Towards a Zero-Configuration Wireless Sensor Network Architecture for Smart Buildings

Lars Schor, Philipp Sommer, Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
{lschor, sommer, wattenhofer}@tik.ee.ethz.ch

Abstract

Today's buildings account for a large fraction of our energy consumption. In an effort to economize scarce fossil fuels on earth, sensor networks are a valuable tool to increase the energy efficiency of buildings without severely reducing our quality of life. Within a smart building many sensors and actuators are interconnected to form a control system. Nowadays, the deployment of a building control system is complicated because of different communication standards. In this paper, we present a web services-based approach to integrate resource constrained sensor and actuator nodes into IP-based networks. A key feature of our approach is its capability for automatic service discovery. For this purpose, we implemented an API to access services on sensor nodes following the architectural style of representational state transfer (REST). We implemented a prototype application based on TinyOS 2.1 on a custom sensor node platform with 8 Kbytes of RAM and an IEEE 802.15.4 compliant radio transceiver.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Sensor Networks, Energy Efficiency, Implementation

1 Introduction

Improving the energy efficiency of buildings is an important step towards a more sustainable lifestyle, leading to significant cost savings for real-estate owners. Nowadays, energy use in buildings is responsible for roughly 40 % of total energy use and for 36 % of the CO_2 emissions in the EU area [2]. Recently, the European Commission pledged to cut the annual consumption of primary energy by 20 % by 2020. While recent advances in the field of material science led to an improved energy efficiency of the building envelope, a lot of energy is consumed by different equipment such as HVAC (heating, ventilating, and air-conditioning), lightning, home and office appliances. In a first step, decentralized sensor nodes of different types are required to report the current energy usage or environmental conditions to a centralized monitoring system. A smart power outlet will report the current energy usage of the attached device to a central server. Temperature sensors are used to react upon variations in room temperature. Actuator nodes are responsible to control small subsystems within the building. A presence sensor can automatically switch off the ceiling lightning when an employee has left its office. However, even more energy can be saved when different subsystems cooperate with each other. For example, the central control system can switch off the office lights, lower the heating and send an employee's PC to standby mode when the door system reports that the employee has left the building.

In this paper, we present an approach to integrate tiny wireless sensor or actuator nodes into an IP-based network. Sensors and actuators are represented as resources of the corresponding node and are made accessible using a RESTful web service. While many existing approaches provide web services at a smart gateway, we show that it is feasible to provide web services at each node, even when using a very resource-constrained hardware platform. We present a prototype implementation on a mote-class hardware and evaluate the performance of our approach.

2 System Architecture

Building automation and control systems rely on many sensors and actuators placed at different locations throughout a building. Reducing the power consumption of a modern building requires continuous monitoring of various environmental parameters inside and outside the building. The key requirement for an efficient monitoring and controlling is that all sensors and actuators are addressable over the network. Different proprietary protocols and industry standards for building automation have been proposed over the years, e.g., CEBus, EIB, BACnet or Local Control Network (LCN). However, integration of different services based on the Internet protocol (IP) suite has become an important trend during recent years. While standard IP-based protocols introduce some overhead compared to a customized protocol, the increased possibilities to connect different devices certainly pay off. Although we are using tiny devices, we still want to

BuildSys'09, November 3, 2009, Berkeley, CA, USA.

