Experimental Evaluation of Energy Balance Algorithms in the SenseWALL Sensor Network Test-bed

Constantinos Marios Angelopoulos
Computer Technology Institute (CTI)
and University of Patras, Greece
Email: aggeloko@ceid.upatras.gr

Dionysios Efstathiou
Computer Technology Institute (CTI)
and University of Patras, Greece
Email: eustathi@ceid.upatras.gr

Sotiris Nikoletseas
Computer Technology Institute (CTI)
and University of Patras, Greece
Email: nikole@cti.gr

***Abstract*—We present key aspects (hardware, software, topology, networking) of *SenseWall*, an experimental sensor network test-bed we have created for the implementation and engineering of distributed sensor network algorithms. We then describe how SenseWall has been in particular used to implement two recent state of the art algorithms for energy balanced sensor data propagation. We elaborate on the issues and challenges created by the restrictions and particularities of the experimental test-bed and how we dealt with them. We also carry out a detailed performance evaluation comparing the energy balance protocols to two baseline protocols that include only either single hop or direct data transmissions.**

I. Introduction

We study the problem of how to achieve energy-balanced data propagation in distributed wireless sensor networks ([1], [2], [4], [5]). The energy balance property guarantees that the average energy spent per sensor is the same for all sensors in the network at any time during the network operation. This property is crucial for prolonging the network lifetime by avoiding early energy depletion of sensors and the non-utilization of available energy on sensors. It is particularly relevant to smart, green buildings of the Future Internet where wireless sensors are used to propagate data related to current ambient conditions (light, temperature, human presence, etc); by balancing the energy dissipation among the sensors in the deployed network its smooth operation is prolonged and the provision of building optimization services is further facilitated.

In particular the SenseWall experimental sensor network test-bed has been created in the context of the EU-FIRE R&D Project HOBNET. The Hobnet project (June 2010-May 2013)[3] is a project that uses existing FIRE technologies and platforms for Future Internet applications focused on automation and energy efficiency for smart/green buildings. The main research areas of Hobnet are the use of IPv6/6LoWPAN infrastructure, 6lowApp standardization, new algorithmic models that fit the objectives of the project and are scalable and energy efficient, rapid development and integration for building management and support for deployment and monitoring of the applications on existing FIRE testbeds.

In this study we implement and experimentally evaluate in our testbed two energy balancing protocols and compare them to two pure data propagation schemes; multi-hop routing and routing with direct transmissions only.

II. The Routing Protocols

A. The Distance-based P_i Protocol

Lifespan maximization can be achieved by keeping energy balance. The energy balance property guarantees that the average per sensor energy dissipation is the same for all sensors in the network. This property is important since it prolongs the lifetime of the network by avoiding early energy depletion of sensors. To overcome an unbalanced energy consumption scheme, the authors in [2] propose a mixed routing strategy where in each step the algorithm decides probabilistically and locally whether to propagate data one-hop towards the *Sink*, or to send it directly to the *Sink*. This randomized choice balances the one-hop transmissions with the direct transmissions to the *Sink*. Thus, a trade-off emerges between cheap but slow multi-hop propagation and direct but energy consuming transmissions that "bypass" the sensors lying close to the *Sink*, thus propagating data fast. Note that, in most protocols, motes close to the *Sink* tend to be overused and die out early resulting in a disconnected network. In [2], via detailed analysis, the probabilities for each propagation choice are estimated in order to guarantee energy balance.

Authors describe the protocol in such a way that all needed estimations can easily be performed by current technology sensors using simple to obtain information. However, in order to bring the protocol down to real sensors, we need to revise some of the assumptions originally made in [2] for analysis purpose. We also need to cope with the restrictions imposed by the programming language (NesC [6]), the operating system (TinyOS [7]) and the sensor motes themselves (TelosB). These restrictions apart from traditional limitations of WSNs (i.e. limited energy, low computational power) also include, lack of floating point numbers, limited program memory and no support for dynamic memory allocation.

B. The Energy-based E_i Protocol

In [5] the authors revisit the family of mixed strategy routing schemes and propose an on-line distributed algorithm for lifespan maximization. In each step, the algorithm decides to propagate data either one-hop towards the *Sink*, or to send it directly to the *Sink*. This decision is based on explicit information about the energy spent by the current node and the energy spent by the nodes that are one-hop closer to the *Sink* than itself.

