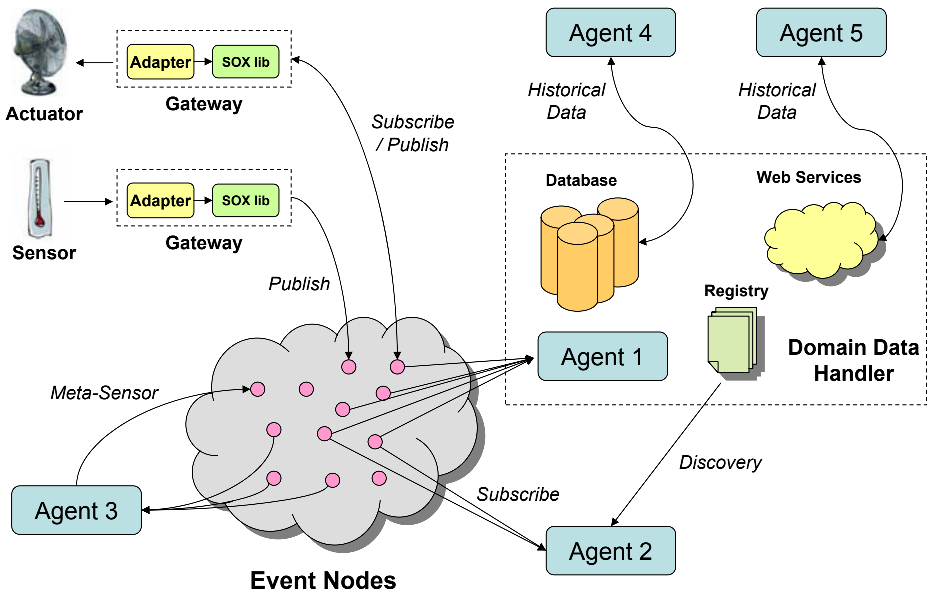

Fig. 2. XMPP Publish-Subscribe transactions to support collection of sensor networking data.

destination client. The addition of namespaces appended to the addresses allows for the creation of multiple views.

XMPP supports publish-subscribe messaging where JIDs can send and receive messages through what are called *event nodes*. Event nodes are addressable data channels that allow clients to publish and/or subscribe to event feeds. Nodes may also maintain a history of events, provide meta-information about the event feed as well as contain access control lists. This push model of communication provides a powerful mechanism for distributing sensor data to any interested application or user.

XMPP satisfies our initial requirements in the following ways:

1) XMPP provides a standard, scalable messaging and presence protocol with security features such as user/group authorization, authentication, and access control. Since XMPP is already an Internet standard, we can leverage commercially available servers that are maintained by the community.
2) XMPP's addressing and messaging scheme is optimized for short messages with point-to-point as well as broadcast capabilities. The addressing scheme is not bound to a physical network location making it ideal for mobile devices.
3) XMPP provides a publish-subscribe functionality for pushing sensor data. This is an ideal model for mass distribution of data.
4) XMPP provides organized event messages with an internal database for storing transaction records.
5) XMPP can utilize clustering or replication to meet scale demands as well as provide primary backup fault-tolerance.

XMPP has its limitations in our context. Some examples of these limitations are:the need for data schemas to structure transducer data, adapters to interface hardware devices with XMPP, data services to support applications and software agents that can be easily developed and deployed to analyze and respond to the messages.However with the addition of our tools and supporting agents, which will be addressed in Section IV, it enables large-scale sensing and addresses all these issues.

## C. Transducer Layer

The transducer layer consists of end-devices that typically have the ability to measure or change some physical characteristic of the environment. This might be as simple as a temperature sensor, or could include a sensor like a cellular phone that is detected using Bluetooth while moving past a gateway. The collection nodes in a classical wireless sensor network would be considered part of the sensor layer. In cases like that of an energy measurement device connected to a linux host, the energy device would be considered a transducer, while the linux host would be a gateway typically running an adapter (described in Section IV-B).

Each transducer in Sensor Andrew exists as an XMPP event node that has information published by an adapter on its behalf. Figure 2 shows an overview of the XMPP publish-subscribe system. Each event node contains an XML specification describing the transducer's capabilities. Event nodes can be hierarchically organized so that subscribers can be notified when particular related groups of nodes produce data. Figure 5 shows a typical sensor message from a FireFly wireless sensor node.

Many of the Sensor Andrew applications require support for mobility. XMPP inherently supports mobility with its addressing scheme and presence protocol. As gateway devices enter and leave the network, their presence is detected and logged. Mobile devices are specially tagged and published as event nodes in a separate pool. In cases where mobile nodes are not as resource-constrained as infrastructure nodes (for example, a cellular phone), they are given full JID addresses so that they can generate and receive XMPP messages with Internet clients.

