This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2010 proceedings

User-Centric Plug-and-Play Functionality for IPv6-enabled Wireless Sensor Networks

Paulo A. C. S. Neves[1,2,3], Binod Vaidya[1], and Joel J. P. C. Rodrigues[1,2]
[1]*Instituto de Telecomunicações*, Portugal
[2]Department of Informatics, University of Beira Interior, Covilhã, Portugal
[3]Superior School of Technology, Polytechnic Institute of Castelo Branco, Portugal
pneves@co.it.pt, bnvaidya@co.it.pt, joeljr@ieee.org

***Abstract*— Smart nodes that sense the environment and communicate wirelessly to reach a sink node create wireless sensor networks. One of the main research challenges regarding wireless sensor networks is user deployment, namely in terms of configuration and management. On non-commercial solutions the user typically must be aware of the underlying technology to obtain sensing services. Internet connectivity is also desirable, so future deployments must take into consideration this feature, enabling realistic ubiquitous computing. This paper presents a user-centric solution for IPv6-enabled wireless sensor networks, using the Contiki operating system and Crossbow TelosB motes, featuring a Plug-and-Play like experience. One of the motes provides sink node capability to the network, through USB connection with a personal computer, which sends and receives data, presenting it to the user. A dedicated serial protocol for USB communication with the sink was developed and extensively debugged, featuring sink querying and network configuration. The current testbed uses User Datagram Protocol over IPv6, with 6LoWPAN, and IEEE 802.15.4 wireless communication between the sensor network motes and the sink device. A Plug-and-Play like operation is achieved through zero-user configuration, since the user only needs to plug in the sink and give power to the remote motes.**

I. INTRODUCTION

Wireless sensor networks (WSNs) application began in military applications, namely on enemy detection and targeting [1]. Due to the growing interest and hardware availability such networks began to spread over to several civil applications, such as environmental monitoring, smart spaces, habitat monitoring, animal tracking, and healthcare, among others [2].

WSNs have yet failed to find the killer application for general public adoption. Such *status quo* is mainly due to lack of auto-configuration tools and corresponding APIs, which should allow Plug-and-Play like operation of WSNs – just plug in and use. Although faced as almost a ideological issue, with research on both sides of the "barricade", we believe that IPv6 is the needed technology to integrate WSNs into ubiquitous computing, providing a complete sensing system.

The connection of WSNs to the Internet is as desirable as needed [3]. Two main approaches exist to achieve Internet connectivity: proxy-based approach and smart sensor node IPv6 stack, namely with the implementation of 6LoWPAN [4]. The first approach uses dedicated routing protocols for WSNs, performing protocol conversion in the sink node. The second is to use IPv6 inside the sensor network, through a suitable TCP/IP stack.

Two major operating systems lead the way on firmware development for motes: ContikiOS [5] and TinyOS [6]. ContikiOS features an implementation of IPv6 over IEEE 802.15.4 [7], using the 6LoWPAN specification. The TCP/IPv6 sensor stack in Contiki, named uIP6, features TCP and UDP over IPv6, among other features, granting the IPv6 ready silver seal from the IPSO – IP for Smart Objects. We chose ContikiOS over TinyOS for the maturity of its IPv6 stack when compared to blip (Berkley IP) implementation over TinyOS and the more C-like programming of Contiki versus the nesC used by TinyOS.

This paper presents an approach for monitoring and configuration for WSNs without user intervention. Using the Universal Serial Bus (USB) communication we create a sink station with a Crossbow TelosB mote and UDP/IPv6 wireless communication (uIP6) with other TelosB motes on the network. The testbed remote nodes send sensing values to the sink, which may respond with configuration commands.

The remainder of the paper is as follows. Section II provides background information on related technologies. Section III presents the current approach architecture, featuring Plug-and-Play like operation, while on section IV implementation details are exposed. Section V shows testbed validation, and section VI concludes the paper, adding guidelines for future work.

II. BACKGROUND

This section encompasses some background technical contents, namely IPv6 over WSNs and the operating system that enabled us to develop the node's firmware, the ContikiOS.