Let n be a sensor. V_n is the neighborhood of node n. Node n knows the energy spent by each node in its neighborhood. Let m be the sensor of neighborhood V_n with the lowest energy spent. When n holds data it makes the following decision:

- If node n has spent more energy than m, then n sends the message to m (spending one energy unit).
- Otherwise, n sends the message directly to the $Sink$ spending d^2 energy units, where d is the distance from n to the $Sink$.

C. The Pure Multi-hop Protocol

In this protocol every node propagates data to the sink in a multi-hop way. In particular, let n be a sensor and i the sector it belongs to. The node n can forward its own generated data and relay data from previous sectors $(i+1, i+2, \cdots)$ by forwarding messages to a random node of the next sector $i-1$ towards the sink. This protocol is cheap but its latency can be high since it creates a bottleneck region around the *Sink*.

D. The Pure Direct transmission Protocol

In contrast to the previous pure multi-hop protocol, in this direct transmission protocol every node of the network sends the data that generates directly to the $Sink$ with maximum transmission power. This protocol propagates data very fast but its energy dissipation is very high.

III. Hardware Architecture and Networking

The hardware components of our experimental test-bed include a set of 28 TelosB motes (see Fig. 1) connected to a control Base Station PC via a USB tree which is consisting by USB cables and hubs, for controlling the network and collecting experimental data from the motes without interfering with the wireless communications, thus leaving the wireless medium free for the routing algorithms.

Fig. 1: The 802.15.4 TelosB Mote.

We deployed the motes in a sector-shaped topology as shown in Fig. 2 in order to approach the theoretical model of the algorithms described in [2] and [5].

The motes of the test-bed are connected to the Base Station desktop PC via USB cables (see Fig. 3) and each mote is also supplied with two AA rechargeable batteries (1.2V - 2600mAh). The TelosB motes have integrated an USB interface, enabling us to control easily the whole network (mass flushing/programming of the nodes, reseting the nodes, etc.) and receive packet-statistics through the wired USB backbone.

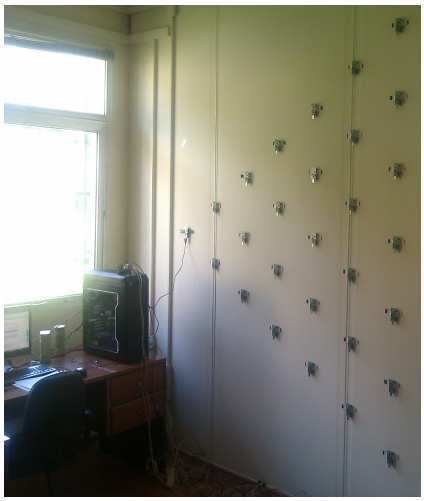

Fig. 2: The node topology.

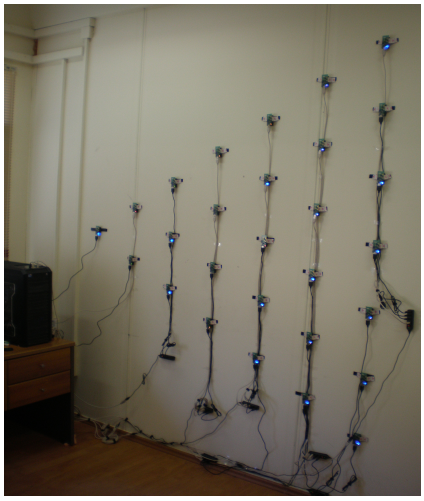

Fig. 3: The USB-wired node topology.

Desktop PC runs the MySQL server and the MoteProgrammer Java application (Fig. 4). The $Sink$ sends the received messages to the PC using the serial UART interface and then the Java application stores them in a MySQL database. The MySQL server is used to log the data from the experiment while the MoteProgrammer application enables the user to control the nodes by flushing, resetting, changing the event generation rate, changing the sampling rate, etc. and to perform off-line analysis of the collected data by making queries for the various performance evaluation metrics.