The heterogeneous nature of Sensor Andrew requires supporting vastly different types of transducer devices. To this point, we have focused largely on resource-constrained or low datarate devices like wireless sensor nodes. Sensor Andrew must also support high datarate devices such as video streaming systems. For devices with high bandwidth requirements, XMPP offers a hand-off mechanism for establishing a secure link between two clients. Even though the datarate would be prohibitively high for the XMPP server to store all of the data, this handoff approach enables the server to catalog the type of data and which clients were involved in the transaction. This can be utilized later for searching purposes or for correlating other sensing events with the high bandwidth streams.

### *D. Gateway Layer*

The gateway layer consists of devices that typically have access to the Internet. As described in the next section, these devices run a full XMPP client with associated adapters for any attached transducers. Gateway devices have the ability to create and manage nodes for which they publish data. Devices at the server layer can subscribe to these event nodes, or they can directly address messages to the gateway. For example, a gateway in a classical wireless sensor network would publish sensor values for each sensor node in the system each of which is represented by a corresponding event node and accept configuration messages from Internet agents.

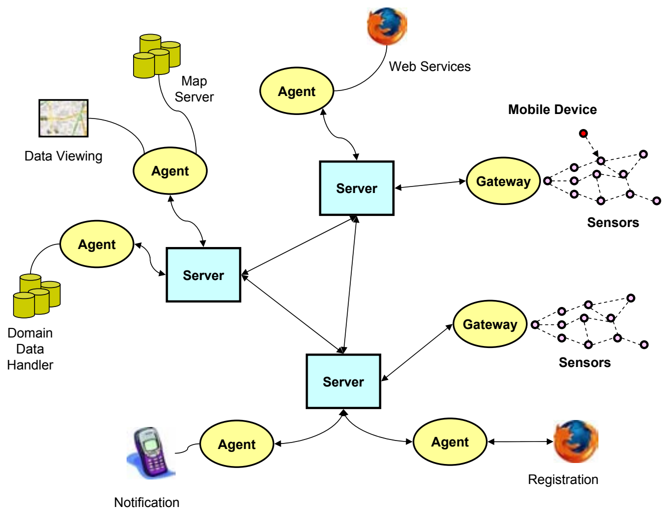

Fig. 3. Sensor Andrew services provided by multiple servers.

### *E. Server Layer*

The server layer consists of the XMPP servers along with various client applications called *agents*. The purpose of the server layer is to provide a simple means for applications running on desktop class machines to communicate with each other. Applications can not only subscribe to event nodes, but they can also publish their own *meta-events*. These meta-events can then be consumed and used by others.

### *F. Actuation Support*

Actuation in Sensor Andrew must deal with security and resource sharing. Actuation takes place as a split-phase operation with an action signal followed by a completion callback. First, gateway devices that support actuators are required to subscribe to their respective actuator event nodes. An agent can publish an actuation request to the event node which is then translated by the gateway's adapter into a native command for the actuator. Once the actuator operation has completed its transaction, a new state value is published back to its event node. An interested agent could subscribe to this event node to confirm the requested transaction.

## IV. System Components

We now describe the detailed implementation of the various components described in the previous section.

### *A. Sensors Over XMPP (SOX)*

We have built a Sensor Over XMPP (SOX) library as a layer on top of XMPP that provides a set of common tools as well as a uniform interface for all Sensor Andrew applications. Table I shows the current SOX command-line tools that wrap various API calls for simple use on any Unix-based computer. In order to support a variety of hardware platforms and operating systems, we have implemented the SOX library in C, .NET, LabVIEW, Java and Python. Figure 5 shows an example of a SOX message generated by a FireFly node, is a low-cost wireless sensor network platform capable of data acquisition, processing and multi-hop mesh communication.

### *B. SOX Adapters*

Adapters are pieces of software that convert transducer data into SOX compatible messages. These interfaces run on the gateway layer collecting and formatting information from the transducer layer. The following list outlines a subset of our current Sensor Andrew adapters and what host devices they operate on:

- **A Leech** is an agent that polls existing legacy databases searching for new entries for specified devices and publishes these to an event node. For many legacy systems with proprietary communication protocols, pulling information from a database is one of the few practical solutions for accessing the information. Currently, the Leech extracts data from Enersure [18] devices, from Trendpoint Systems, as well as various BACnet devices on campus.