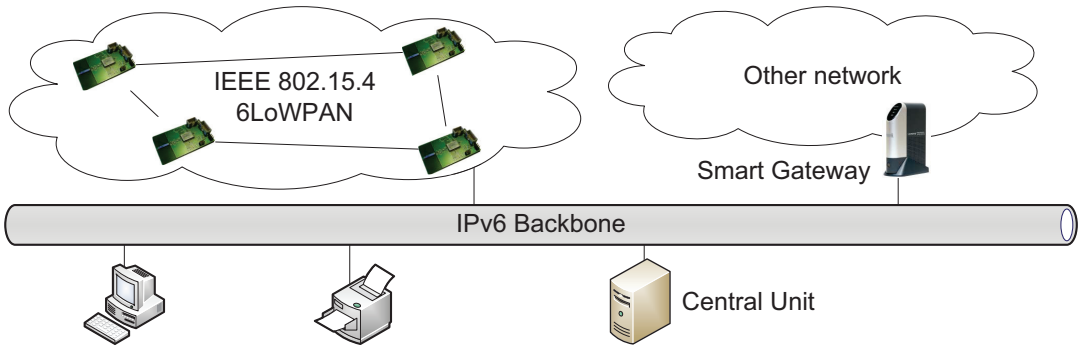

Figure 1. Architecture of a monitoring and control system for smart buildings. 6LoWPAN-enabled wireless sensor nodes are directly integrated into the IPv6 network (left). Access to non IP-enabled devices is provided through a smart gateway at the network boundary (right). Sensors and actuators are accessible by a central unit.

use the standard protocols which made the Web so powerful. It is even possible to run proprietary protocols in certain parts of the network. A smart gateway at the network boundary is then used to provide access for IP-based protocols. As a large number of devices may be placed at different locations throughout the building, connecting them using wires, e.g., by Ethernet, is often not practical. The IEEE 802.11 wireless LAN standard solves the problem of laying wires, but the energy consumption of the radio transceiver makes it infeasible to operate devices on batteries. We address this problem by employing a wireless sensor network based on the IEEE 802.15.4 physical layer standard which is optimized for energy efficient communication with low data rates. The 6LoWPAN [12] header compression scheme allows to efficiently send IPv6 packets over IEEE 802.15.4-based networks. The overall architecture of the system presented in this work is outlined in Figure 1.

2.1 Configuration and Service Discovery

Auto-configuration of network nodes is a mandatory feature when managing large networks. We employ the stateless address auto-configuration mechanisms provided by the IPv6 protocol. After startup, a node sends a *router solicitation* request to the link-local multicast address. A router will respond with a *router advertisement* message containing configuration parameters for the current network. Otherwise, a sensor node can obtain its IPv6 address by issuing a DHCP request. Once the address configuration phase has been completed, a node starts to advertise its offered services to the other nodes in the network. We use the *Multicast DNS (mDNS)* protocol [1] proposed by Apple which is also known as Bonjour (formerly Rendezvous). mDNS uses special DNS resource records to provide information about services offered by a device such as the service type, domain name and additional configuration parameters. A node can query a certain service type by sending a DNS packet to the link-local multicast address. Nodes which offer services of the corresponding type will send an answer. mDNS is implemented in many existing network peripherals, e.g., printers, scanners and for different operating systems (currently mainly Mac OS and Linux).

2.2 Web Services for Wireless Sensor Nodes

Web services allow the interaction between different devices over the network in order to exchange data or to trigger certain actions. Data is exchanged between peers using the *Hypertext Transfer Protocol (HTTP)*. Traditional enterprise solutions tend to use the *Simple Object Access Protocol (SOAP)* together with XML for the data representation. Recently, a more lightweight solution called *Representational State Transfer (REST)* has been proposed which is built on top of the *GET*, *POST*, *PUT* and *DELETE* methods of HTTP [6]. Web services following these principles are denoted as *RESTful web services* in the remainder of this paper.

2.2.1 Data Access Schema

Sensor nodes are responsible to provide information about their sensors and actuators to the central monitoring server. One possibility to guarantee that the server has up-to-date information is that the sensor node periodically sends status messages to the monitoring server. An alternative approach would be that the server polls the state of the sensors according to demand. While the first approach is preferable if no actuators are connected to the sensor node, the second variant offers more flexibility since the same interface can be used to read out the state of a sensor as well as to modify the state of actuators. The requirements of our Web-based API are similar to the second option, which is preferable for the implementation.

A second architectural decision affects the device where the web service has to be implemented. Two options are common in literature: either implementing the API in a gateway, which connects the sensor nodes with the Internet, or directly implementing the API on Web-enabled devices. The second option offers more flexibility since no changes to the gateway functionality are required when deploying new device types.

