Remote Monitoring and Control of Wireless Sensor Networks

Răzvan Tătăroiu*, Dan Tudose**

** Computer Science Department, Polithenica University of Bucharest, Romania*
(e-mail: razvan.tataroiu@cs.pub.ro)
*** Computer Science Department, Polithenica University of Bucharest, Romania*
(e-mail: dan.tudose@cs.pub.ro)

Abstract: Wireless sensor networks (WSNs) are employed in environmental monitoring, vehicle tracking, building management, body monitoring and other applications. Power sources for network nodes are often limited, which imposes restrictions on hardware resources and their use by the embedded software. In developing more efficient software, it is useful to obtain performance metrics from all network nodes in a centralized manner, as well as upgrading the embedded software or configuring its parameters. It is also useful for a WSN user or administrator to obtain sensor readings and network health and performance metrics, and to control actuators or change software behavior remotely. Remote access to the WSN is especially important when nodes are physically inaccessible, mobile, or spread over a large geographical area. We have developed a method for monitoring and controlling WSN nodes from a graphical interface over an Internet connection, using the successful MonALISA framework. Data is gathered from the network and stored in an Internet-based repository, from where it can be read remotely using a graphical client program. A control service is accessed through the same client and routes commands and control data to the nodes. The interface between the WSN and the Internet services contains an abstraction layer, allowing uniform access to nodes built using various technologies and running different software and protocols.

1. INTRODUCTION

Wireless Sensor Networks (WSNs) or more generally Wireless Sensor and Actuator Networks (WS&ANs) are employed in a multitude of data acquisition, data processing, and control applications (Callaway 2004). Their advantages over traditional wired sensor and actuator networks include node mobility, increased reliability (due to the possibility of adaptive multi-hop data routing), easier installation and lower deployment cost. There are situations where wired data acquisition networks are impractical, such as environmental monitoring over large areas, or „intelligent" wearable devices forming so-called Body Sensor Networks.

A typical WSN node is capable of reading sensor information, controlling external actuators, processing data and communicating over a radio channel.

WSN nodes have specific hardware characteristics and limitations. Most WSN nodes have limited available energy: some rely on batteries and some employ environmental energy harvesting techniques such as solar panels, wind- or vibration-powered generators or thermoelectric generators (Rahimi 2003). Therefore WSN nodes tend to be small embedded systems with few processing resources and low bit rate, low range radio links. Cost and size restrictions impose similar constraints.

Fig. 1 depicts the main components of a WSN node. The lightly-coloured ones are optional.

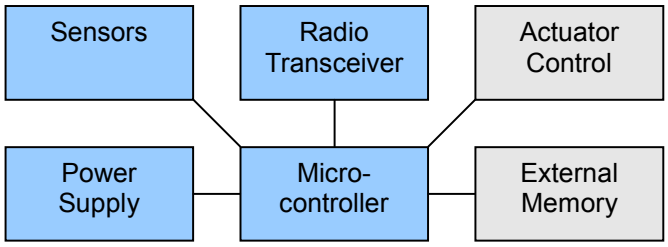

Fig. 1. WSN node structure

Hardware limitations give rise to specific software aspects. Software running on WSN nodes must be power-aware. Ideally the microcontroller spends most of the time in very low-power sleep states, and the radio transmits data in small bursts. Radio chips used on WSN nodes draw significant current when transmitting data, but also when listening for data, sometimes even more than when transmitting (Texas Instruments 2007), therefore special low-duty-cycle communication protocols and algorithms are ideally employed, such that the radio is completely off most of the time. Although all the nodes in a WSN can share a radio channel, due to the low transmitter power only nodes in close proximity can communicate. From a software standpoint this is both beneficial (no interference between nodes spaced far apart) and problematic (the need for multi-hop data routing arises).

Traditional wired sensor networks usually employ one or more master nodes with generous hardware resources and energy available, such as a PC-type computer, therefore data

processing usually happens on the master nodes. Wireless sensor nodes on the other hand are more autonomous because of the limited communication capacity between sensor nodes and master nodes. In some cases WSNs are purely peer-to-peer networks, lacking master nodes altogether. WSNs can perform data processing and aggregation inside the network, reducing the need to centralize large amounts of data. Given the fact that processing data on the nodes and forward the results between them can be energetically cheaper and more reliable than sending all the raw sensor data to a central node, advanced WSNs function as distributed processing systems.