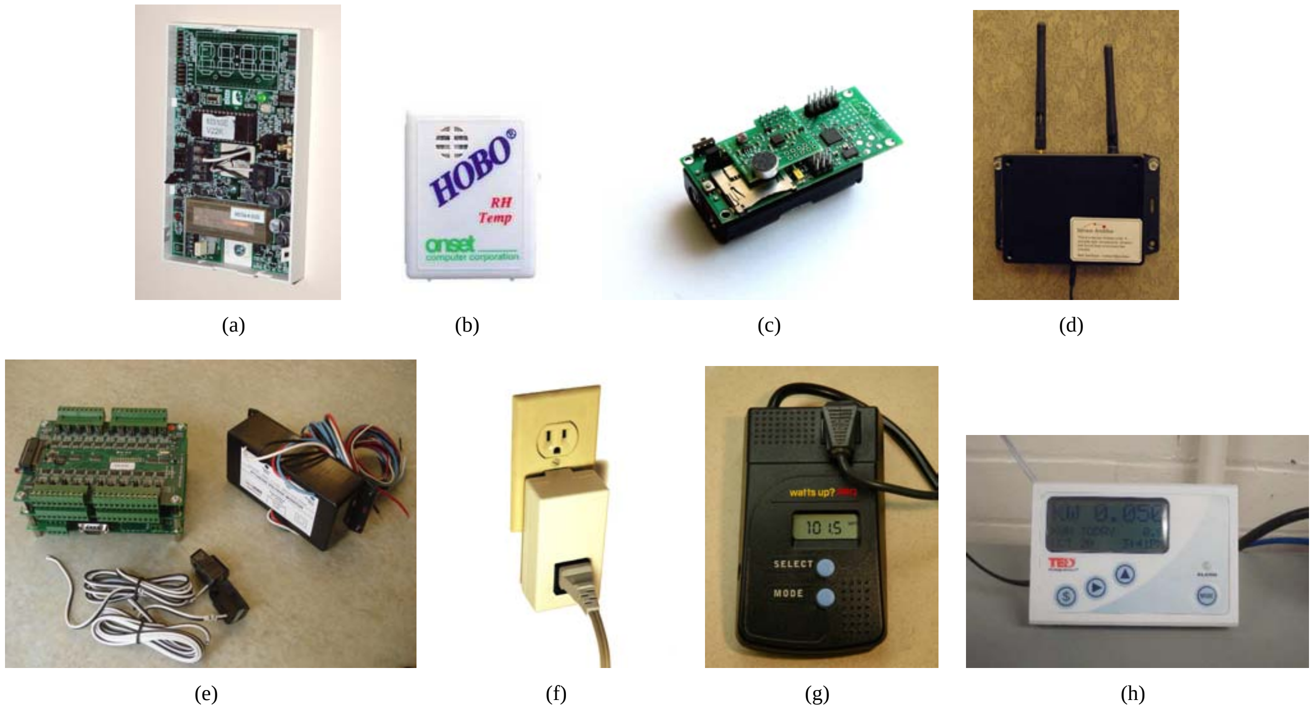

Fig. 4. Various Sensor Andrew transducer devices: (a) BacNET $CO_2$ sensor as part of a buildings HVAC system (b) Hobo temperature and humidity sensor (c) FireFly sensor node (d) FireFly gateway (e) Enesure circuit breaker power metering system (f) FireFly power outlet sensing and control node (g) Watts-Up? Pro outlet energy monitor (h) The Energy Detective home energy monitoring system. Note in (d) the Sensor Andrew privacy policy sticker required on any devices placed in public locations.

| SOX command | Function |
|---|---|
| `sox_create_event_node` | Create an event node for future data publishing |
| `sox_delete_event_node` | Remove an event node |
| `sox_add_publisher` | Add access for a user or group to publish |
| `sox_add_subscriber` | Add access for a user or group to subscribe |
| `sox_authenticate` | Check if user/group has access to a node |
| `sox_admin_create_user` | Create a user account |
| `sox_admin_delete_user` | Delete a user account |
| `sox_admin_create_group` | Create a group |
| `sox_admin_delete_group` | Delete a group |
| `sox_add_member` | Add user to a group |
| `sox_remove_member` | Remove a user from a group |
| `sox_get_last_data_from_node` | Return last data from an event node |
| `sox_publish` | Publish data to node |
| `sox_subscribe_example` | Subscribe to node dumping its data to the console |
| `sox_send_msg_to_user` | Send a message directly to a a user without pub-sub |

TABLE I
SOX COMMAND LINE TOOLS. THESE ARE WRAPPERS AROUND THE MAIN SOX API FUNCTIONS.

- **The FireFly Sensor Network** is comprised of wireless 802.15.4 devices [19] shown in Figure 4(c). Gateways, as shown in Figure 4(d), have adapters that execute on a Gumstix embedded linux board which bridges to the sensor network using RS232 serial communications. The adapter running on the FireFly node's gateway must facilitate bi-directional communication. Commands from SOX agents configure the network while outgoing sensor values are published.
- **The Energy Detective (TED)** produced by Energy Inc, shown in Figure 4(h) is a device used for monitoring home electrical consumption at the main household breaker. The adapter either runs on a PC or an embedded linux device wired directly to the TED.
- **Watts-Up? PRO** is a socket-level energy measurement device with a serial port for exporting data. This adapter runs on an embedded linux host or nearby PC.
- **HOBO** by Onset Computers, shown in Figure 4(b), is a low-cost USB or serial device that typically connects directly to a desktop computer. It is ideal for capturing environmental data near office computers around campus. The adapter is a Java daemon that executes in the background of Windows and Linux computers.
- **Other** devices such as mobile inertial sensors, desktop notification devices and IR transducers have been used in past experiments. Many short-term applications simply pass data to the SOX command line tools as an easy and effective way to get devices online.