2.2.2 RESTful API for Sensor Nodes

Our approach implements a RESTful Application Programming Interface (API) on sensor nodes to provide access to sensors and actuators through the Web. As each sensor node may connect with different sensors and actuators, manual configuration of a central monitoring system is infeasible if a large number of sensor nodes are used. Based on the RESTful API, we introduce a plug-and-play approach,

```
/                           (root collection)
/temperature/*              (collection)
/temperature/celsius        (member)
/temperature/fahrenheit     (member)
/humidity/*                 (collection)
/humidity/relative          (member)
/humidity/absolute          (member)
```

Figure 2. Example for the structure of resources provided by a RESTful web service on a sensor node with an integrated temperature and humidity sensor.

which enables the automatic discovery of sensor nodes in a wireless network, but also of the functionality they provide.

The Representational State Transfer (REST) protocol [6] can be outlined as a collection of network architecture principles and is nowadays used by various Web 2.0 applications to offer their functionality over an API. Unlike other web services, such as SOAP, which rely on other application layer protocols, REST uses the HTTP protocol as application platform. The functionality of a system is implemented as a set of resources that can be identified using the corresponding URI; thus both the sensor node and the temperature sensor get an own URL. A RESTful web service is a collection of resources. By using the four basic operations provided by the HTTP protocol (GET, PUT, POST and DELETE), clients can interact with the resources. Each operation executes a corresponding action like updating a resource or creating a new entry in a collection. Its lightweight stack simplifies the integration of REST in devices with limited resources such as sensor nodes [13]. Not only the device is a resource, but all sensors itself provide a resource that can be accessed over the network. As web services are not limited to sensor nodes, they can be used in any devices integrated in the monitoring and controlling system of a smart building.

2.2.3 Resource Discovery and Access

As an individual sensor node potentially offers many different resources, the device has to provide functionality for automatic discovery of resources. Therefore, the RESTful API provides a modular structure as outlined in Figure 2. The API differs between three types of resources: ROOT COLLECTION, COLLECTION and MEMBER. The ROOT COLLECTION offers a list of all collections the sensor node provides, the COLLECTION can be addressed to get a list of all its members while the actual functionality of a sensor or actuator is implemented as a MEMBER resource. This modular design enables another device such as a central unit to autodiscover the functionality offered by a sensor node. As the server can only request the information it is interested in, the message sizes are kept small, and the number of messages is reduced. The response to a GET operation is presented in the JavaScript Object Notation (JSON) which is a lightweight and platform independent data exchange format. Parsing and creating the JSON representation of data requires less overhead on resource constrained sensor nodes compared to the XML format.

3 Implementation

In this section, we describe the implementation of a RESTful communication protocol for wireless sensor networks. While the proposed API can easily be implemented on systems such as PCs, printers or servers, various limitations have to be considered for the implementation on resource constrained wireless sensor platforms.

3.1 Hardware Platform

The hardware platform used for the implementation of the proposed communication protocol is a custom design based on the ZigBit900 MCU wireless modules from Atmel. The ZigBit modules are featuring an ATmega1281 microcontroller and 802.15.4/ZigBee compatible radio module (Atmel ATRF212) integrated into a small form factor. The ATmega1281 microcontroller comes with 8 Kbytes RAM and 128 Kbytes program ROM. The RF212 radio module is compatible with the IEEE 802.15.4 standard for low data rate communication. We built the *Pixie* prototype platform, shown in Figure 3, which is based on the ZigBit900 module; it provides connectors for different peripheral devices. The RF signal can be connected to the on-board chip antenna or to an external antenna. The small form factor and the various possibilities to connect additional sensors or other peripherals make the Pixie nodes an ideal prototyping platform for wireless sensor network applications. Based on the prospective location of the sensor node in a deployment, one might attach different sensors or actuators to the node. The Pixie platform provides extension headers to connect additional components through GPIO pins or the I2C, SPI or UART interface. This extensibility in combination with the resource discovery mechanism presented in Section 2.2.3 makes the Pixie nodes a convenient platform for prototyping sensor networks for smart buildings.