The TelosB mote uses the IEEE 802.15.4 compliant RF

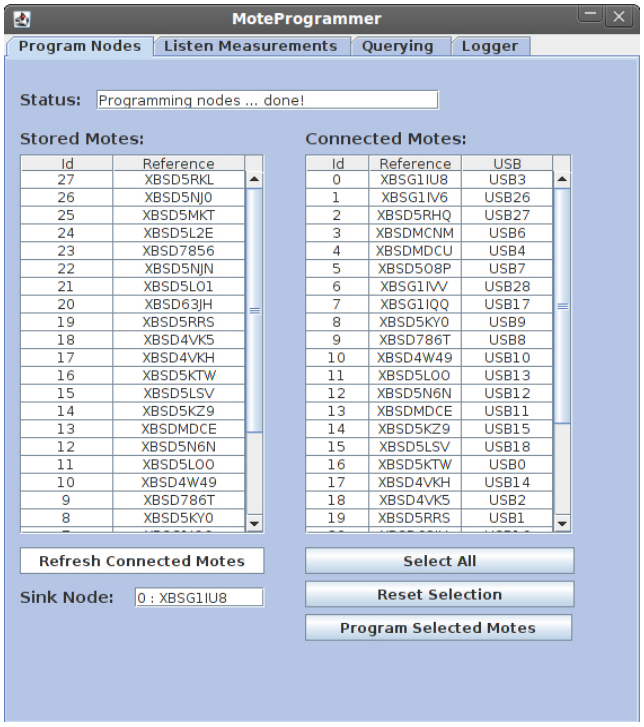

Fig. 4: User interface for controlling the test-bed and processing experimental data.

transceiver developed by Chipcom (CC2420). We assign unique 16-bit addresses to the TelosB motes ranging from 1 to 27 and we assign 0 to the $Sink$.

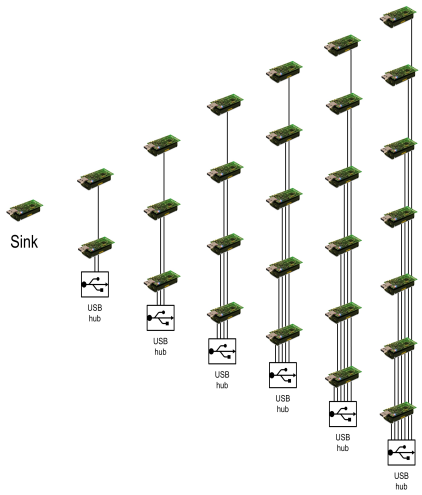

Fig. 5: The node topology.

IV. Software Architecture and Algorithms

The general overview of the architecture is depicted in Fig. 6.

A. P_i Algorithm Implementation

We implement the energy balancing protocol described in [2] in TinyOS 2.1.0. The module consists of three files. A header file where all necessary structs, message types and macro-variables are defined; a configuration file where the wiring among the components used is defined; and a source file with the source code of the module we implement. In the header file we define the ID of the root of the network as $BASESTATION\_ADDR$ and the event generation rate as $DEFAULT\_INTERVAL$.

We also define $pimsg\_t$, which is the message type that we will be using in this implementation. Each message, apart from sensory data, will also contain information about the source of the event (i.e. the ID of the mote that initially generated the event) and the current number of hops the message has made during its propagation so far. Also, we define the array $neighbours[]$. This array holds the ID's of the motes of the next sector towards the *Sink*, which consists of all the motes that are one hop closer to the *Sink* than the current mote. Finally, every mote maintains locally a counter $msg\_counter$ that counts the number of events it has generated. This counter is used in the calculation of success rate.

In the configuration file we declare that we will use *DemoSensorC* for generating events, as if we were using real sensors. The reason we made this option is that we are focusing on data propagation and not on monitoring. Furthermore, as the sensors are deployed indoors, all readings would have been expected to be similar. We will also be using the *RandomMlcgC* component as random generator. This component is a fast implementation of the Park-Miller Minimal Standard Generator for pseudo-random numbers and it provides an interface that receives a seed for initialization. This way, assuming that we provide a purely random seed, we will not take the same sequences of generated numbers each time we run an experiment. As seed we choose to use the readings from the *VoltageC* component, that is the current voltage that the mote is supplied with. We consider this value to contain adequate randomness, as it depends on many factors (i.e. how long the batteries have been unused). Every time a mote decides to transmit directly to the *Sink*, it has to set the transmission power to the maximum as it may lie several hops away. To do so we use the *CC2420PacketC* component that provides the *setPower* interface.

