Middleware Performance Evaluation in Wireless Sensor Networks

Admilson R. L. Ribeiro, Lilian C. Freitas, Carlos R. L. Francês, João Crisóstomo W. A. Costa

***Abstract*- The use of middleware to develop distributed applications liberates the programmer of the concerns (communication and coordination among software components) imposed by the distributed network environment. Besides these concerns, in wireless sensor networks are considered, also, their specific characteristics (address, mobility, amount and limited resources of the sensor nodes). This paper describes the performance evaluation of SensorBus, a message-oriented adaptive middleware for wireless sensor networks that uses policies to assist to the several characteristics of the wireless sensor networks. Policies are implemented in an application profile, through metadata, encoded in XML documents. Performance evaluation of SensorBus is made through their two main services: message and context. We used measurement techniques to carry out the experiments with objective of analyzing the impact of two routing protocols in the middleware also to show that using metadata to incorporate policies in the middleware does not imply undue overheads.**

***Index Terms*—Metadata, Middleware, Performance Evaluation, Wireless Sensor Networks**

I. Introduction

Wireless sensor networks (WSNs) consist of a group of sensor nodes distributed in a physical area with purpose of detecting occurrences in environment. These networks are used in specific applications as environmental monitoring, tracking of vehicles, habitat monitoring, among others. Thus, the utility of a WSN bases on its capacity to provide information of large areas in response to queries made by user. Development of WSN applications needs to consider specific characteristics of these networks, when compared with traditional wireless networks [1]: (1) attribute-based naming - sensor nodes can be addressed by their own attributes or by attributes captured in the physical environment; (2) variable mobility - sensor nodes dispersed in a forest for collecting information are static while sensor nodes put in a surface of a river for collecting information about pollution are mobiles; (3) high density - due to the sensing range of covering of a sensor node to be smaller than its range of radio reach, the size of the covering area demands corresponding size in the amount of sensor nodes; and (4) limited resources of sensor nodes - energy is more limited in WSNS than in other types of wireless networks due to the nature of the sensor devices and to the difficulty in recharging their batteries in inhospitable areas. Thus, it is observed that applications are closely related to WSN design, for each application type there is need to modify the WSN design to meet certain characteristics. Therefore, WSN applications need to be adaptive, that introduces an additional burden on the application programmer.

Middleware tries to solve the adaptation problem of WSN applications. Several middleware [2]-[6] were developed for these networks. These solutions support network re-configuration in two ways: mobile code [2]-[4] and by flashing the mote's instruction memory [5]-[6]. Solutions that use memory reprogramming take a long time and consume a lot of energy transferring every code through WSN so that modifying memories of sensor nodes. Solutions by mobile agents support weak mobility, because the execution state is not transferred.

To outline the limitations of the solutions above, we developed SensorBus, a middleware for WSN applications based on principle of metadata. Through metadata, we obtain separation of concerns, that is, we distinguish what the middleware does than from how the middleware does it. Thus, it is possible to add re-configuration in the middleware through a profile coded in XML (eXtensible Markup Language) [7]. This profile contains description of policies to address to the main characteristics of WSNs. The policies are divided in application policies (for instance, sampling period), communication policies (central or distributed routing) and context policies (for instance, battery level). SensorBus behaves as a provider of customizable services, where customization happens through metadata, which encode the behavior of middleware to answer services request of applications in several contexts.

In spite of the great amount of middleware for WSN, still there is not a methodology to do performance evaluation of this kind of software. Usually, the evaluation is made to approach certain characteristic, for instance, service quality. Thus, our main contribution is the implementation of a methodology for performance evaluation of middleware based on services, where each service is analyzed in agreement with its position in the stack of OSI (Open Systems Interconnection) protocols [8].

The remainder of the paper is organized as follows. Section II presents the main components of SensorBus and describes how it was implemented. Section III describes the

Admilson R. L. Ribeiro is with the Computing and Electrical Engineering Department, Federal University of Pará, Belém, Pará, Brazil (e-mail:admilson@ufpa.br).

Lilian C. Freitas is with the Computing and Electrical Engineering Department, Federal University of Pará, Belém, Pará, Brazil (phone: +55-91-3201-7740; fax: +55-91-3201-7740; e-mail: liliancf@ufpa.br).

Carlos R. L. Francês is with the Computing and Electrical Engineering Department, Federal University of Pará, Belém, Pará, Brazil (e-mail: rfrances@ufpa.br).