*1) XMPP Server:* Sensor Andrew is designed to operate using a standard XMPP server that supports messaging as well as event publishing. For this reason, there are multiple viable enterprise class servers that can support our architecture. In our current deployment, we are using the *Openfire* [20] server from Ignite Real-Time Software. Openfire is an open-source server that provides a web interface for configuration and management. This interface allows us to quickly view gateways that are currently active and manage users. Access control for users is provided through access control lists

```
<DeviceInstallation id="0x000003A2" type="FIREFLY"
            timestamp="2008-05-10T10:23:00">
   <TransducerInstallation name="Light" id="0001" >
      <TransducerValue value="100"/>
      <TransducerDescription
            Manufacturer="Advanced Photonics Inc"
            PartNumber="PDV-P9003"
            MinValue="0.00" MaxValue="1024.0"/>
   </TranducerInstallation>
   <TransducerInstallation name="Temperature" id="0002" >
      <TransducerValue value="57.6"
            timestamp="2008-05-10T10:23:00"/>
       <TransducerValue value="58.1"
            timestamp="2008-05-10T10:21:30"/>
   </TranducerInstallation>
   <TransducerInstallation name="Voltage" id="0004" >
```

Fig. 5. SOX message for a FireFly node publishing data.

(ACLs) associated with users, groups and event nodes. XMPP supports methods allowing one client to determine and modify the permissions of another client. Security is managed as whitelists and blacklists associated with users and individual event nodes. Entries into each list are in full JID format (event nodes can have JID formatted addresses) allowing access control across multiple trusted servers. The details of how the server manages access control are outside the scope of this paper, but the interested reader can find more information about access control in the XEP-0074 extension of XMPP.

## C. Data Handler

The Data Handler is a web application that oversees all read/write activity on the transducer registry and transducer value archive. At the core, the Data Handler contains the registry and archive schema, business rules, and read/write functions. It was implemented in an object-relational mapping (ORM) framework in order to meld an object-oriented model-view-controller (MVC) library with a relational database. We chose to use the Python web application framework Django because of the many free components available and the ease of deploying the entire application–from database tables to URL re-writing rules–on several platforms, including the Google App Engine. Using this library, several tools were created to support the core interactions with the registry and archive.

*1) Schema:* The transducer registry and data value archive schema defines the relationships among the numerous kinds of metadata that Sensor Andrew supports. As shown (at a high level) in Figure 6, the schema describes how a transducer is installed on a device at a location in order to measure one or more data values. The transducer installation (or TI) consists of a particular instance of a transducer, which is of a type that senses a particular physical phenomenon. The transducer is installed on a device (which typically performs the analog-to-digital conversion and communicates with the gateway or acts as a gateway itself). The transducer and device can be situated within arbitrary-depth location hierarchies, which can support building-floor-room schemes as well as more complex space divisions, and are also referenced to a common longitude/latitude/altitude coordinate system. Transducers can also be ”tagged” as relating to one or more physical ”systems”, denoting what different users consider the transducer to be measuring or controlling, such as a zone within a heating system or a circuit in a lighting system. Finally, one or more data values are linked to a device timestamp. Storing this transducer metadata in a structured fashion is important, as different users might need to query the system in different ways, such as finding temperature sensors in a particular building, investigating which device is currently hosting a particular actuator, or checking to see which of the user's sensors have not reported values recently.

*2) Registry interface:* Most users will not find it practical to build their own interface to the registry and archive, so a web interface was constructed to allow browsing, editing, and creating transducer and device metadata records in the registry. It is intended to help users to understand how to map metadata values that they must gather from various specification sheets and configuration files onto the schema. The registry interface will guide them through the data entry process in order to lower the barrier to participation in the Sensor Andrew network. It also allows users to browse and search existing transducers and archived data.

*3) Web services API:* For users wishing to build custom applications that need to read or write transducer metadata in the registry, a web services API exposes the Data Handler's functionality to HTTP and XMPP requests. This API supports transducer discovery queries, requests for detailed transducer metadata, and registry record creation and updating, as well as serving transducer data values from the archive. Results can be formatted in XML, CSV, or human-readable HTML, as well as graphic charts for time-series data. A version of the web service's HTTP URI-based commands is mapped to an XMPP API, allowing XMPP clients to interact with the registry and archive without needing an HTTP library.