From a user's standpoint, a WSN must provide a number of services, such as reporting events of environmental pollution, reporting the formation of traffic jams in a city, identifying a person's urgent health problem, managing the air conditioning and lighting in a building, etcetera. There is no need for the end-user to receive real-time data from all sensors in the network, but only information that is relevant. Service-oriented WSNs use this approach - the nodes run software services that read and process large amounts of sensor data, as well as user commands, and send small amounts of relevant data back to the user, thus utilizing the radio channel efficiently. Service frameworks such as the Tiny Task Network (Titan) define tasks (services) that have a number of input and output pipes (Lombriser 2007). The tasks are assigned by a scheduler to the available network nodes according to the node location, available sensors and actuators, node processor load and radio link quality, and the tasks' pipes are linked according to a service graph. More tasks can run on the same node, exchanging data through local pipes, or pipes can be connected between nodes. Tasks could also be moved between nodes in order to increase efficiency or in the event of a node or link failure. This functionality is transparent to the tasks.

2. WSN MONITORING REQUIREMENTS

From a software developer's standpoint, obtaining the lowest possible power consumption and the highest data link reliability is of significant importance. The engineer needs to be able to monitor the performance of the system when developing and testing software for WSN nodes. Indicators such as processor and memory load, wakeup frequency, duration of high-power states, amount of data sent and received over the radio, are important in developing and optimizing WSN software. These parameters need to be known for all network nodes. Higher-level, service-specific parameters, as well as lower-level parameters such as radio link quality, are also important for the scheduling algorithms in service-oriented networks and for self-healing or adaptive routing algorithms. Raw sensor data also needs to be monitored when debugging data processing software or when configuring the network after installation. Also, some simple applications only require obtaining periodic sensor readings on a central computer. Network administrators and users may also be interested in monitoring the health of the WSN nodes (such as the remaining battery charge), sensor data or performance metrics.

WSNs whose nodes are in inaccessible locations or spread over a large area clearly require software provisions to allow remote monitoring, without the need for physical access to the nodes. Even when WSNs occupy a small area and their nodes are easily accessible, connecting a dedicated debugging interface to the nodes can be cumbersome and expensive when the nodes are in large numbers.

When using a WSN or when developing software, the user must also be able to control the network. Setting application-specific software parameters, controlling actuators directly, enabling and disabling services, upgrading the software running on the nodes are examples where control of the WSN is necessary.

WSN nodes are built by multiple vendors and may vary in size, power consumption, microprocessor architecture or sensor interfaces (*Fig. 2*). Many real-time operating systems and network protocol stacks can run on WSN nodes, such as TinyOS (Hill 2000), Contiki (Dunkels 2004), Sensinode NanoStack, etc.

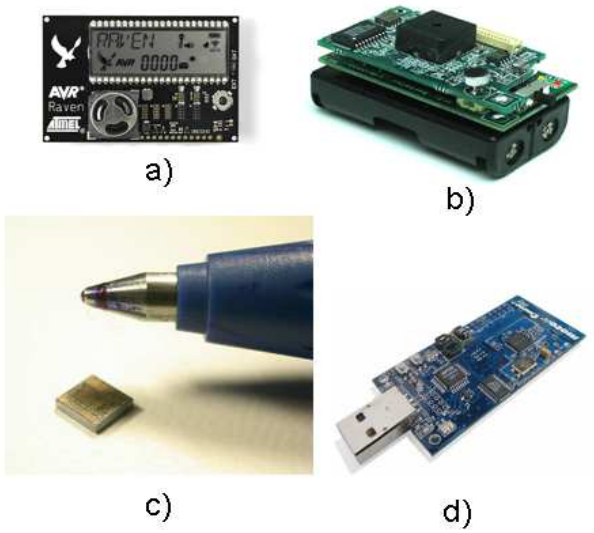

Fig. 2. Different WSN nodes: a) Atmel AVR RAVEN, b) Berkeley Mica mote, c) Berkeley Spec mote, d) Tmote Sky

It is desirable to monitor and control WSNs built using different technologies through a unified interface. In creating and managing heterogeneous WSNs it is useful to collect data from their homogeneous subnetworks and send data and commands under a common interface.