João Crisóstomo W. A. Costa is with the Computing and Electrical Engineering Department, Federal University of Pará, Belém, Pará, Brazil (e-mail: jweyl@ufpa.br).

performance evaluation platform. Section IV presents the measurement methodology to carry out the experiments. Section V presents the performance evaluation results of SensorBus services. Section VI provides the concluding remarks of the paper.

II. Design and Implementation

SensorBus is a message-oriented middleware (MOM) constituted of three mechanisms: publish-subscribe paradigm, constraint language and application filters. Publish-subscribe paradigm is used to meet the attribute-based naming and also to incorporate the operation way for occurrence of events. Constraint language is used to facilitate the work of on-line programming of applications. Application filters are used to make data internal aggregation, reducing the data flow in the network, therefore decreasing the power consumption in sensor nodes.

As shown in Fig. 1, SensorBus architecture is made up of three main components: application service, message service and context service. Application service provides an API (Application Programming Interface) of high level that simplifies construction of applications. Message service is responsible for providing communication and coordination for distributed components, turning transparent, for user, these issues. Service context is responsible for managing heterogeneous sensors that gather information from the internal and external environment to sensor node. For each resource that the middleware manages, there is an adapter that interacts with a physical sensor, processing its information and thus obtaining a value demanded by application. Complete description of the SensorBus design can be found in [9].

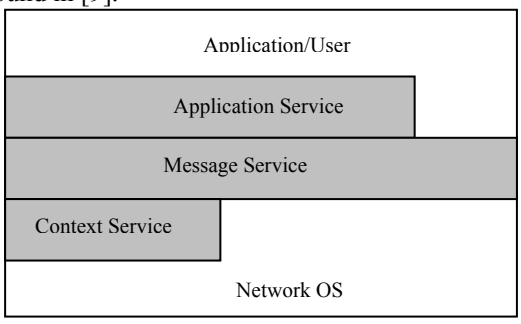

Fig. 1. SensorBus Architecture.

The current implementation of SensorBus is constituted of three software modules: a module that executes in PC (Personal Computer), another module that executes in the sensor nodes and a necessary module to interconnect the previous modules. The module that executes in PC was implemented in Java using JDK 1.4.2 while application profiles were codified in XML. XML was used because it supports a representation of information that is manipulated easily by computers and well known for people. We used also some technologies XML, in matter DOM (Document Object Model) [10] and XPath (XML Path) [11]. The use available XML parsers reduced considerably the time of development of the middleware.

In nesC language [12], we developed the module that executes in the sensor nodes. This module is responsible for query processing system to extract information from the WSN. Given determined query, specifying the interest data, this module collects the data of the sensor nodes in the environment, filters them, aggregates them and makes the routing for PC.

We implemented a proxy in Java to do the interconnection between the PCs network and the sensor network. The proxy executes in a PC that is connected to sensor network through the serial port. It is used to read data packets that arrive for the serial port and to send them through a TCP/IP (Transmission Control Protocol / Internet Protocol) port connection [8], so that programs can communicate with WSN through sink node. We used the tool MIG (Message Interface Generator) [13] to generate Java classes automatically that correspond to the types of active messages used in the components, coded in nesC, that execute in the motes. Using MIG, we outlined the difficulty of translating the message formats for use in the proxy in Java.

To verify the SensorBus usability, we implemented a query application of environmental variables. The attributes that can be collected are temperature, light, pressure and humidity. After, we decided which the policies should be encoded in the application profiles. As application policies, we established the sampling period, query type (event-driven or continuous) and yes/no allowing aggregates. In relation to communication policies, we used routing type (central or distributed) and specification of broadcast interval. Alarm about battery levels, memory, and radio signal strength were chosen as context policies.

III. Performance Evaluation Platform

The hardware platform for performance evaluation of SensorBus is constituted of equipments with Intel processors and interfaces 802.11b for wireless communication. The PC connected to the sink node consists of a laptop Dell Latitude equipped with 512 MB of RAM (Random Access Memory) and Centrino processor with 1.6 GHz executing the Red Hat Linux 9 operating system.