*4) Historical data logging:* While some users will use their own XMPP clients to log transducer data for historical analysis, this is considered such a core feature that it is offered as part of the Data Handler. If this option is selected when registering the transducer installation, the Data Handler subscribes to the XMPP event node for that transducer and archives all data values published to the event node. The historical data is then available via the web interface or API to only those users whose JIDs are authorized to subscribe to the event node from which the data originally came.

*5) XMPP server integration:* The Data Handler inherits the XMPP server's permissions model by requiring users to log in with a valid JID, which it then uses to authenticate them with the XMPP server. This also allows the Data Handler to send requests to the XMPP server on the user's behalf, such as creating event nodes of which the user, not the Data Handler, is the owner. The Data Handler can also publish ”node creation” events on the administrative event node, as well as sending and receiving other XMPP messages and commands.

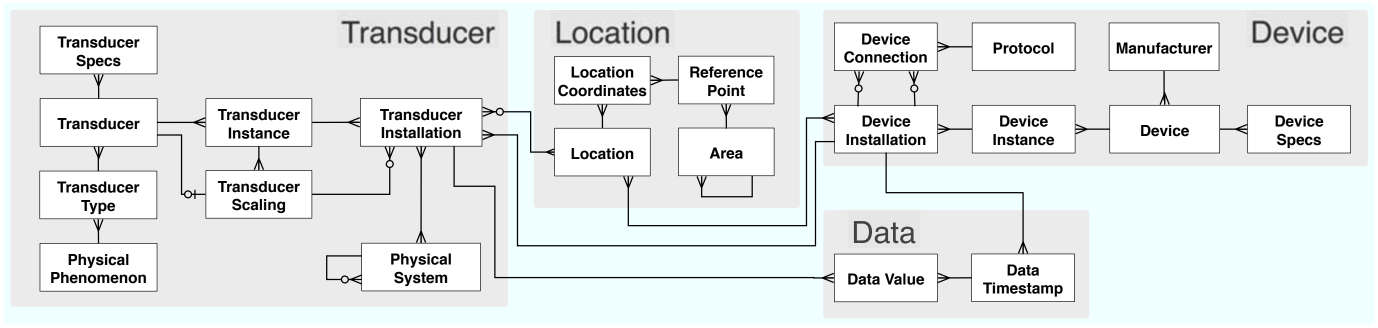

Fig. 6. Sensor Andrew Transducer Registry Schema.

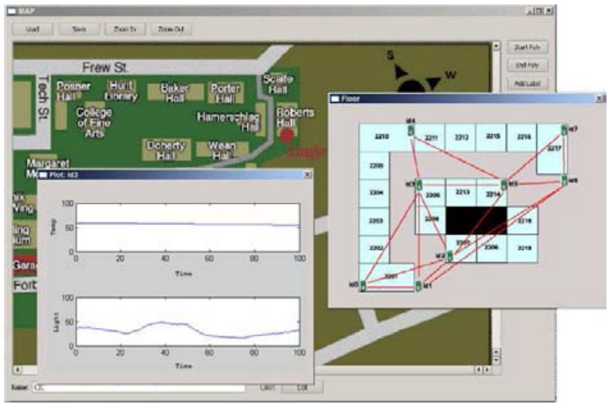

Fig. 7. *SenseView* screenshot displaying sensor values for a node on campus.

## D. Server Layer Agents

We next describe example applications that run as SOX agents subscribed to various event streams. While ongoing, these are two early applications.

*1) SenseView: SenseView* is a tool that enables hierarchical and visual browsing of physical location information and sensor values. Visual maps can be created by composing polygons, each with the ability to link to a different view. Access to real-time data is provided by directly subscribing to event nodes captured as links in the map. The event nodes also provide attribute information describing the sensors. Map information is fetched from a dedicated map server with its own access control lists based on SOX authentication. Much like a web browser with hyperlinks, *SenseView* allows a user to traverse through different views by clicking on different parts of the map. The user can select and subscribe to available event nodes given the correct permissions. Once subscribed to a data source, *SenseView* graphically displays data as it is being published to the XMPP server.

Figure 7 shows a screenshot of *SenseView* with the top-level campus map displayed. Maps can be customized based on address and namespace providing application specific views. It is also worth noting that event nodes are not required to be literally sensors. They could also be higher level meta-events as described in the next section.

*2) Event Notification System:* The event notification system is an application that allows events to be combined in order to form more complex events. For example, a `floor-fire` event could be defined as the combination of multiple temperature sensors located throughout the floor raising above a particular threshold. Users can interact with the system to combine primitive event values together using Boolean operators to develop meta-events. These meta-events can then be published back to the XMPP server where they are eligible to be combined with additional events forming an event hierarchy. Our current implementation of this event system has an additional client that is responsible for monitoring a subset of events and notifying users via email, text message or a webpage when selected conditions change.