At the beginning every mote in the network performs all the necessary initializations, that is initializing local variables, tables and structures as well as preparing the radio tranceiver and all relevant hardware. It also initializes the random generator using as seed the energy of the mote. As said, we consider this value to contain adequate randomness, as it depends on many factors (i.e. how long the batteries have been unused). Also, every mote, except for the *Sink*, commences the event generation procedure. Events are generated locally in a periodic manner; therefore the corresponding procedure is controlled by a timer. Every time the timer is fired the mote increases by one the variables indicating the number of events generated so far. Based on which sector the mote belongs to, it decides with probability P_i to transmit the data directly to the *Sink*, or with probability $1 - P_i$ to propagate its data to the next sector. The formula that provides P_i is rigorously

computed in [2] to be equal to

$$P_i = 1 - \frac{1}{(i+1)(i-1)}$$

where i the number of hops from the current sector to the *Sink*. Using the component *RandomMlcgC*, the mote generates a random 16-bit number. Then, the mote transmits directly to the *Sink* or propagates to the next sector based on whether the following control is true or not (correspondingly):

```
if ( DiceResult % inverse-Pi  == 0 )
```

where *DiceResult* is the random generated number, % the modulo operator and *inverse-Pi*$= 1/P_i * 10000$. The latter is due to the fact that TelosB motes use the MSP430 microcontroller that does not have a hardware floating point unit. Thus, all floating point operations are expensive in terms of computational complexity and code space, as they should be implemented in software. So, we use fixed point operations; we multiply with a big constant (always the same) before any calculation. If the mote decides to propagate the data to the next sector, then another random number is generated. Then, based on that number, a mote from the next sector is chosen (array *neighbours[]*) and data are sent to it.

This procedure is also followed when the mote receives data from another sector. It probabilistically decides whether it is going to transmit them directly to the *Sink* or if it is going to propagate them to the next sector.

B. E_i Algorithm Implementation

In order to implement this protocol in TinyOS for TelosB motes, we follow the same architecture as in the previous section. We use the same components and interfaces for starting up and handling the necessary hardware modules (i.e. radio, LEDs, etc), as well as generating random numbers and handling the transmission power. However, in this case we use an extra type of message, $NRGmsg_t$. Each mote uses this type of messages to periodically collect the remaining energy of the nodes of the next sector towards the *Sink* that holds in an array $neighbhours - nrg[]$. The frequency at which every node updates $neighbhours - nrg[]$ is much smaller than the event generation rate in order not to induce significant overhead to radio communication. Finally, every time a mote has data to route (either data that generated itself, either data that the mote relays), it compares its own remaining energy to the highest value of the array $neighbhours - nrg[]$. If the remaining energy of the mote is higher, then it transmits directly to the *Sink*. Otherwise, if there is a mote to the next sector that has more remaining energy (has spent less energy), then the mote propagates the data to that mote.

C. The MeasurementsLogger Component

We implement a new component in TinyOS 2.1.0 as a binding interface between the desktop software control application and the routing protocol we evaluate on SenseWALL. The role of this component is to setup the parameters of the experiments (i.e. event generation rate, energy sampling rate, experiment duration, etc.) and to monitor the evolution of the routing protocol by enabling the logging of the performance measurements described in V-B.

The module consists of four files. One containing the signature of the interface provided by MeasurementsLogger component, a header file where all necessary structs, message types and macro-variables are defined; a configuration file where the wiring among the components used is defined; and a source file with the source code of the module we implement. The header file can be automatically created by the Java application and contains all the macrovariables that correspond to the parameters of the experiment.

D. Software to control SenseWALL

The Java application that runs in the Base Station automatically detects the motes that are connected to the desktop PC via USB ports and allows the administrator of the network to interact with the motes e.g. the application allows the user to flush (program) the motes with the nesC application binary code, to reset the motes and communicate with them in order to change the event generation rate, the energy measurement sampling rate, the experiment duration and store the experimental measurements into a MySQL database. Also, the Desktop application provides an interface for computing the following performance evaluation metrics: a) **data delivery latency** b) **average energy consumption** and c) **success ratio**.