The sensor nodes consist of Micaz motes of Crossbow Technology, Inc equipped with processor ATMega129L [14] of 7.3728 MHz with 128 KB of program memory (FLASH), 4KB of data memory (SRAM). This platform still has 512 KB of external memory (FLASH) for reading of measurements of sensor nodes and a module of radio CC2420 [15] of 2400 MHz capable to offer a bandwidth total of 250 Kbps. An expansion slot accommodates a variety of sensing cards such as light, temperature, magnetic field, sound, and so on. The Micaz motes execute the TinyOS operating system [13] that has a programming model based on components, provided by the nesC language.

To accomplish the measurements, a WSN was used composed by 8 sensor nodes, including the sink node (connected to the station base), transmitting in the frequency of 2,048GHz and power transmission of 0 dBm. The tests were accomplished in an area of 10m x 50m.

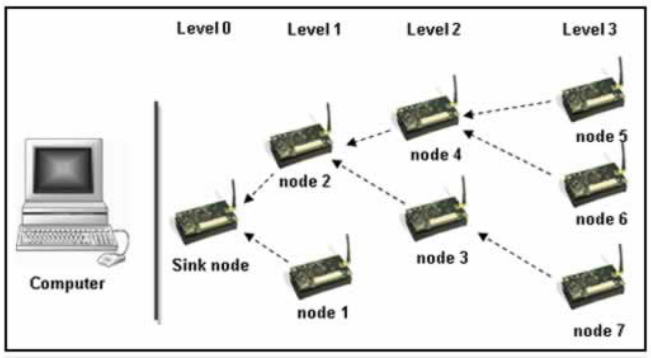

Fig. 2. WSN Configuration.

We adopted a configuration composed by 3 levels, as shown in Fig. 2. In the level 1, the sensor nodes are connected directly to the sink node. In the level 2, the sensor nodes communicate with an intermediate node, until arriving to the sink node. Already in the level 3, the sensor nodes communicate with two intermediate nodes, until arriving to their destinies. The objective of this configuration is to force the packet routing in the network.

IV. Measurement Methodology

For performance evaluation of SensorBus, we evaluated their two main services: message service and context service. Each service suffers the effect that happens in the several layers of the protocol stack that support WSNs. As shown in Fig. 3, in these networks, the essential protocol layers are the MAC protocol on data link layer and the routing protocol on the network layer. MAC protocol creates the network topology and shares the transmission medium among sensor nodes. The routing protocol allows communication via multi-hop paths. The transport protocol that implements end-to-end flow control is rarely used in WSNs. The middleware layer is equivalent to the presentation layer in the OSI model.

Fig. 3 shows that each SensorBus service is associated to certain group of layers. For instance, message service is on the level of the network layer and suffers influence of all layers that are below it. Application service is on top of stack and suffers influence of all layers while the middleware layer is responsible for policies management.

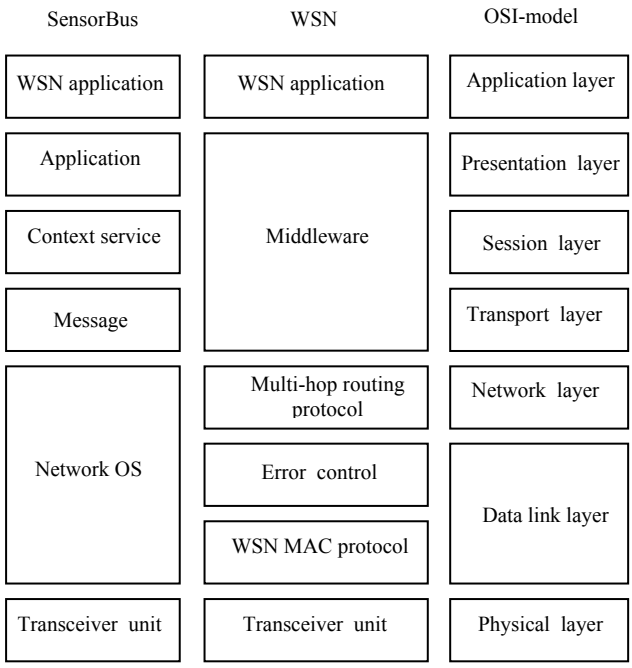

Fig. 3. OSI Model, WSN and SensorBus services.

A. Message Service

For performance evaluation of the message service we considered packet delivery in the physical layer, MAC layer and network layer. As the packet delivery in those networks is influenced strongly by the routing protocol, we analyzed the impact of two multi-hop protocols in the message service while we controlled the factors that influence the physical layer and MAC layer.