IPv6 for WSNs was considered for some years a myth. As a result several routing and transport protocols were developed for WSNs [8, 9]. However, a recent wave of research studies prove that IP and WSNs can work well together [10]. Specifically through the 6LoWPAN specification, where "transmission of IPv6 packets over IEEE 802.15.4 networks" is described, adoption on both research and off-the-shelf solutions has risen.

Contiki is an operating system for embedded smart objects, namely WSN nodes. The current version, 2.3, features IPv6 routing on the TelosB platform, extended hardware support and uIPv6 sensor stack. The operation system (OS) features an event driven kernel, thread-like implementation through "protothreads", a threading model that allows threads, but using the same stack, hence resulting in a memory-efficient approach. Events can be both system and programmer-defined, and event timers provide support for periodic events.

ContikiOS is based on the C language, using the specific C

978-1-4244-6404-3/10/$26.00 ©2010 IEEE

compiler for the target platform. Due to its build system applications recompile to different targets only require a simple change to the "make" command, application code is kept outside OS directories.

One of the biggest features of Contiki is uIP6 [7], a smart sensor node stack featuring TCP/UDP and IPv6, namely through *sicslowpan* (6LoWPAN implementation for Contiki). This implementation was awarded the IPv6 ready silver seal from the IPv6 Ready Logo Program.

This paper refers to Plug-and-Play like experience as the lack of user intervention on the network, namely with the introduction of new and different nodes. The user just needs to plug-in the sink mote through USB to start monitoring and, if desired, issue configuration commands.

WSN can benefit from PnP capabilities in several applications. Imagine the simple scenario of a body sensor network. Some patients may need specific biosensors, while others do not. If PnP is not available, every time new sensors must be attached to the network, configuration procedures are needed. Moreover, on dynamic and heterogeneous networks, where several different mote types can be used, can also benefit from PnP.

III. System Architecture

This section describes the architecture of the testbed. Our testbed currently consists of 10 Crossbow TelosB motes, one sink and up to 9 wirelessly (CC2420 IEEE 802.15.4 compliant radios) connected remote nodes, as Fig. 1 depicts. The TelosB features 10KB of RAM, 48KB or Flash ROM, three LEDs (blue, red and green), one user-programmable press button and a reset button (for program reboot, not flash reset). This platform also features integrated USB connection.

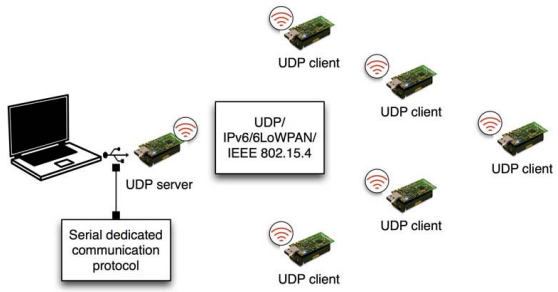

Fig. 1. Testbed architecture.

The sink connects to the personal computer that is monitoring the network, presenting several sensor and mote data according to the current TelosB architecture: ambient temperature, normal light, ultraviolet light and humidity sensors. Once the sink mote is connected, the commands may flow without user intervention. Two types of serial commands were defined, one that relates to the sink, querying sink's data and other for connected motes sensor data.

The sensed values are transmitted to the UDP server (sink) that automatically assigns a numerical node ID, associating with the received transmitter mote IPv6 address. This is useful for sensor network description, instead of using the full IPv6 address to identify a node on the personal computer. Moreover, to send a command to the sink about a given node, much less characters are sent.

This system architecture is simple enough for users, without the need for configuration. Just plug in the sink, power the motes and data acquisition and communication begins. As a result it can be very suited to locally deployed WSNs, namely the case with body wide sensor networks (BSNs) [11].

A. Serial Protocol – Monitoring

We defined and implemented a dedicated serial protocol between the personal computer and the mote. The protocol defines the commands that the sink node may accept and send respective possible replies. Two main contexts were defined: sink commands (commands begin with a *BS*, as in Base Station), and smart node commands (commands begin with a *SN*, as in Smart Node).

Command format starts with the context (*BS* or *SN*). For the *BS* context the command itself follows the context, since the sink is directly attached to the computer and does not requires identification. However for the smart nodes (*SN*) the node ID follows the context. The node ID is then bounded by a special character for easier separation of fields, the "#" character.

