Rapid prototyping suite of IEEE 802.15.4-compliant Sensor Networks

Mangesh Chitnis[1], Paolo Gai[2], Giuseppe Lipari[1], Paolo Pagano[1], and Antonio Romano[1]

[1]Scuola Superiore Sant'Anna
Piazza Martiri della libertà, 33
56127 Pisa (I)
{m.chitnis, lipari, p.pagano, a.romano}@sssup.it

[2]Evidence s.r.l.
Via Moruzzi, 1
56124 Pisa (I)
pj@evidence.eu.com

Abstract

This paper presents a toolsuite for rapid prototyping and implementation of real-time applications on Wireless Sensor Networks. The work is motivated by the need to use WSNs in industrial control contexts, where the sampling rate, the workload is much higher than typical current applications of WSNs, and the real-time constraints are much tighter. We present a simulator for early evaluation of the real-time behavior of a WSN application; and a real-time operating system that implement appropriate real-time scheduling policies to allow timing analysis and guarantee timing constraints. We provide a demo based on a simple but realistic network scenario showing that simulation is in agreement with experimental results.

1 Introduction

Many companies have recently started to consider the use of Wireless Sensor Networks (WSN) in industrial automation systems and process control. The most urging motivation is the need to reduce the amount of physical wires in a industrial plant. Reducing cables can significantly reduce the cost for building and maintaining the system. Moreover, with less cables it is easier to dynamically reconfigure the machines, and implement plug-and-play components. An example of research effort in this sense is the the RI-MACS project [5], whose goal is to increase reconfigurability and adaptiveness of industrial automation platforms by using WSNs in selected cases. In the context of RI-MACS, we are investigating the possibility to use WSNs for process control and configuration.

However, the requirements of these systems are substantially different from those of other domains of WSNs. In addition to requirements for increased robustness and fault tolerance, each node is expected to perform a substantial amount of computation in real-time. The rate at which data must be sampled is higher than typical WSNs application for environmental monitoring. For example, the sampling rate of a sensor can go up to hundreds of Hz. Also, many tasks may need to be executed concurrently. For example, we may have tasks for data filtering, actuation, diagnostic, logging, communication, etc. In practice, the computational load on each node (in terms of amount of processing time needed by the application tasks) may become relevant. In addition, such applications exhibit real-time constraints, sometimes hard real-time ones. Many activities, like sampling and actuation, must be triggered periodically and executed with bounded response time, and late sensor messages may not be considered acceptable, otherwise the system may not work properly.

In a real-time system, it is important to guarantee system *predictability*. The designer must check at system design time that the timing constraints will be respected under all conditions. A schedulability analysis for real-time applications must be performed to guarantee that the system will work properly under worst-case load conditions.

Unfortunately, the use of proper real-time mechanisms in WSNs has not been deeply investigated until now. While few real-time Operating Systems have been proposed in the research literature and in the commercial market, TinyOS is by large the most popular Operating System for WSN. Unfortunately, TinyOS does not support predictable real-time scheduling policies. At the network level, mechanisms for real-time communication and Quality of Service (QoS) control (like the Collision Free Period access included in IEEE 802.15.4 standard[3]) are not implemented. But most importantly, what is missing is a proper comparative study and evaluation of the impact that various communication protocols and Operating System policies have on the real-time behavior of the system.

Another problem concerns the lack of proper simulation tools. Simulation plays an important role both for research and for industrial practice. In academic research, simula-

1-4244-1455-5/07/$25.00

tion is a fundamental tool to compare different algorithms and protocols, and to assess the performance of proposed solutions on complex realistic settings. In industrial design, it is important to simulate the system before deployment for early assesment of performance, for system dimensioning, and to identify potential bottlenecks and problems.

2 The rapid prototyping suite

We propose to the community under GPL license:

- the RTNS [6] package, able to model and simulate, not only the functional but also the temporal behavior, both at the network level and at the CPU level;
- the Erika Enterprise [2] Operating System customized for Wireless Sensor Network.