In the physical layer, the framing functions and bit error detection or correction are affected for several factors. First, environmental characteristics can cause multi-path signal reception or signal attenuation. Second, distance between sender and receiver can determine the received signal strength. Finally, small variations in the circuits of sender and receiver or in battery levels can affect the functions of the physical layer. Our experiment was accomplished in a same stable environment, without environmental interferences, so that physical layer influences were the minimum possible.

In addition to the factors that influence the physical layer, on the MAC layer the functions of access discipline to the channel and error detection are affected for two factors: the amount of messages generated by the sensor nodes and the topology (space relationship among sensor nodes). In the experiment, we maintained the same topology and workload to avoid the influence of that layer. We used the B-MAC data link protocol due its efficiency in packet delivery when compared with other data link protocols [16].

Multi-hop protocols used to analyze the impact in the message service were: LEPSM (Link Estimation and Parent Selection Method) [17] and MintRoute [18]. The LEPSM multi-hop routing algorithm is based on the mechanism of link estimate and parent selection for the multi-hop execution. LEPS algorithm is responsible for monitoring whole received traffic in the node receiving directly the updating messages of route in a single hop. These messages can be sent of the neighbors inside of unique scale of hops. Internally, LEPS algorithm manages the closer available

neighbors and it decides the destiny of the following hop based on the semantics of the minimum path. For default, LEPS algorithm sends a message of route updating to every 10 seconds and it calculates again after 50 seconds (5 messages of route updating).

Another multi-hop routing protocol was MintRoute, also called WMEMA (Window Mean with Exponentially Weighted Moving Average). This protocol is based on the technique of link estimation that calculates the average tax of success on the period of time and it adjusts through curve adjustment technique EWMA (Exponentially Weighted Moving Average) [18].

For performance evaluation of the multi-hop protocols we used the Surge application [5], an application that accompanies TinyOS. It is an application to exemplify the use of the LEPSM multi-hop protocol and it is divided in two modules: a module developed to execute in the sensor nodes and a module that executes in the station base. Surge application that executes in the sensor nodes collects information of brightness of the sensor nodes and sends to the network through the station base connected to the sink node. The module that executes in PC is a Java application that can be used to visualize the logical network topology and the sensing readings. Both modules of the Surge application were used in the evaluation of the multi-hop routing protocols; however some alterations were necessary to evaluate the MintRoute protocol. For default, Surge application uses the LEPSM protocol, so that to use the MintRoute protocol we have altered the libraries of the Surge application to configure it in agreement with that protocol.

We used the measurement technique with the following metrics: throughput, power consumption and packet delivery fraction (PDF). Surge application provided throughput and PDF. Throughput is obtained by the number of messages that incoming to a determined sensor node by time unit. In our experiment, as the Surge application processes and sends messages, the measured throughput was transmitting messages by second in each sensor node. PDF is obtained by the relationship between receiving packets and transmitting packets in each sensor node.

Power consumption was obtained through the measure of current and voltage of a sensor node, using an oscilloscope in an electronic laboratory. With the current and the measured voltage, the consumed energy by an electronic device is calculated by the equation:

$E = V \times I \times \Delta t$.

Where E represents energy in Joules; V, voltage in Volts; I, current in Amperes; and t, time in seconds.

B. Context Service

The main objective of performance evaluation of this service is to validate the thesis that SensorBus requests only a small increment in terms of elapsed time to answer a service request, compared with approach that does not use policies customization where the provided services are not configured to assist to the user's preferences and the context conditions.

In order of validating this service, we implemented a synthetic benchmark that provides as performance metric, the elapsed time. The elapsed time (in milliseconds) is measured among the instant that a service request is issued, and the instant that a policy is selected and initialized to answer the service request. We recorded the elapsed time for 20 measurements and the final result refers to the average of the measured values in each one of the 20 repetitions.

V. Performance Results

A. Message Service

1) Throughput

Fig. 4 shows average throughput obtained for LEPSM and MintRoute protocols. As it can be observed, the performance of the LEPSM protocol was superior in most of the sensor nodes of the network, considering the topology presented in the section III.