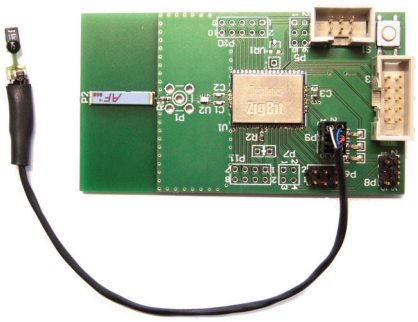

Figure 3. The Pixie node is a flexible prototyping platform which is equipped with extension headers for external peripherals. In this picture, a Sensirion SHT75 temperature/humidity sensor is attached to the node.

3.2 TinyOS Implementation

In this section, we discuss how the RESTful API can be efficiently implemented targeting sensor nodes having only 8 Kbytes of RAM and running on batteries. We ported the ZigBit module to the TinyOS 2.1 operating system which required only small modifications due to the similarity of the ZigBit module to existing platforms. Implementing web services on sensor nodes introduces three additional layers: IP,

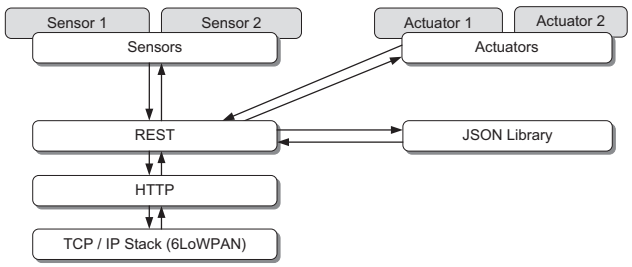

Figure 4. TinyOS implementation of the RESTful API. The HTTP and REST layer on top of the TCP/IP communication stack provide the RESTful API with help of an additional JSON library.

```
{
  "device": "temperature",   // name of the resource
  "method": [                // supported methods
    "G"                      // of the resource (GET)
  ],
  "param": [                 // array with all parameters
    {
      "n": "celcius",        // name
      "v": 26,               // value
      "t": "i",              // data type (integer)
      "u": 0                 // updatable
    }
  ]
}
```

Figure 5. Example of a JSON object sent by the RESTful web service on the node in response to a GET request.

TCP and HTTP layer. We address the overhead introduced by each of these layers and discuss approaches to minimize the resource costs in order that the implementation fulfills the requirements of a low-power sensor node.

In order that other devices can directly connect to the sensor nodes, they have to be IP-addressable. As the transmission of IPv6 packets over an IEEE 802.15.4 network introduces a large overhead in the message length, further compression is necessary. The 6LoWPAN standard addresses this problem by introducing an adaptation layer to compress the IPv6 header and to fragment IPv6 packets in order to enable IPv6 communication in WSNs [9, 10]. Our implementation is based on `blip`, the 6LoWPAN stack included in TinyOS 2.1.1. We use TCP for establishing reliable data connections between the sensor nodes and the central unit. Various aspects to reduce the overhead introduced by using TCP as transport protocol are discussed in [15]. In particular, by using persistent TCP connections, the latencies of a request can drastically reduced. As a typical response of a sensor node consists only of two or three data packets, the establishing and closing of a TCP connection provides a larger overhead. As the central unit regularly connects to the sensor node, persistent TCP connections reduce the resource costs in terms of the number of messages.

3.2.1 Service Advertisement

We implemented service discovery according to Apple's mDNS proposal. Each node periodically sends a DNS packet over UDP to the IPv6 link-local multicast address to announce the presence of its RESTful web service in the network. We introduced the new service type "`_rest-sensor._ws._tcp`" for the RESTful web service on the sensor nodes. To reduce the complexity of the implementation on the sensor nodes, we only send out announcements; we omit the functionality for processing incoming mDNS packets from other hosts. Since mDNS packets are sent to the link-local address, they are not automatically forwarded in a wireless multi-hop network. Therefore, intermediate nodes have to relay incoming mDNS packets towards the border router.