## E. Security and Privacy

Given the physical nature of the information collected and exchanged throughout Sensor Andrew, one has to be naturally concerned about security and privacy. Our privacy mechanisms range from technological solutions, such as encryption, to informal policies such as proper information distribution to the social community and labeling of devices. The Principal Investigator of each project that is part of Sensor Andrew is required to sign our privacy policy and go through a checklist to ensure that they are compliant with the privacy policy as well as the university's Interanl Review Board (IRB) requirements. Devices placed in public areas must clearly display what information they are capturing and where further information about the project can be located.

Furthermore, built-in security measures in the architecture are extremely important. All server-layer communication takes place over XMPP using SSL connections. All client applications are also required to authenticate with a user-name and password. Any guest access to the network is automatically restricted by the access control lists to anonymous data that could not be used to identify individuals. Whenever possible, security is used within individual subnets. For example, our example wireless sensor networking deployment uses encryption for all infrastructure communications.

## F. SOX Specific Enhancements

Anytime a system leverages existing components there are bound to be technical as well as design incompatibilities. So far in this section, we have described many of the technologies selected for each layer. Now we summarize some of the enhancements to these technologies required for Sensor Andrew. We developed the SOX library and data schema required to use XMPP for sensor data. This included an extensive set of adapters required for interfacing with transducers. Modification of the XMPP server was required to add group permissions to publish-subscribe event nodes since by default access control only applies to users. We are suggesting this ACL addition along with our schema as part of a SOX extension protocol to the XMPP community. As described in Section V-B, we built a lightweight XMPP message protocol for compatibility with highly resource-constrained devices like wireless sensor nodes. We developed an extensive set of core SOX agents for registration, discovery, logging and viewing of sensor data. We also provide application agents for tracking mobile devices and alerting users of events. The Sensor Andrew infrastructure allows the addition of new transducers to seamlessly work with existing applications.

### G. *Limitations*

This subsection addresses some of the limitations in the Sensor Andrew architecture. Our requirement of transaction logging forces all messages to go through a server even if the action could be completed with a point-to-point transaction. To help alleviate bottlenecks, a unique server can be used for a single domain of interest. For high-speed sensor data, there is currently no database logging capability. A transaction record exists, but the bandwidth of streaming data would be too much burden on the main message server. Our current permissions model does not support fine grained access control over individual record elements; it only provides access control at the event node level.

## V. Early Experiences

In this section, we evaluate various aspects of the Sensor Andrew architecture. The goals identified in Section II have been largely satisfied by the various components in the architecture. We have provided a communication layer which can support a large number of clients and continue to scale through distributed server addressing. Leveraging existing enterprise servers makes deployment and management of this infrastructure highly configurable and easy to manage. The generic nature of our XML-based messages allows for extensibility making it simple to extend our schemata for new devices. The architecture supports privacy and security through the use of encrypted connections and access control for users and groups.

Given these solutions, we now discuss where our architecture fits into the full design trade-off space. Our design tends to be more push-oriented and hence slightly more centralized compared to a fully distributed configuration. A push architecture has advantages when large volumes of requests are made to resource-constrained devices. These devices could have large latencies and restrictions on how often they can be sampled over any particular time period. A pull architecture that polls devices on demand would be ideal if few and/or sporadic requests are made to devices. This could potentially be more energy-efficient if, for example, the push architecture was transmitting data without any active subscribers. If required, XMPP does have functionality to initiate out-of-band communication and can be used to support pull operations.

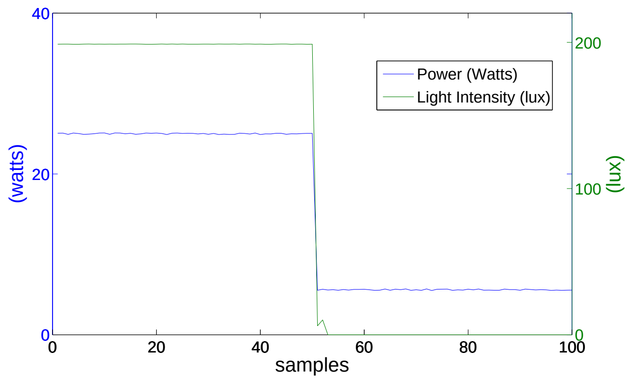

Fig. 8. Power values from a circuit breaker that is feeding lights to a conference room, accompanied by light intensity measurements of the same space.

### A. *Building Energy Monitoring*

The first application we present focuses on accomplishing energy savings through non-intrusive electrical load monitoring of a building and user feedback. The premise here is that better decisions can be made towards saving energy if the operational schedule of most of the appliances in a building is known. One way of determining this schedule is to install electrical meters on each of the appliances and then have them all networked together and reporting to a central location. Another approach is to install a single electrical meter at the main feed of the building, and by making use of signal processing and statistical pattern recognition techniques, decompose this total load into the individual appliances composing it. The project we describe follows the latter path, with the rationale that the approach reduces hardware and labor costs, while possibly providing the same results, as shown in [21].