3. SYSTEM ARCHITECTURE

We developed a method for monitoring and controlling a variety of WSNs remotely over the Internet, based on the successful MonALISA framework (Monitoring Agents using a Large Integrated Services Architecture) (Newman 2003, Legrand 2004). Our method uses an abstraction layer to provide remote monitoring and control to essentially any kind of WSN.

MonALISA is a joint development of CERN, Caltech and UPB, typically used in monitoring large-scale systems such as computer clusters. It can be used to monitor and control any kind of system, including WSNs, as long as the appropriate interfacing software is available.

In short, MonALISA employs repositories to which data can be sent remotely using a portable software module named ApMon (Application Monitor) and to which users can connect with graphical client programs to view the data remotely. The client software can also access control services that run next to the data repositories using a secure, authenticated protocol. The connections can be established over the Internet, allowing user access to the WSN from any location, or over a local area network. Because multiple ApMon instances can run independently from the data repository, a WSN composed of multiple islands, in different locations, can be transparently managed as one single entity.

WSNs are typically accessed through one or more devices generically called gateways or routers. These are typically connected to large computers such as PCs, along with specific drivers. Together with the driver, the gateway allows user programs to access the WSN through standard Internet protocols such as IPv6, or through special interface programs.

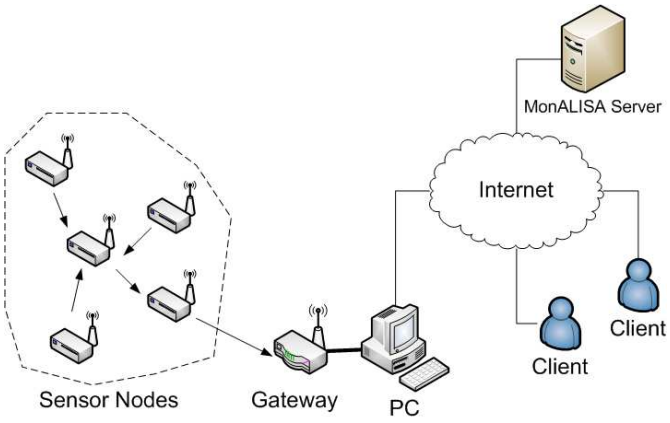

Fig. 3. WSN monitoring and control architecture

A monitoring service („monalisa-wsn") runs on the computer with the WSN gateway, calling WSN-specific programs that report incoming data. These programs call the WSN drivers and perform WSN-specific data formatting, while presenting a unified interface to the monitoring service, effectively forming an abstraction layer. The monitoring service then uses ApMon to upload the data into a MonALISA repository running on the same computer or on a remote server. Users can then connect to the repository through a graphical client program and retrieve the data and analyze it. Using asymmetric key authentication, the user can connect to a MonALISA WSN control module that runs next to the data repository. A control service („monalisa-wsn-ctl") runs next to the WSN gateway and connects to the MonALISA control module, enabling the user to send data and commands to the WSN nodes. *Fig. 3* shows a schematic diagram of the WSN monitoring and control system.

Data in MonALISA is organized as parameter-value pairs pertaining to a „host" or „node". Hosts are grouped into clusters, which are grouped into farms (*Fig. 5*). This stems from its main usage as a grid monitoring framework. Farms, clusters, hosts and parameters are identified by their name and presented to the user in a hierarchical interface. The application that uploads data is free to define any host name, parameter name or parameter value. A convenient way of using MonALISA to monitor WSNs is to present a WSN as a host-type entity with a list of parameters. The names of the parameters include a part which identifies the WSN node in case of node-specific parameters. For example, if monitoring temperature sensor readings from a WSN containing 4 nodes, the parameters may be named „temperature1" to „temperature4". The user can filter parameters by name in order to concentrate on data of immediate concern.