3.2.2 Web Services Implementation

The final goal we are pursuing is to access the sensors and actuators of a low-power device using a high-level API. For this purpose, we implemented the RESTful API described in Section 2 on sensor nodes using the 6LoWPAN communication stack described above. Following the layering outlined in Figure 4, the API has been implemented in TinyOS.

Targeting the Pixie nodes having only a small amount of RAM, we implemented a lightweight HTTP layer serving basic tasks of a web server. This layer is the link between the communication stack and the RESTful API. The web server forwards incoming requests to the REST parser and provides the basic functionality to generate HTTP packets. The key component of our implementation is the REST layer which dispatches incoming requests to the handler method of the corresponding collection, thus mapping a resource to an URI. Furthermore it generates the response to a root collection request. As the manual generation of JSON objects would be a huge overhead and prone to error, an additional library has been developed to support the reformatting of a response as a JSON object.

A sensor or actuator needs only to bind itself with its collection name to the REST layer and to implement an event-handler for receiving a GET respectively PUT request in order to provide its functionality in the RESTful API. Receiving a GET request for a resource, the REST layer forwards the request to the related sensor or actuator. A sensor answers to a GET request with a JSON object containing one or more parameters as outlined in Figure 5. The JSON library helps to create JSON objects by offering services which automatically generate the JSON envelope.

PUT requests are supported to interact with actuators. The corresponding parameters are added to the PUT request as raw data lines as the implementation of a JSON parser would generate a large overhead on a sensor node. When receiving a PUT request, the actuator needs to parse the parameters and updates its state according to their values.

The whole implementation is fully platform independent. Compiling the RESTful API for the Pixie platform with various sensors and actuators connected to the device, the RAM footprint has a size of 7.6 Kbytes while the ROM footprint, i.e. the amount of Flash required, has a size of 43 Kbytes. Only 0.7 Kbytes of RAM and 2 Kbytes of ROM are used by the RESTful API and the mDNS implementation while the rest is mainly used for the TCP and 6LoWPAN implementation.

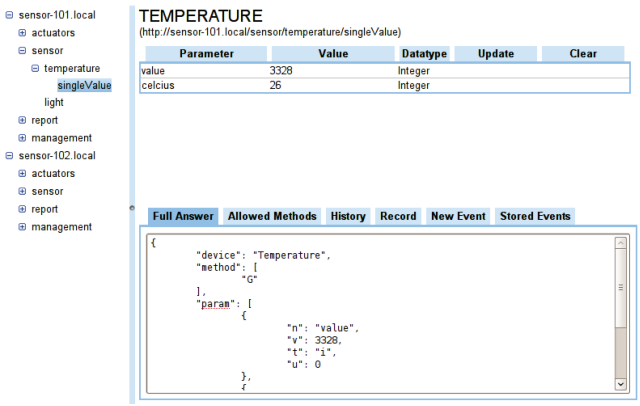

Figure 6. Screenshot of the web interface provided by the central unit. The set of resources provided by the sensor nodes are displayed in a tree view (left). Once a resource is selected, properties can be viewed and edited in the main panel (right).

3.2.3 Energy Efficiency

Reducing the energy consumption of the nodes in the sensor network itself is a crucial aspect in our system. Although external power is in general available in buildings to operate the sensor nodes, one might equip nodes with batteries to reduce the cost of the wiring, and to increase the flexibility in the node placement. While the microcontroller of a node can be placed in power-save mode when idle, the radio chip has to be enabled to catch incoming packets. The power consumption of the RF212 radio chip used on the Pixie platform is 19 mA when transmitting, 9 mA in the receive mode and 0.2 µA in the power-save mode. To prolong the node lifetime, the radio chip has to be operated in the power-save mode as often as possible. Low-power listening (LPL) reduces the time a node spends for idle listening by duty-cycling the radio chip [14]. A node wakes up periodically from sleep mode to check the radio channel for activity. If there is activity detected, it remains in the receive mode until the packet is received successfully, otherwise it returns back to sleep. The sender node has to transmit a preamble which is at least as long as the sampling interval of the receiving node. The sampling interval and the preamble length can be tuned to meet the desired duty-cycle. However, the per-hop delay increases with an increasing sampling interval. The TinyOS RF212 radio driver provides a basic LPL implementation.