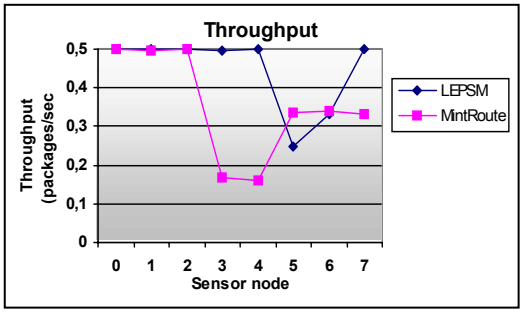

Fig. 4. Throughput for multi-hop protocols.

Doing an analysis for level, we verified that in the level 0 (where it is the sink node) all of the protocols maintained the expected performance (throughput of 0,5 packets/sec), because the sink node is connected directly to the computer, there is not the possibility of packet loss.

In the level 1, for the fact of sensor nodes be connected to the sink node (destiny node), there was not the need of any complexity in the algorithm for discovery of the next hop, in that way, both protocols presented equivalent performance.

In the levels 2 and 3, in that the packet routing demands a larger efficiency of the routing algorithms, the LEPSM protocol presented a superior performance in relation to the MintRoute protocol. Demonstrating that the criterion of discovery of the next hop through the minimum path is more efficient than the link estimation. That because in sensor nodes moved away from the station base, the quality of the link as criterion can provoke a loop in the routing of packets.

2) Packet Delivery Fraction

Fig. 5 shows the percentage of packets sent to each sensor node, obtained for the LEPSM and MintRoute protocols.

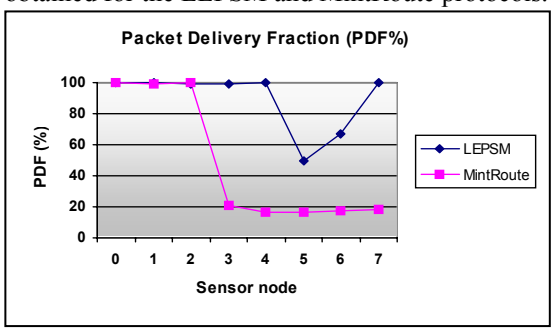

Fig. 5. PDF for the multi-hop protocols.

In agreement with the illustration, we observed again that

the performance of the analyzed protocols does not diverge when the sensor nodes are connected directly to the sink node, presenting a PDF close to 100%. Already in the levels 2 and 3, we observed the superior performance of the LEPSM protocol in relation to the MintRoute protocol. This is due the great loss of packets generated in communication loops occurred among the most distant nodes from the sink node (levels 2 and 3).

3) Power Consumption

The data presented in Fig. 6 represent the average power consumption for each sensor node. Because usually a sink node (node 0) is not powered by batteries, we do not study its power consumption.

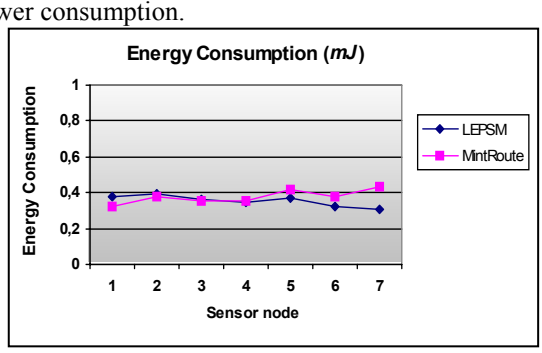

Fig. 6. Power consumption for multi-hop protocols.

In agreement with the illustration, the two protocols presented similar performance just for the nodes 2, 3 and 4, diverging for the node 1 and for the nodes of the level 3. Through a quantitative analysis, we observed that the MintRoute protocol provided a total average consumption 8% larger in relation to the provided by the LEPSM protocol. This inferior performance presented by MintRoute protocol is due to the already mentioned communication loop occurred in the levels 2 and 3, generated by criterion of hop discovery that take account the estimate of quality of the link.

B. Context Service

Fig. 7 shows the effect that metadata has on the elapsed time of a customization of context policy over a basic middleware without use of policies. The intersection of the curve with the axis Y represents the use of SensorBus in a static way (without policies); in this case the elapsed time is the same for any amount of sensor nodes.

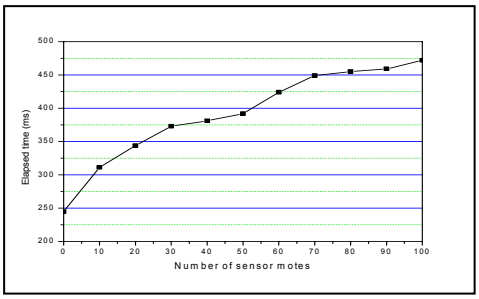

Fig. 7. Impact of Metadata.