Together these tools play the role of a rapid protyping suite, permitting to simulate and test in real hardware a variety of protocols and distributed applications suitable for real-time operations of WSN.

2.1 The simulator

The RTNS simulator is a combination of the popular NS-2 package [4] for network protocol simulation, and of RTSim [7] for Real-Time OS (RTOS) simulation. Thanks to our simulator, it is possible to model the real-time tasks running on each node, with their priority, execution time, and a high-level pseudo-code modelling their functional behavior, and the network protocol used for communication between nodes. In this way, it is possible to model end-to-end activities, and measure the delay in transferring data from sensor to destination.

The RTNS allows to simulate the networking aspects via NS-2 as well as the real-time Operating Systems aspects via RTSim. In NS-2, protocols are implemented as Agents: any class that implements a protocol has to extend the Agent class. Instances of an agent class are the endpoints of wired and wireless connections. They are identified by INET address and port and are the lowest layer able to pack and inject messages into the network. Application code is modeled by the Application class. Applications use agents to send and receive messages.

To simulate the behavior of the Operating System running on a node we construct an NS-2 Application called RT-App abstracting all the features of a Real-Time Kernel. The simulator creates an instance of RT-App for each node, and enables the connected agent (RT-Agent) to send and receive data packets. RT-App creates aperiodic tasks for "networking" and "computational" purposes and schedules them adopting one of the policies implemented in RTSim, e.g. First Come First Served (FCFS), Earliest Deadline First (EDF), Fixed Priority (FP), etc.

The Operating System puts these tasks into the Running status following the scheduling policy selected at the time when RT-App is instantiated.

Because of the adopted scheduling algorithm, the tasks can be delayed by some time from the activation. In the case of a priority-based scheduler like FP, the preemption of the CPU depends on the difference in priority between the running task and the one which has been suddenly activated. In a scheduler based on activation times, the tasks are queued up and executed in FIFO order. In the latter case, the user is not asked to specify a priority ad the delays are strongly dependent on the load of the node.

RTNS can simulate single and multi-hop communications for all the network topologies defined in the IEEE 802.15.4 standard. The MAC and Physical layers of the Network Stack are natively simulated by NS-2 and RTNS only wraps these services into tasks handled by RT-App.

2.2 The Operating System

The OS counterpart in this prototyping suite is ERIKA Enterprise (EE) commercialized by Evidence srl. EE consists of a single and multi-processor real-time OS kernel implementing a collection of APIs similar to those of OSEK/VDX standard for automotive embedded controllers. It implements a shared memory model for a layered architecture composed by a substrate Kernel acting on a Hardware Abstraction Layer (HAL) dependent on the specific platform. A set of software modules implements the task management and the scheduling policy.

It is well suited to respond to the stringent pre-requisites of Wireless Sensor Networks, because of:

- the minimal footprint in terms of RAM (about 3 KBytes), partially dependent on the hardware platform and the adopted configuration;
- the hard real-time compliance, because it complements FP scheduling providing support for Immediate Priority Ceiling and Preemption Thresholds on single CPU systems;
- the supported Atmel AVR 5 architecture, very popular in WSN contexts;
- the application layer customizable to fit diversified scenarios where WSN are deployed.

To implement the Network Stack substrate we adapted the libraries provided by Atmel[1] and compliant with the IEEE 802.15.4 standard. The Atmel set of functions, organized as a network library, have been linked to ERIKA core.

To the best of the authors knowledge, in so far only a few Operating Systems have been ported to AVR 5 architecture; namely Nano-RK, TinyOS and recently ERIKA.

Conditional compilation is used to customize the cases of ordinary devices and Personal Area Network (PAN) coordinator.

The network initialization is organized by means of an aperiodic task: on the coordinator side, such a task establishes a new network, sets the communication mode (peer-to-peer or beacon mode, single or multiple cluster, etc.), and runs the negotiation for device association (until the devices are registered in the PAN Information Base, PIB); on the device side, the task activity goes through the association negotiation stage, ending when the device is registered in the PIB.