4 Applications for Smart Buildings

As a proof of concept for our approach, we developed a web application to monitor and control the sensor network using the Google Web Toolkit.[1] The goal of the web application running on the central unit is to provide a user-friendly way to get information from the sensor nodes and to control the actuators. The back-end layer communicates with the sensor nodes over the RESTful API described in Section 2. To detect new devices plugged in the network, the central unit listens to multicast DNS messages sent out by the clients. Whenever the web application detects a new device supporting the service type "`_rest-sensor._ws._tcp`", it automatically discovers the functionality offered by the device. Furthermore, the web application allows to define a set of rules which trigger certain actions based on a specified event. A typical rule may have the form *"if the motion sensor in room B4 does not detect any movement for five minutes, turn off the light in B4 and send the PC to standby mode"*. As the functionality of the devices is discovered by the central unit, the effort required to set up monitoring applications for smart building can be significantly reduced. In order that a device can be controlled by the web application, it only has to send out multicast DNS messages and to implement the RESTful API.

In addition, sensor nodes can provide push-based access to sensor data using a simple subscription mechanism based on the RESTful API. Once the central unit or another node subscribes to a data stream, the node periodically transmits sensor readings to the subscriber using UDP packets. By employing this two-fold approach, we can combine the flexibility provided by RESTful web services with the smaller overhead of UDP.

5 Evaluation

In this section, we evaluate the performance of our implementation of web services on tiny sensor devices. We measured the response time for a HTTP request to the web service on the sensor node. During the measurement period, the central unit triggers periodic HTTP GET requests to the sensor node. Measurement results are shown in Figure 7. Requests which can be answered with a small HTTP payload length have a response time of less than 200 ms. It can be seen that the response time increases almost linearly with increasing payload length. Persistent TCP connections lead to a smaller response time with the drawback of increased memory use on the sensor node.

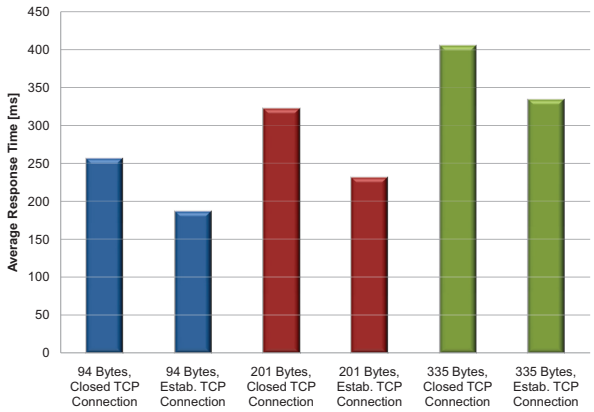

Figure 7. Average response times for HTTP requests to the sensor node for different payload sizes (with and without persistent TCP connections).

[1] `http://code.google.com/webtoolkit/`

6 Related Work

Connecting wireless sensor nodes to the Internet has been widely explored in the last few years. Sensor nodes running on proprietary standards like ZigBee need an application layer gateway to connect to other networks. As our devices need to be accessible using standard protocols, implementing TCP/IP on the sensor nodes seems to be more suited [3, 5]. The first IPv4-compatible network stack targeting 8-bit microcontrollers has been presented in [4]. However, the increasing demand for IP addresses initiated by the vision of ubiquitous sensor nodes can be better handled with IPv6. 6LoWPAN [10, 9] is a standard for implementing IPv6 on wireless sensor networks. The integration of XML based web services into embedded devices has been shown in [8] for an ARM7 microcontroller with 256 Kbytes ROM and 32 Kbytes RAM. Priyantha et al. [15] focused on an efficient implementation of web services on sensor nodes by reducing the power-consumption and minimizing the overhead introduced by the transport layer while using XML and WDSL data formats. Using sensor nodes as RESTful resources has been proposed in [7] targeting the Sun SPOT platform, which has a 32-bit processor and 512 Kbytes RAM. We use similar ideas for accessing sensor nodes using HTTP methods and coding the answer as JSON objects, but we target mote class devices with an order of magnitude less memory and more severe energy constraints. While we use a full TCP/IP stack running on our sensor nodes, a proxy server is used in [7] to bridge the requests to the devices. Approaches to use sensor nodes to reduce the power consumptions in buildings are presented in [7, 15]. For instance, the ACme project [11] uses sensor devices with an energy meter to report the current power consumption to a central server.