The parameters are sampled at defined intervals by the monalisa-wsn program by polling the WSN, or they can be reported by the WSN services automatically. In any case, users can view parameter values in near-real-time, as well as their history. *Fig. 4* shows an example of monitoring a sensor network composed of two islands located in two different locations. Each island is connected to a PC via a gateway device and each PC is running WSN-specific drivers and polling adapters. One island is using Sensinode NanoSensor hardware, which is readily capable of measuring temperature and light, and the other is using Atmel Raven hardware which only measures temperature readily. Numerous other types of sensors can be added to both platforms.

Fig. 5 shows the MonALISA graphical client main window, with our WSN selected and its monitored parameters listed. Parameters numbered starting with 1 are the Raven subnetwork, and parameters numbered starting from 5 are the Sensinode subnetwork. *Fig. 6* shows the recent history, up to the current values, for some of the monitored parameters.

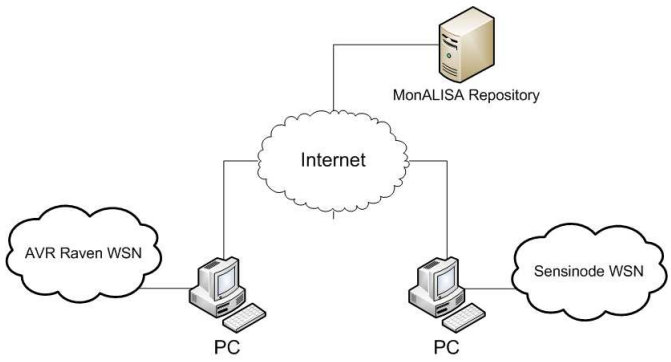

Fig. 4. Monitoring WSN islands installed in different geographical locations

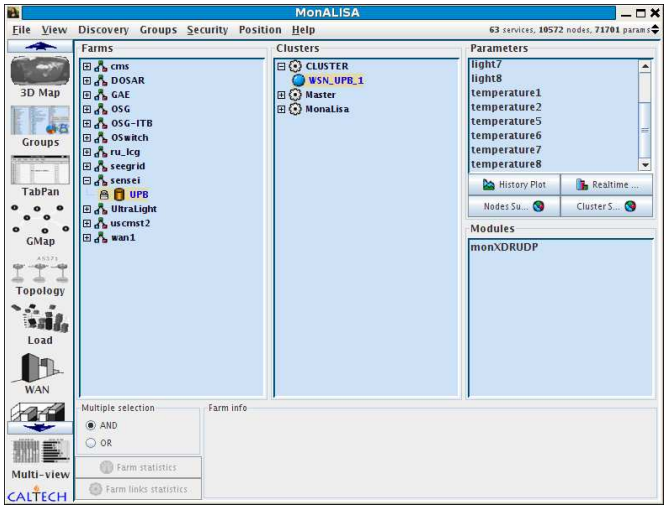

Fig. 5. MonALISA graphical user interface showing monitored WSN parameters.

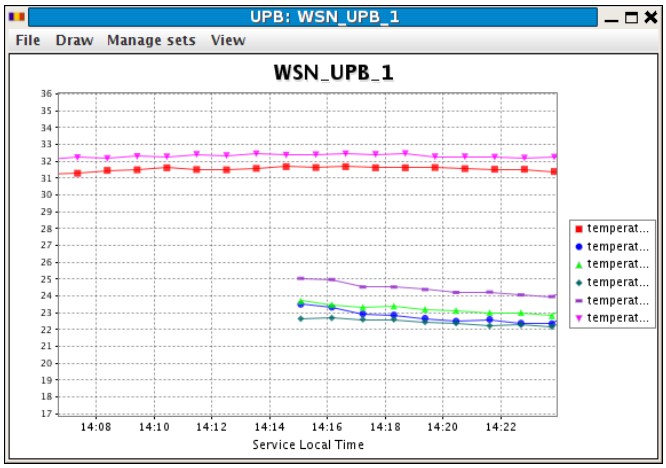

Fig. 6. Data aggregation example from geographically distinct areas monitored by two WSN islands.