As the figure shows, the increment in the elapsed time is more or less linear in relation to the number of sensor nodes. Even in a situation with considerable number of sensor nodes, concerning 100, the performance of SensorBus is rather good, as the elapsed time is below 0.5 second.

VI. Conclusion

In this paper, we described the performance evaluation of SensorBus, a middleware for WSNs that explores metadata to meet requirements of the applications. Performance evaluation of the message service was accomplished with objective of verifying the effect of the routing protocol in the middleware, and thus to obtain a protocol of larger efficiency under the point of view of consumption of energy. With evaluation of the context service we verified the impact of the establishment of policies in the middleware. The evaluation of those two services was accomplished through measurements obtained in a real platform of WSN.

From the presented results, we verified that, in general, the LEPSM and MintRoute protocols presented similar performance, considering the close sensor nodes to the sink node. This is due the low demanded complexity of the routing protocol in these situations. However, evaluating the behavior of the protocols in the moved away sensor nodes from the sink node, we verified that the MintRoute protocol presented an inferior performance in relation to the LEPSM protocol, demonstrating that the algorithm based on the criterion of link quality, used by the MintRoute protocol, is not efficient when the destiny node is moved away from the origin node, provoking in many cases loops in the routing of the packets, as exemplified through the presented results.

Thus, we demonstrated that in certain areas of location of WSN, the LEPSM protocol is more efficient to be used in the middleware and that the use of policies to accomplish the middleware re-configuration does not implicate in significant additional cost in performance.

References

[1] Rentala, P., Musunuri, R., Gandham, S. and Saxena, U. “Survey on Sensor Networks”, University of Texas, Dept. of Computer Science, 2002.

[2] Smart Messages project. http://discolab.rutgers.edu/sm, 2003.

[3] Boulis, A., Han, C.-C., and Srivastava, M. “Design and implementation of framework for efficient and programmable sensor networks”, in Proc. of MobiSys, 2003.

[4] Liu, T. and Martonosi, M. “Impala: A middleware system for managing autonomic, parallel sensor systems.” In ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP´03), June 2003.

[5] Crossbow Technology, Inc., XNP – X Network Programming. http://www.tinyos.net/tinyos-1.x/.

[6] Hui, J. and Culler, D. “The dynamic behavior of a data dissemination protocol for network programming at scale”, in Proceedings of the 2nd International conference on Embedded networked sensor systems. ACM Press, 2004.

[7] Bray, T., Paoli, J., and Sperberg-McQueen, C. M. Extensible Markup Language. Recommendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium, 1998.

[8] Stallings, W. Data & Computer Communications, Prentice-Hall, Englewood Cliffs, NJ, USA, 6th edition, 2001.

[9] Ribeiro, A. R. L., Freitas, L. C., Silva, F. C. S., Francês, C. R. and Costa, J. C. W. “SensorBus: A Middleware Model for Wireless Sensor Networks”, in Proceedings of the 3rd IFIP/ACM Latin America Networking Conference, ACM Press, October 2005.

[10] Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A. L., Nicol, G., Robie, J., Sutor, R., Wilson, C., and Wood, L. Document Object Model (DOM) Level 1 Specification. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium, 1998.

[11] Clark, J. and DeRose, S. XML Path Language (XPath). Technical Report http://www.w3.org/TR/xpath, World Wide Web Consortium, 1999.

[12] Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D. "The nesC Language: A Holistic Approach to Networked Embedded Systems" in Proc. Programming Language Design and Implementation, Jun. 2003

[13] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. E., Pister, K. S. J. "System Architecture directions for networked sensor", in Proc 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.

[14] Atmel inc., http://www.atmel.com/products/avr/

[15] Chipcon inc., http://www.chipcon.com/

[16] Polastre, J. and Culler, D. "B-mac: An adaptive csma layer for low-power operation", UC Berkeley, Tech. Rep cs294-f03/bmac, December 2003.

[17] Hohlt, B. A. "The Design and Evaluation of Network Power Scheduling for Sensor Networks". A dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science. University of California, Berkeley. Spring., 2005

[18] Woo, A., T. Tong, and Culler D. "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks", Proc. ACM SENSYS, Los Angeles, CA. November, 2003.