Several commands are common to both the sink and the smart sensor nodes. Commands such as "ip" that returns the given node's IPv6 address, "sens" that return the current sensor readings, and "txp" that returns the programmed transmission power. An example of the command needed for base station sensor values is "BSsens", while the same command for node ID 5 is "SN5#sens".

Two commands are exclusively for the sink node, namely "mn" that returns the number of nodes in the network including the base station and "list" that returns the current node IDs present in the network. Finally the "ident" command features node identification of a given remote node, by flashing the red LED for 10 seconds, at 2Hz. The full list of implemented commands is presented in Table I.

TABLE I. SINK QUERY COMMANDS DESCRIPTION.

Serial Command		Purpose
Context	Command	
BS/SN	ip	Return the node's IPv6 address
BS/SN	sens	Retrieve last sensor's readings, separated by "|"
BS/SN	txp	Retrieve currently programmed IEEE 802.15.4 radio transmission power
BS/SN	power	Retrieve current node power consumption
BS/SN	reset	Clear data structures. BS – all structures, SN – specific node's structures. Returns OK.
SN	ident	Command used to "identify" a remote node by blinking the red LED for 10 seconds at 2Hz. Returns "OK" if node exists.
BS	list	List the current network mote's ID, separated by "."
BS	mn	Return the current number of motes present in the network

In terms of command replies, the sink can reply with the command result or four specific errors: "NAC", "CER", "INE" and "NDA" (Fig. 2). "NAC" reply is used when a command is not recognized (Not A Command), "CER" stands for *Command Error* and may be sent by the sink when the main

command is recognized, but the remaining of the command is not recognized. "INE" stands for Inexistent Node Error, and is sent by the sink when a command is issued for a non-existing node ID. Finally "NDA" (No Data Available) can be replied when a command has no data, for instance when no remotes nodes have been identified on the "BSlist" command.

B. Serial Protocol – Configuration

The proposed serial protocol also takes into account configuration. Two commands can be issued to a specific network node (SN context) or to all nodes (BS context): "sample" and "txtime". The "sample" command allows the reprogramming of the time period between sensor readings, while the "txtime" allows the reprogramming of the time between UDP communications. As a result sample time must be equal or less than "txtime". The reply from the sink on this commands are just an "OK" if the command succeeded and a "NOK" if the command found a problem, for instance a configuration value out of bounds.

These types of commands are "complex commands", in the sense that they require further parsing. As an example, the command "BStxtime3" changes all remote nodes transmission period to 3 seconds.

To send commands to the network remote smart sensing nodes, the sink uses the communication of the sensor readings from a given node. It then alters the behavior and sends a reply to the originating node with configuration data.

C. UDP Communication

UDP communication allows the remote smart sensor nodes to communicate with the sink. A pre-defined message consists of the current values of sensor readings, separated by a pipe symbol "|". The server IPv6 address and UDP port is hardcoded in the client firmware, while IPv6 local-link addresses are managed by uIP6.

As above-mentioned, the server sends configuration commands to the remote smart sensor using UDP communication. UDP is lighter than TCP in terms of protocol overhead and energy consumption. To take advantage of the approach the server only replies to the client when a configuration message must be sent, or the mote cannot be added to the current sink.

The server may need to send configuration commands or just reply that no space is available for storing the mote's data. The server takes advantage of communication initiated by the client to reply accordingly. If no space is available an "NDA" is replied. If one or the two controlling timers (sampling and transmission) must be reset, the server replies with a "TS" – Timer Set command with two numbers separated by "|". The first one resets the sampling timer, while the second resets the transmission timer. Both are expressed in seconds.

Due to the inherent data cache mechanism implemented on the sink node, and limited available RAM memory, each sink can monitor up to 10 remote nodes. When a 11^{th} node tries to be attached to the same sink, the sink replies with a "NSA" (No Storage Available) – Fig. 3 a).

IV. SINK AND NODE FIRMWARE

This monitoring and configuration tool relies mainly on firmware on the wireless sensor network smart nodes. Two firmware implementations, the server and the client were developed. Since the client does not establishes USB communication we used the TelosB LEDs to acknowledge operation. The green LED is used for status information, while the blue LED is used for UDP communication. The red LED, on the remote clients, is used to identify a given node, by blinking continuously for 5 seconds.