Although all WSN nodes can run an infinite variety of custom-designed real-time operating systems and embedded applications, a readily-available solution is often preferred. For instance, the Sensinode NanoSensors can run the Sensinode NanoStack, which is a 6LoWPAN implementation (IPv6 over low-power wireless personal area networks). For debugging purposes and simple applications, it can also run a simpler version that uses MAC addressing instead of IPv6. The Atmel Raven comes pre-programmed with a ZigBee-based network stack that uses 16-bit node IDs for addressing. The nodes have to explicitly associate with a coordinator (gateway), unlike Sensinode NanoSensors which can be detected by broadcast queries. The Raven can be programmed with the Contiki operating system, which implements 6LoWPAN, allowing IP addressing.

Above the network protocol, each case uses a different application protocol for polling data from the sensors. NanoSensors use two command-line programs called „nPing" and „SSI-Browser" to detect sensor nodes and obtain sensor values respectively. The Raven nodes running the default software use a graphical program that connects to a „wireless services" back-end. The protocols used are known. Contiki uses a web interface accessible directly from a web browser over IPv6, but can also be configured to use a protocol with lower overhead.

Each case needs an adaptation program that is called from the main monitoring service and returns parameter-value pairs in a consistent, WSN-independent format. This abstraction layer has been implemented and tested for the technologies listed above. A version for the Titan framework can be developed, which would allow monitoring performance parameters and custom service data. *Fig. 7* details the software architecture of our solution.

The adaptation program can choose to be executed periodically by monalisa-wsn and return a data point each time, or to provide data points on the standard output periodically. The first option is used when polling the WSN and the second is used when the WSN itself pushes the data. When polling the WSN, each node from a list is queried for certain data such as sensor readings, performance metrics or debugging information. The nodes are identified by their address, which can have a wide variety of formats, depending on the hardware and software used. A technology-specific detection program, „detect", is used to build that list, which monalisa-wsn then uses transparently. The list can also be built dynamically when nodes announce their presence to the gateway, as is the case with the Raven platform. In this case the corresponding „detect" runs permanently and updates the list when needed.

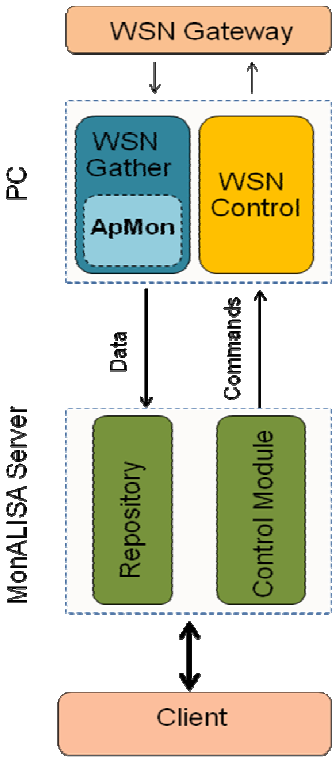

Fig. 7. Detailed software architecture

The monalisa-wsn-control program and the MonALISA control module provide remote control capabilities to the WSN. A WSN-specific adaptation program is also needed. User access to the control module is restricted using public key authentication.

A generic control interface is provided, where global WSN parameters, as well as node-specific parameters can be set. Their values can be typically read back through the monitoring interface. The parameter names, their data types and acceptable ranges are specified in an XML file. An example XML section is listed below, illustrating the framework capabilities:

```
<wsn_control>
<global>
<param name="report_interval" type="int" unit="s">
<constraint type="le" value="3600"/>
<constraint type="gt" value="2"/>
</param>
</global>
<node type="raven">
<param name="node_name" type="string"/>
<param name="output0" type="bool"/>
<param name="pkt_forwarding" type="bool"/>
<param name="temp_alarm_enable" type="bool"/>
<param name="temp_alarm_low" type="int" unit="*C"/>
<param name="temp_alarm_high" type="int" unit="*C"/>
</node>
</wsn_control>
```

In this example, a global parameter named „report_interval" sets the polling period for data monitoring, in seconds, limited between 2 and 3600. Each Raven node can have a user-friendly name assigned, possibly suggesting its location or designated function. It's open-collector actuator control output can be enabled or disabled. It can be configured to forward data packets from farther nodes to the gateway. A temperature alarm service can be started on selected nodes, which in this example would sound the buzzer fitted to the node in case the measured temperature exceeds the settable high or low thresholds.