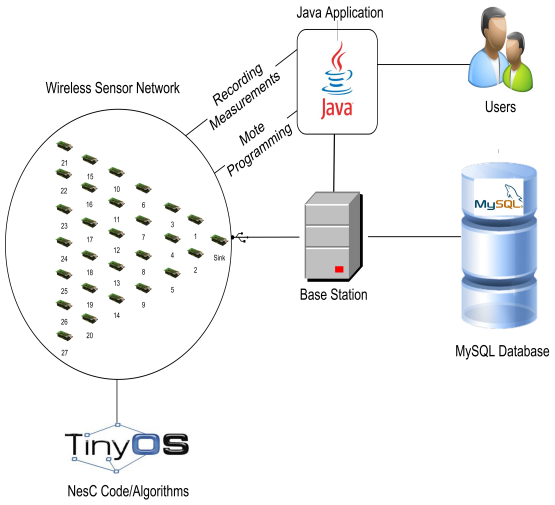

Fig. 6: Software Component Architecture.

V. Performance Evaluation

A. Experimental Setup

Each node generates a total of 1.000 events/messages with a rate of 0.2 events/second. Every 50 events (or 250 seconds) a node sends its current energy to the $Sink$. Before conducting any experiment we first fully recharge the batteries in order to have a common reference for all experiments.

B. Metrics Computation

We compute the above mentioned metrics as following: a) **data delivery latency** which is the average number of hops needed to reach the $Sink$. Each generated message has a $hop\_counter$ field. Each time a node relays a message, increases the $hop\_counter$ field before forwarding the message. In order to calculate the total data delivery latency of the node, we add the $hop\_counter$ of all received messages.

b) **average energy consumption** is the average energy spent per node during the network operation (we measure the energy consumption per node by subtracting the battery level of the node at the end of the experiment from the initial battery level of the node before the start of the experiment). In order to measure the energy consumption of the motes, we run our experiments by deactivating the USB power supply and the motes use only the battery supply. The motes periodically sample the current battery level using the Msp430InternalVoltage component of TinyOS. At the end of the experiment, we enable the USB supply and via the Java application we ask from the motes that participated to the experiment to send their energy measurements to the Base Station which stores all the collected data to a MySQL database.

c) **success ratio** is the ratio of the total number of packets received by the sink to the number of packets generated by the whole network. Each node assigns to a new event/message that generates an incremental packet sequence number $msg\_counter$. Upon reception on $Sink$ of a node's generated message, we divide the total number of received messages from that node by the maximum packet sequence number ($msg\_counter$) of that node.

C. Experimental Results

Figures 7 and 8 depict the average data delivery latency per sector and in total for all four protocols. In general (except for the jumps only protocol), we note that as the distance between the generated event and the *Sink* is increasing, the time the message requires in order to be delivered is growing proportionally. Particularly, the multi-hop propagation scheme is severely affected by the distance, in terms of latency. In contrast, when directly transmitting to the *Sink* distance does not affect the time delivery of messages. These findings are also depicted in Figure 8 where the performance of the E_i and P_i protocols lies between the two extremes of multi-hop propagation and direct transmissions, so the energy balance protocols indeed achieve a good latency-cost trade-off. This is due to the hybrid nature of these two protocols. Note that due to the different way E_i and P_i decide whether to make a jump or to propagate to the next sector, there is a significant difference in energy consumption. As E_i is more prone to direct transmissions, it also has shorter data delivery times.

Figures 9 and 10 depict the findings about energy consumption. As expected, when using direct transmissions in order to deliver data to the *Sink*, motes lying in distant sectors consume much more energy; even 240% more energy than motes close to the *Sink*. On the other hand, when using multi-hop propagation, motes closer to the *Sink* consume nearly

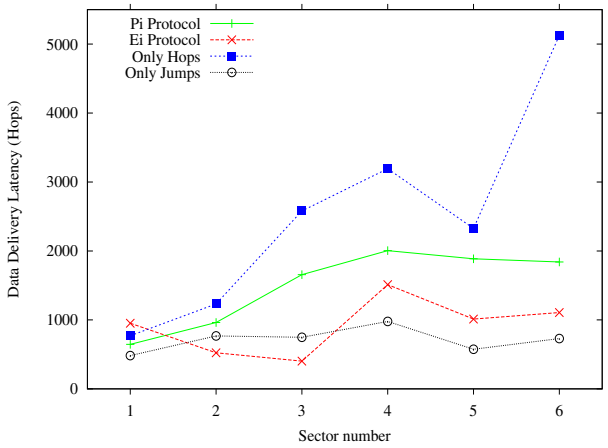

Fig. 7: Average data delivery latency per sector.