7 Conclusion

In this paper, we presented an approach to interconnect different sensor and actuator nodes in building control and monitoring systems. Even though we are targeting tiny sensor nodes with limited memory and processing power, we propose the use of lightweight web services based on Representational State Transfer (REST). Sensor nodes run a small web server on top of a TCP/IP stack to provide access to sensor data and actuators using HTTP requests. Data is represented in the JSON format which is a more lightweight alternative to XML. Using a simple browsing mechanism, clients can fetch a list of services offered by a device. Service discovery based on multicast DNS messages enables the system to integrate new devices without additional configuration effort. Using established and widely spread technologies which are not limited to WSNs, enables the connection of various types of devices, which is a must for smart buildings. We implemented the system using TinyOS on Pixie nodes, a new prototyping platform based around the ZigBit module. A web interface running on a central unit provides an user-friendly way to interact with the sensor nodes. Finally, we showed by measurements that the system offers an acceptable performance given the limited computing power and memory constraints of the hardware platform.

8 Acknowledgments

The authors would like to thank Miklós Maróti and Stephen Dawson-Haggerty for their support related to the RF212 radio and blip implementation. Furthermore, we would also like to thank Thomas Fahrni, Richard Huber and Mustafa Yücel for their help with the Pixie hardware.

References

[1] S. Cheshire and M. Krochma. Internet-Draft: Multicast DNS. http://tools.ietf.org/id/draft-cheshire-dnsext-multicastdns-06.txt, Aug. 2006.

[2] Communication from the European Commission. Energy Efficiency: Delivering the 20% target. (772), Nov. 2008.

[3] D. E. Culler and G. Tolle. Embedded Web Services: Making Sense out of Diverse Sensors. *Sensors*, May 2007.

[4] A. Dunkels. Full TCP/IP for 8 Bit Architectures. In *Proc. of First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys)*, San Francisco, CA, USA, May 2003.

[5] A. Dunkels and J. Vasseur. IP for Smart Objects. IPSO Alliance White Paper #1, Sept. 2008.

[6] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.

[7] D. Guinard and V. Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM)*, Madrid, Spain, Apr. 2009.

[8] J. Helander. Deeply Embedded XML Communication: Towards an Interoperable and Seamless World. In *Proc. of the 5th ACM International Conference on Embedded Software (EMSOFT)*, Jersey City, NJ, USA, 2005.

[9] J. Hui, D. Culler, and S. Chakrabarti. 6LoWPAN: Incorporating IEEE 802.15.4 into the IP Architecture. IPSO Alliance White Paper #3, Jan. 2009.

[10] J. W. Hui and D. E. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proc. of 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, Raleigh, NC, USA, 2008.

[11] X. Jiang, S. Dawson-Haggerty, P. Dutta, and D. Culler. Design and Implementation of a High-Fidelity AC Metering Network. In *Proc. of 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, San Francisco, CA, USA, Apr. 2009.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007.

[13] C. Pautasso, O. Zimmermann, and F. Leymann. Restful Web Services vs. "big" Web Services: Making the Right Architectural Decision. In *Proc. of the 17th international conference on World Wide Web (WWW)*, 2008.

[14] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proc. of the 2nd ACM Conference on Embedded Network Sensor Systems (SenSys)*, Baltimore, MD, USA, 2004.

[15] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In *Proc. of 6th ACM Conference on Embedded Network Sensor Systems (SenSys)*, Raleigh, NC, USA, 2008.