The sensor implements a UDP server, a serial communication service through USB and also monitors its own sensors. The remote smart sensor nodes implement a UDP client and monitor its own sensor values.

Both firmware implementations use Contiki Protothreads. A first protothread is auto-started, the initialization protothread, which allows initialization of data structures, initialization of sensors, and USB communication on the sink. This protothread then starts the other protothreads.

A. Data Structures

Both sink and smart sensor node have data structures to store information. A structure was defined to store, for each node, the node's ID, IPv6 address, sensor status, and sensor values. The sampling and transmission timer periods are also included, together with other data such as power consumption. We designated this structure as *mote_data*.

The *mote_data* structure also includes a status element that codes the current node's status. The status can have five different values at the server side and four different values in the client. If status equals 1 it codes the fact that the data structure exists, but no data is available, while 2 represents the state that data is available.

In the server side values 3, 5 and 6 codify the need to send reply to the client, issuing configuration commands. Value 3 corresponds to timer programming, through the "TS" reply, while the value 5 sends a request for identification (through the blink of the client's remote red LED). Finally, 6 code the status of both timer configuration and identification.

The client adds the values 3 and 4 to the status value, coding the timers that must be set: value 3 corresponds to the transmission timer, while 4 corresponds to the sampling timer.

We opted for dynamic memory allocation and linked list structure on the sink for remote node's data storage. On the sensor node the structure is the same, with the exception that since a single node is monitored, no linked list is used.

B. Sink Firmware

On the sink, three main protothreads were defined (working in infinite loop scenarios): *udp_server_process*, *read_sensors_process* and *serial_command_parser_process*. The first one is responsible for the UDP server creation and maintenance. It creates an UDP server on ports 3000/3001, continuously accepting packets from clients. The *read_sensors_process protothread* is common to both sink and smart node firmware, and is responsible for periodically retrieving local sensor data. Finally, *serial_command_parser_process* is unique to the sink node,

processing serial communication, verifying and taking actions accordingly.

The most complex protothread is presented in Fig. 2, corresponding to the parsing and reply over USB connection. This protothread runs on infinite loop waiting for serial data (commands). Upon reception the context is identified. This allows the distinction between sink-destined and remote smart node destined commands. If the context is the smart node, the node ID must be identified. The search over the pool (linked list of mote data), dictates if the node ID exist or not. If it does not exist an "INE" string is replied. Although not shown in the figure, the "NDA" reply is used when the corresponding node has no data available, in the "Process command and reply" processing.

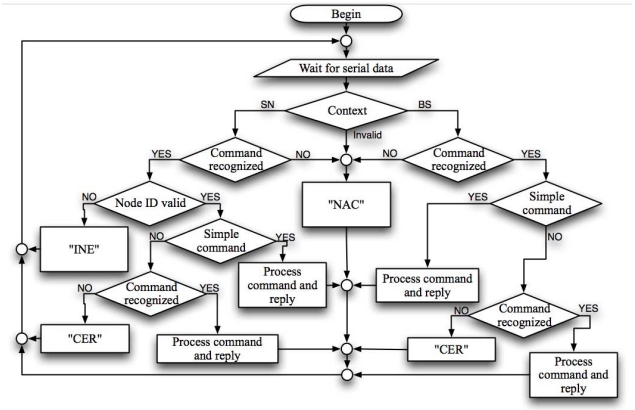

Fig. 2. Flowchart of the *serial_command_parser_process* protothread.

If a command is complex, further processing is needed. The command string is tested for match with one of the defined complex commands: "sample" and "txtime". If a match occurs, the command is processed. If not recognized, a "CER" is replied. With this protothread not only all possible commands are tested, but all issued commands, valid or not, force a response from the sink. As a result a no reply from the sink may only occur if the sink is disconnected from the computer, being it phisically or logically.