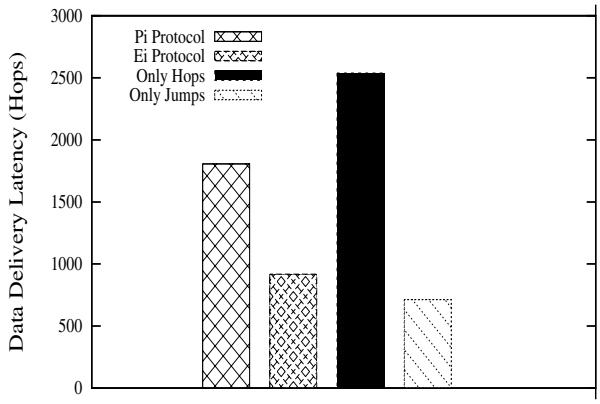

Fig. 8: Total average data delivery latency.

double the energy compare to the rest of the network. This is due to the bottleneck effect these motes come up with; they serve as relays to the entire network flow. The P_i protocol actually manages to balance the dissipated energy across the network as all sectors, more or less, dissipate the same amount of energy. In total, again the multi-hop propagation scheme and direct transmission represent the two extremes with the other two protocols lying between them towards satisfactory performance trade-offs. Note that the E_i protocol consumes slightly more energy than the P_i protocol. This is due to the fact that E_i is more prone to direct transmissions than the P_i protocol.

Figures 11 and 12 depict the success rate with which each protocol delivers data to the *Sink*. In a per sector basis, we note that all protocols follow the same pattern. The variation of the results are due to the big number of motes deployed in a very confined space (27 motes in a $5 \times 3m^2$ area). However, in total all protocols perform more or less the same in terms of success rate. We however note that in these measurements we report only the success rate while the network remains connected; if we could let the network evolve for more time, then the two protocols without energy balance would earlier result to network discontinuities thus their success rate would

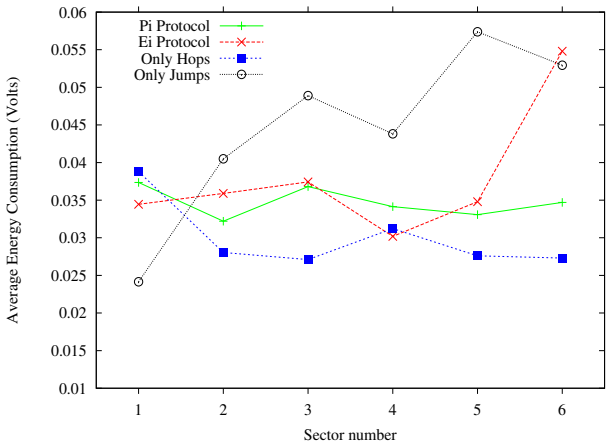

Fig. 9: Average energy consumption per sector.

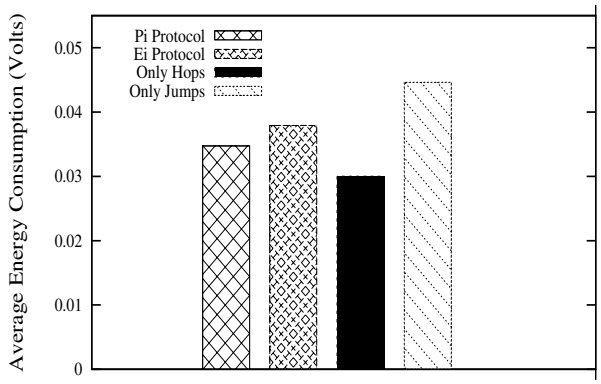

Fig. 10: Total average energy consumption.

be much less than that of the two energy balance protocols.