For this application a number of different commercial and research-grade sensing devices are used: (1) Two electrical panels located in the Porter Hall building on the campus were retrofitted with EnerSure [18] electric circuit monitors, shown in Figure 4(e). These devices acquire different power metrics from all the individual circuits in the panel, and can be polled via TCP/IP or RS-232 using Modbus. (2) A number of FireFly power sensing nodes, shown in Figure 4(f), are used to measure the consumption of separate appliances throughout the building, as a way to obtain ground truth data. (3) Other power meters are used intermittently to obtain load profiles for certain appliances and/or as ground truth sources. (4) Lastly, there are a few light intensity and temperature sensors scattered across different rooms in the building, which can provide useful information that will help the disaggregation task (e.g. if a light is turned on, it will manifest itself as both a power draw and as a light intensity change).

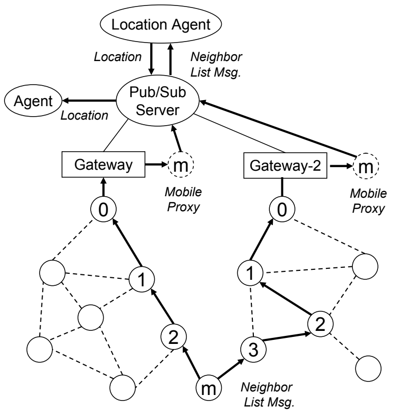

Fig. 9. Mobile node communicating with Sensor Andrew location agent using a wireless sensor network.

Load disaggregation is a difficult task, and the non-intrusive approach requires efficient use of the hardware and data that is already available in the building. Sensor Andrew provided valuable resources to help achieve the goals of this project mainly by allowing easy access to the different sensors located throughout the building which were put in place by other researchers for other projects. Additionally, Sensor Andrew acted as a central data and metadata repository for all the metering records which the project generates. Data from the EnerSure devices were being acquired by a Perl polling-agent [22] and logged in a MySQL database. The Leech adapter was used to make data from these devices available to the Sensor Andrew network. Similarly, XMPP adapters were used to publish data from the other power meters (FireFly, Watts Up? PRO, The Energy Detective), as well as the environmental sensors (HOBO, FireFly).

The graph presented in Figure 8 shows power measurements obtained from an electrical circuit in the building that is feeding the lights of a room, overlapped with the readings obtained from a light intensity sensor placed in the same space.

### B. Wireless Sensor Networking

In this section, we describe an effort that both senses the environment and provides a pervasive communication infrastructure for mobile devices. One promising application of the network is the ability to track the location of mobile sensor nodes. These nodes could either be carried by students as part of a social network or tags attached to valuable equipment for asset tracking.

Multiple buildings across campus have been outfitted with FireFly wireless sensor networking nodes. Each node operates from two D-cell batteries and communicates over multiple hops to a powered gateway that has access to the campus network. FireFly nodes are primarily used to collect and publish light, temperature, acceleration, noise level, battery voltage and network topology values once every four minutes using the SAMPL [23] networking protocol. SAMPL also provides a generic communication interface allowing nodes to directly query the infrastructure nodes as well as send abbreviated SOX messages to and from Sensor Andrew via the gateway. Each gateway manages between 20 and 32 nodes that form a subnet. As a mobile node moves through campus, it can send *ping* messages to identify nearby infrastructure nodes. SAMPL's lightweight SOX message type allows a mobile node to securely send its password, an arbitrary 100 byte payload and a destination JID to the gateway. As shown in Figure 9, the gateway can login on behalf of the mobile node and forward this data to its destination. The message includes a disconnect timeout so that reply messages can be forwarded back to the mobile device. As is depicted in Figure 9, if the mobile node is between two subnets, then both gateways will login on the device's behalf and arbitrate messages. This provides an effective handoff mechanism. Also, since XMPP provides presence information when a user logs in, agents can be notified when a mobile node becomes connected to the network.

Based on motion activity levels derived from an accelerometer, mobile nodes can aggregate neighbors and transmit a message through the sensor network to a Sensor Andrew location agent. This location agent will then use the neighbor list and receive signal strength information from the mobile node to provide a coarse-grained location of the mobile device which it can publish back to the mobile devices' event node. In turn, other interested agents can subscribe to this information to identify the location of any number of mobile devices.

## VI. Conclusions and Future Work

In this report, we presented a multi-disciplinary campus-wide scalable sensor network called Sensor Andrew that is designed to host a heterogeneous mix of sensing and low-power applications. We presented the requirements, goals and design tradeoffs associated with such large-scale heterogeneous sensing and actuation systems. Specifically, the goals of Sensor Andrew are to support ubiquitous large-scale monitoring, operation and control of infrastructure in a way that is extensible, easy to use, and provides security while maintaining privacy. Our architecture provides a complete communication framework allowing new projects to easily be integrated with existing projects so as to extend overall capabilities. A three-tiered architecture allows for ease of management, and facilitates security and privacy controls. Open-source software customized and integrated with our extensions enables seamless and scalable communications across layers.