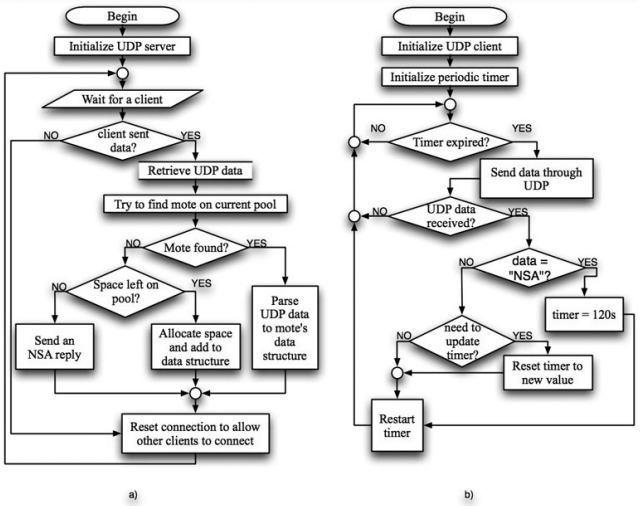

Fig. 3. Flowchart of a) *udp_server_process* and b) *udp_server_client*.

Fig. 3 a) presents the flowchart for the *udp_server_process* protothread. The server waits for an incoming client packet. If data is received, it processes data, according to a predefined format, similar to the one used in the sensor values output, with "|" separated values. If the server acknowledges that a command must be issued to the client, namely informing of no space available in the pool of clients, or timing values configuration, it sends back an UDP message to the client. Finally, the server allows other clients to communicate.

On Fig. 4 the common *sensors_read_process* is presented. This protothread is common to both node types (sink and remote smart nodes), where a timer, which can also be set by the server, controls the sampling frequency. This process also signals that data is available through the status variable. This is the simplest protothread of our current implementation.

C. Smart Node Firmware

Fig. 3 b) depicts the *udp_client_process* protothread, responsible for client communication. The client initializes the UDP client and the periodic timer that controls communication period.

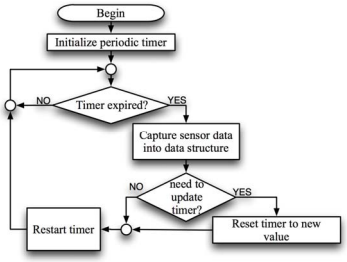

Fig. 4. Flowchart of the *sensors_read_process* protothread.

The communication period is, upon startup and by default, 4 seconds. If the timer expires data must be sent to the server. If the server replies with data, the client must analyze it. On "NDA" reply the client waits 2 minutes before attempting another communication with the server. If the reply is TS and two numbers separated by a "|" symbol, the mote compares the actual timer values with the locally stored ones. If different they are updated in the respective protothreads.

As stated before, the protothread *sensors_read_process* is very identical to the server corresponding protothread shown on Fig. 4, continuously gathering sensor data.

V. TESTS AND VALIDATION

Tests were mainly performed on the testbed, with standalone tests, mainly over the USB serial connection with the sink. LEDs and serial communication were extensively used. Fig. 5 present the testbed with 6 TelosB motes where the proposal was extensively tested and validated.

One of the main concerns was with memory, namely RAM (Random Access Memory). The current implementation can hold 10 mote's data on the most demanding RAM implementation – the server. Several motes were tested, all commands validated and the testbed with 5 motes and the sink was put into its paces for a full day. No mote suffered any crash and all of them worked properly, which confirms the current implementation as solid and robust.

The testbed shown in Fig. 5 features a laptop for network

monitoring, where the sink is connected. At early stages of development and mainly when testing a new functionality the MSPSIM emulator/simulator was used. The server needs 41'544 bytes of program memory, while the client needs 39'278 bytes. As above-mentioned, only the sink is connected to the computer, while all other five motes are connected wirelessly. The blue led is toggled every time a mote performs wireless communication, thus indicating activity. The mote identification feature may be useful to "find the mote" inside the network.

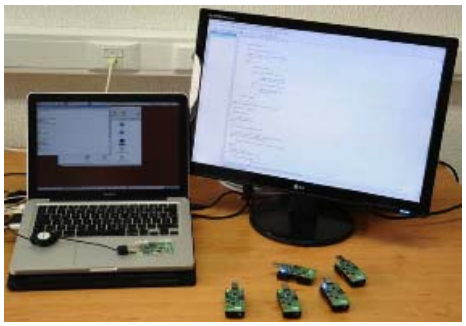

Fig. 5. Picture of the testbed running with 5 remote nodes

Fig. 6 presents a detail of serial communication results through the Contiki-provided *serialdump-linux* program. Part a) presents the issued commands to get sink sensor data, and all remote nodes IPv6 address. The "NAC" in the 8th row is due to the non-existent context "Sn". Part b) presents sensor-reading values.