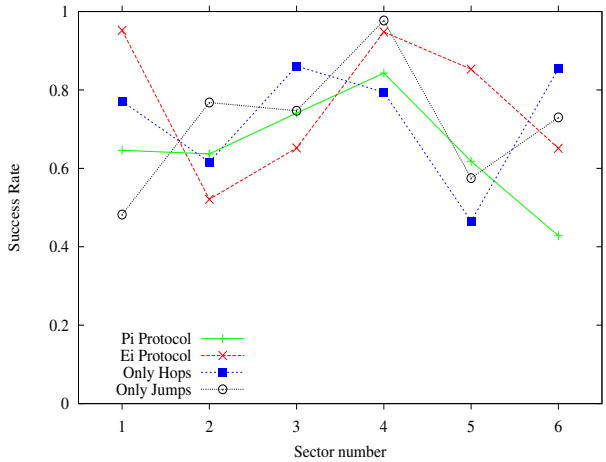

Fig. 11: Success ratio per sector.

VI. Conclusions and Future Work

In this paper we present the experimental performance evaluation of four routing algorithms in our sensor network test-bed. These algorithms include two hybrid (combining single-hop and long transmissions) energy-balancing schemes that we compare to two pure data propagation strategies with either multi-hop or direct transmissions only. We also present the hardware and software architecture of our experimental test-bed and discuss how we dealt with the issues arising when implementing high level algorithms in a real, constrained environment. The detailed experiments we conducted validate the theoretical results on the protocols' performance; indeed, the energy balance protocols more or less equi-distribute the energy dissipation in the network, while achieving good energy-latency trade-offs.

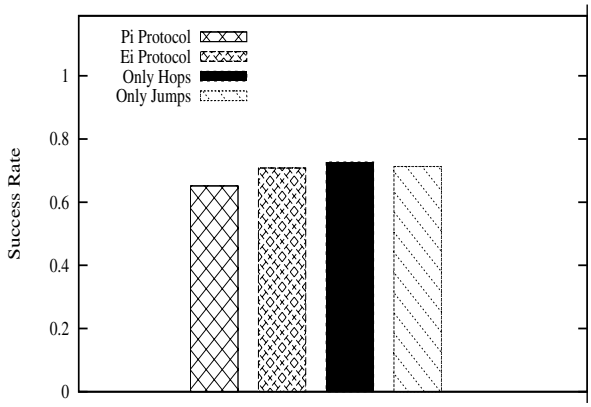

Fig. 12: Total success ratio.

We plan to extend the existing test-bed in two ways, by increasing the total number of nodes and by deploying them in a larger area of interest (potentially occupying several rooms in a building) for testing the performance of the studied algorithms in a larger (network size and spatial) scale. Also, to study the impact of more complicated terrains (like outdoors or indoors with obstacles such as walls, elevators, etc). We intent to implement a wide variety of algorithms (e.g. obstacle avoidance, data collection in mobile networks, other probabilistic algorithms, clustering, etc.) resulting in a critical mass of protocols that will consist a concrete WSN protocol suite. We also plan to enable the monitoring, control and experimentation with the test-bed remotely by using a web-based interface.

References

[1] D. Efstathiou, S. Nikoletseas, and C. Raptopoulos. Near-Optimal Energy Balanced Data Propagation via Limited, Local Network Density Information. *HOBNET Technical Report*, 2011.

[2] C. Efthymiou, S. Nikoletseas, and J. Rolim. Energy balanced data propagation in wireless sensor networks. *Wireless Networks*, 12(6):691-707, November 2006.

[3] Holistic Platform Design for Smart Buildings of the Future Internet. http://www.hobnet-project.eu/

[4] A. Jarry, P. Leone, S. Nikoletseas, and J. Rolim. Optimal Data Gathering Paths and Energy Balance Mechanisms in Wireless Networks. *DCOSS*, pp.288 305, 2010.

[5] A. Jarry, P. Leone, O. Powell, and J. Rolim. An Optimal Data Propagation Algorithm for Maximizing the Lifespan of Sensor Networks. *DCOSS*, Volume 4026/2006, 405-421, 2006.

[6] nesC: A Programming Language for Deeply Networked Systems. http://nescc.sourceforge.net/

[7] TinyOS 2.x Documentation. http://www.tinyos.net/tinyos-2.x/doc/