As future work, we plan to enhance this architecture by providing support for end-to-end real-time applications to better support industrial automation. This would include resource-reservations on various communication and computational components along with prioritized message scheduling. Efforts are already underway to streamline the interface for registering, configuring and querying sensors through web services. We plan to continue development of tools and applications as we continue to explore real-world sensing and actuation applications making datasets publically available for the research community.

## VII. Acknowledgments

This work would not be possible without the help and support of many Sensor Andrew project members. In particular we would like to thank Burcu Akinci, H. Scott Matthews, Bruno Sinopoli, Priya Narasimhan, Matthew Sanfilippo, Anu Pradhan, Pine Liu, Karthik Lakshmanan, and Saurabh Taneja. This work was funded in part by the Pennsylvania Infrastructure Technology Alliance. Sensor Andrew is a project of CENSCIR, the Center for Sensed Critical Infrastructure Research at the Institute for Complex Engineering Systems (ICES) at Carnegie Mellon University. We would also like to thank Bosch Research and Technology Center for their support for the electricity monitoring project.

## References

[1] Mainwaring A., Polastre J., Szewczyk R., Culler D., Anderson J. Wireless Sensor Networks for Habitate Monitoring. *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[2] K. Lorincz ,M. Welsh. A robust, decentralized approach to rf-based location tracking. Technical Report TR-04-04, Harvard University, 2004.

[3] Culler D., Dutta P., Tien Ee C., Fonseca R., Hui J., Levis P., Polastre J., Shenker S., Stoica I., Tolle G., Zhao J. Towards a Sensor Network Architecture: Lowering the Waistline. *Proceedings of the Tenth Workshop on Hot Topics in Operating Systems (HotOS X)*, 2005.

[4] Heidemann J., Silva F., Intanagonwiwat C., Govindan R., Estrin D., Ganesan D. Building Efficient Wireless Sensor Networks with Low-Level Naming. *SOSP*, 2001.

[5] Montenegro G., Kushalnagar N., Hui J., Culler D. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. *Internet Engineering Task Force RFC 4944*, 2007.

[6] Santanche A., Nath S., Liu J., Priyantha B., Zhao F. . SenseWeb: Browsing the Physical World in Real Time. *Demo Abstract, IPSN*, 2006.

[7] Sheth, A. Henson, C. Sahoo, S.S. Semantic Sensor Web. *IEEE Internet Computing*, 2008.

[8] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 2002.

[9] Laibowitz M., Gips J., Aylward R., Pentland A., Paradiso J. A Sensor Network for Social Dynamics. *International Conference on Information Processing in Sensor Networks (IPSN)*, 2006.

[10] Want, A., Jones, A., Hopper, A. A New Location Technique for the Active Office. *IEEE Personal Comm.*, 1997.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Operating Systems Design and Implementation (OSDI)*, 2002.

[12] http://www.bacnet.org/ (viewed 10/21/2008).

[13] http://www.obix.org/ (viewed 10/21/2008).

[14] Woo Suk Lee, Seung Ho Hong. KNX-ZigBee gateway for home automation. *IEEE International Conference on Automation Science and Engineering*, 2008.

[15] Neugschwandtner M., Neugschwandtner G., Kastner W. Web Services in Building Automation: Mapping KNX to oBIX. *5th IEEE International Conference on Industrial Informatics*, 2007.

[16] Gibbons, P. B., Karp, B., Ke Y.,Nath, S., Seshan, S. . IrisNet: An Architecture for a Worldwide Sensor Web. *IEEE Pervasive*, 2003.

[17] http://www.xmpp.org/ (viewed 4/12/2008).

[18] http://www.trendpoint.com/EnerSure.html (viewed 10/21/2008).

[19] Rowe A., Mangharam R., Rajkumar R. FireFly: A Time Synchronized Real-Time Sensor Networking Platform. *Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks, CRC Press Book Chapter*, 2006.

[20] http://www.igniterealtime.org/projects/openfire/ (viewed 4/12/2008).

[21] Shaw S., Leeb S., Norford L., Cox R. Nonintrusive Load Monitoring and Diagnostics in Power Systems. *IEEE Transactions on Instrumentation and Measurement*, 57(7):1445–1454, 2008.

[22] http://www.klein.com/thermd (viewed 10/21/2008).

[23] Rowe A., Lakshmanan K., Rajkumar R. . SAMPL: A Simple Aggregation and Message Passing Layer for Sensor Networks. *International Wireless Internet Conference, WICON*, 2008.