Where a parameter-based control interface is not sufficient, a command-line interface is provided, which is directly linked to the WSN-specific adaptation software without further processing by the control framework. This can be used to upgrade the software running on the nodes, for instance, by sending a .hex memory image file to the adaptation program.

In order to provide increased uptime, a supervisor program can watch that the various components of the framework are running correctly. It can for instance restart programs that have crashed or locked, such as the WSN driver (Sensinode for example uses a stand-alone process as a driver, to which the other programs connect through sockets), the node detection program or the main monalisa-wsn program.

It is important that the WSN monitoring service be able to run on a large variety of computer systems. Some WSNs for example are not connected to a PC, but use an embedded system such as an ATNGW100 for remote access. ApMon and monalisa-wsn are written in Perl, which is a portable scripting language. Perl can run on a large variety of computers, including embedded systems. The WSN drivers are usually written in C and can be compiled for mostly any system if their source code is available. The abstraction layer is written in Perl, Python and Unix shell, making it also highly portable.

The MonALISA graphical client (*Fig. 5, Fig. 6*) is written in Java, therefore it is capable of running on any modern PC operating system.

4. CONCLUSIONS

The capability to monitor and control WSN nodes and WSNs as a whole, without physical access to the nodes, from any remote location, is important when developing WSN software and when administering or using the WSN.

We developed a framework for monitoring and controlling WSNs through a uniform interface, independent of their hardware or software technologies. The framework allows monitoring of low- and high-level parameters and performance indicators for each WSN node. The framework allows parameter-based control of each WSN node, as well as console-based control for complex tasks. The framework also allows managing isolated WSN islands as a single entity.

The framework consists of WSN-independent programs and an abstraction layer composed of WSN-specific adaptation programs. These programs are written in a portable fashion, allowing them to be compiled and run on PC-class systems as well as embedded, resource-limited systems.

5. ACKNOWLEDGMENTS

Research for this paper is ongoing and is motivated by the authors' participation to the SENSEI (Integrating the Physical with the Digital World of the Network of the Future) Integrated Project in the EU's Seventh Framework Programme, in the ICT (Information and Communication Technologies). Thematic Priority of Challenge 1: Pervasive and Trusted Network and Service Infrastructures: ICT-2007.1.1: The Network of the Future.

REFERENCES

Callaway E.. (2004). Wireless Sensor Networks: Architectures and Protocols, *CRC Press*.

Dunkels A., Grönvall B, and Voigt T. (2004). Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: *IEEE Emnets 2004.*

Dunkels A., Vasseur JP.(2008). IP for Smart Objects, Internet Protocol for Smart Objects (IPSO) Alliance

Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K. (2000). System architecture directions for networked sensors. In: *ACM SIGPLAN Notices,* ACM, New York.

Lombriser C., Roggen D., Stäger M. and Tröster G (2007). Titan: A Tiny Task Network for Dynamically Reconfigurable Heterogeneous Sensor Networks. In: *Kommunikation in Verteilten Systemen (KiVS).* Springer Berlin Heidelberg, Berlin

Newman H.B., Legrand I.C., Galvez P., Voicu R., Cirstoiu C. (2003). MonALISA: A Distributed Monitoring Service Architecture. *CHEP03*, La Jolla, California

Newman H.B., Legrand I.C., Voicu R., Cirstoiu C., Grigoras C., Toarta M., Dobre C. (2004). MonALISA: An agent based, dynamic service system to monitor, control and optimize grid based applications. *CHEP04*, Interlaken, Switzerland

Rahimi M., Shah H., Sukhatme G., Heidemann J., and Estrin D. (2003). Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network. In: *Proceedings of the IEEE International Conference on Robotics and Automation, pp. 19-24*. Taipai, Taiwan.

Shelby Z. (2008a). Sensinode White Paper #1: IP-based Wireless Embedded and Sensor Networks: The WiFi of the Embedded World

Shelby Z. (2008b). Sensinode White Paper #2: Using Sensinode Products to Develop 6LoWPAN Networks

Texas Instruments (2007). A True System-on-Chip solution for 2.4 GHz IEEE 802.15.4 / ZigBee®. *http://focus.ti.com/docs/prod/folders/print/cc2430.html*.