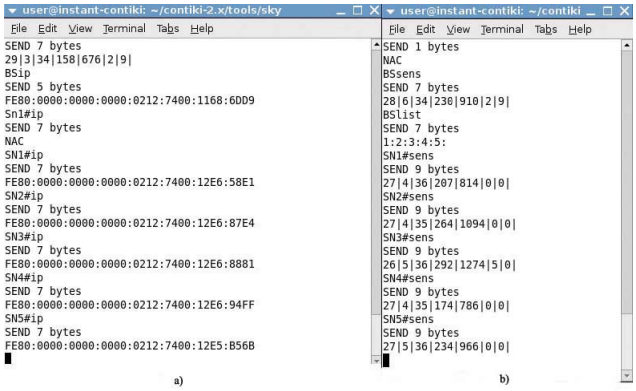

Fig. 6. Communication with sink via USB: a) IPv6 address, b) sensor reading.

VI. CONCLUSIONS

This paper proposed a monitoring and configuration tool for IPv6-enabled wireless sensor network (WSN), featuring Plug-and-Play functionality principle. Using the USB connection the WSN provides sensing services to the user, without need for configuration. The testbed can be used on a BSN-based scenario, where single-hop communications are typically employed.

This proposal is flexible in terms of connected hardware. The sensors monitored are not fixed at all. Although the implementation uses TelosB motes, the meaning of the values provided can be absolutely configurable by the monitoring software.

As future work concerns, and due to the platform similarities, we consider the replacement of the sink with a Bluetooth-enabled mote, such as Shimmer. This will allow the use of wireless for connection not only to computers, but also mobile devices. Another improvement is the ability to store history data on the external flash, namely through the Coffee file system.

ACKNOWLEDGMENTS

Part of this work has been supported by the *Instituto de Telecomunicações*, Next Generation Networks and Applications (NetGNA), Portugal, and by the Euro-NF Network of Excellence from the Seventh Framework Programme of EU.

REFERENCES

[1] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards", *Computer Communications*, ISSN: 0140-3664, vol. 30, pp. 1655-1695, No. 7, 2007.

[2] I. Khemapech, I. Duncan, and A. Miller, "A Survey of Wireless Sensor Networks Technology", in *6th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, Liverpool, UK, June 27-28, 2005.

[3] J. A. Stankovic, "When Sensor and Actuator Networks Cover the World", *ETRI Journal*, vol. 30, pp. 627-633, No. 5, 2008.

[4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944, IETF, 2007.

[5] A. Dunkels, "Operating Systems for Wireless Embedded Devices", in *The Wiley Encyclopedia of Computer Science and Engineering*, W. Sons, Ed.: Wiley & Sons, 2009, pp. 2039-2045.

[6] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gray, J. Hill, M. Welsh, E. brewer, and D. Culler, "TinyOS: An Operating System for Wireless Sensor Networks", in *Ambient intelligence*, Springer-Verlag, Ed., 2004.

[7] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making Sensor Networks IPv6 Ready", in *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, Raleigh, North Carolina, USA, 2008.

[8] K. Akkaya and M. Younis, "A Survey on Routing Protocols for Wireless Sensor Networks", *Elsevier Ad Hoc Network Journal*, Vol. 3/3, pp. 325-349, 2005.

[9] C. Wang, K. Sohraby, B. Li, M. Daneshmand, and Y. Hu, "A Survey of Transport Protocols for Wireless Sensor Networks", *IEEE Network*, vol. 20, pp. 34-40, No. 3, May/June 2006.

[10] J. Rodrigues and P. Neves, "A Survey on IP-based Wireless Sensor Networks Solutions", *International Journal of Communication Systems*, Wiley-Blackwell, 2010 (to appear).

[11] O. Pereira, P. Neves, and J. Rodrigues, "Mobile Solution for Three-tier Biofeedback Data Acquisition and Processing", in *IEEE Global Communications Conference (IEEE GLOBECOM 2008),* New Orleans, LA, USA, November 30 - December 4, 2008, pp. 1-5.