Following the fixed priority scheduling policy implemented in EE, the concurrent execution of computational and network tasks is dependent on the priority assigned at configuration time to each of them. Flattening the priorities of all the tasks, the scheduling policy reduces to FCFS (the one adopted in the popular TinyOS operating system).

3 Proposed demo

As a simple demo we *deploy and simulate* a single clustered, star shaped, WSN where four nodes located at the edges of a square generate Constant Bit Rate (15 Hz rate) traffic towards the PAN Coordinator, sitting in the center of mass of the figure.

The operations are run in RTNS and in a real setup.

The latter consists of 5 Atmel STK-500/501 development boards equipped with Atmega 128 CPU platform and AT86RF230 2.4 GHz transceiver peripheral. We program the micro-controller through a JTag ICE mkII debugger making use of the facilities offered by the AVR-STUDIO IDE.

The sink, sitting in the center, is connected to a PC through a serial line. On the PC a server is running the Data AcQuisition (DAQ). In the initiating devices, two tasks are managed by ERIKA in FP: the net task taking care of the operations related to the Network Stack and the send task (with maximum priority) assuring a coherent trigger of periodic transmissions. In the recipient device, we modeled the activity of the PAN coordinator with two tasks managed either using FCFS or FP: the net task and the load task modeling the computational activity.

A tunable computational activity is obtained by means of a set of periodic tasks with fixed Period (T) and different Execution Time (ET). The ET variations are avoided excluding conditional branches from the task body. Making use of the external interrupt sources (the buttons placed on the STK 500 development board), it is possible to suspend the running task and suddenly activate another one causing some load change.

In the Atmega 128 microcontroller, the hardware interrupts raisen by the arrival of data frames triggers an Interrupt Service Routine (ISR) initiating the data reception at the Physical layer of the network stack. The reception completes at the MAC layer whenever the payload is extracted from the incoming data frames: depending on the adopted scheduling policy and the computational load present at the sink, the completion may be delayed by some time.

Fixing a deadline for reception completion, the demonstration, making use of visualization tools, shows how the number of deadline misses increases as a function of the load. We display the plots coming from simulation and real hardware to demonstrate the reliability of this prototyping suite. In Figure 1, as a matter of possible example, the effect on the delay in message reception caused by a sudden change in ET of the load task is shown. Adopting FCFS scheduling policy, as the ET increases, the receive task has to wait longer to get scheduled and sometimes misses the deadline. Switching to FP solves the problem since the delay in message reception is insensitive to the change in ET.

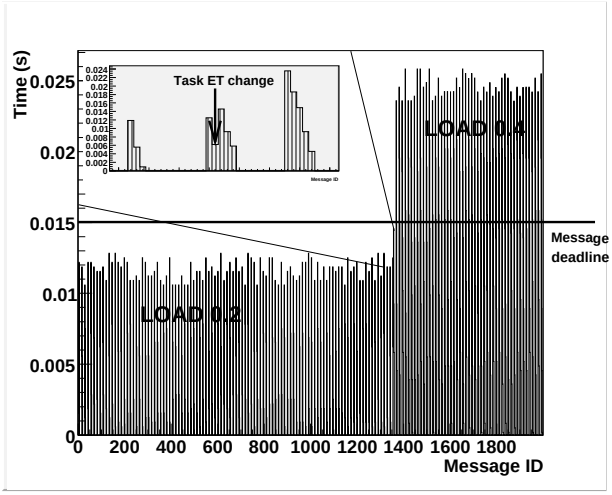

Figure 1. Message delay in packet reception as a function of transmission ID adopting FCFS scheduling policy. The message reception deadline is shown as a level in the plot.

References

[1] Atmel Corporation. http://www.atmel.com.

[2] E.R.I.K.A. http://erika.sssup.it/.

[3] The IEEE 802.15.4 standard. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf.

[4] Information Sciences Institute (University of Southern California, Los Angeles CA, USA), The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/.

[5] The RI-MACS EU project (NMP2-CT-2005-016938). http://www.rimacs.org.

[6] The RTNS simulator. http://rtns.sssup.it.

[7] The RTSim simulator. http://rtsim.sf.net.