A Framework for Modeling Operating System Mechanisms in the Simulation of Network Protocols for Real-Time Distributed Systems

Paolo Pagano, Prashant Batra, and Giuseppe Lipari

Scuola Superiore Sant'Anna
56127 Pisa, Italy
{Paolo.Pagano, Prashant.Batra, Giuseppe.Lipari}@sssup.it

Abstract

In this paper we present a software tool for the simulation of distributed real-time embedded systems. Our tool is based on the popular NS-2 *package for simulating the networking aspects, and on the* RTSim *package for the real-time operating system aspects. By reusing much of the existing code, our simulator covers a very wide range of network protocols and real-time mechanisms.*

After describing the architecture of our tool, we tested it in a simple wireless sensor networks scenario, and we measured the latency in transmitting and receiving messages due to the concurrent activities in the nodes. These effects have been tested against two node scheduling policies, and under different load conditions in the CPU of the nodes.

1. Introduction

Simulation is an important analysis tool in the development of distributed systems, in testing new network protocols, and for assessing the performance of protocols. In many important cases, simulation complements effectively off-line mathematical analysis tools, especially for large and complex systems with hundreds of nodes.

Performance assessment via simulation is particularly important for embedded real-time systems, where timing guarantees (delay and throughput) play a very important role. In a hard real-time system (avionics systems, industrial control, etc.), missing a deadline on some data delivery may in certain cases compromise the correctness of the system. Soft real-time systems (multimedia devices, telecoms, etc.) are less critical, but the Quality of Service (QoS) delivered to the final user depends on some performance index like average and maximum delays.

1-4244-0910-1/07/$20.00

As with any other system, the simulation of a real-time distributed system consists first in building a model of the system (network topology, protocols and behavior inside the nodes) and then a model of the external environment (incoming packets, sensor data and events, etc.) with which the system interacts. The model must be *reasonably precise*: it must be able to capture all the *interesting* behaviors of the system without being too complex.

Existing software tools for network simulation, and in particular the popular NS-2 [2] (Network Simulator 2), allow to simulate a wide range of network protocols and distributed systems, both wired and wireless, including very dynamic systems like Mobile Ad-hoc Networks (MANETs) and Wireless Sensor Networks (WSNs). These tools have been specifically designed to study the behavior of the communication. To the best of the authors' knowledge, in all simulation models reported in the literature until now, the temporal behavior of the application running in the node is not modeled. In particular, all computations performed in the node (for example the presence of load due to other applications) have zero delay. The reason is that for modeling network effects, the impact of the application timing and of the operating systems can be considered negligible. However, for real-time embedded systems, the impact of the other processing activities and of the operating system delay in the node can be quite relevant.

In this paper, we address this problem by presenting a software tool based on the NS-2 simulator [2] and on the RTSim simulator [18] that allows to easily simulate the networking aspects via NS-2 as well as the real-time operating systems aspects via RTSim. Although the use of the tool is very general, in this paper we will demonstrate the usefulness on a specific problem, that is to evaluate the reduction in some indicators of QoS in a very simple WSN scenario.

The paper is organized as follows: in Section 2 we describe the state-of-the-art in network and os simulation and motivate the choice for building a software tool based on

NS-2 and RTSim; in Section 3, after a brief academic introduction on both the packages, we describe how we integrated them; in Section 4 we recall some RT issues relevant in WSN applications; in Section 5 we briefly describe how we modeled the behavior of the TinyOS [10] operating system; we also show the results of a simulation run in a very simple WSN scenario comparing such results with those obtained with a simple Fixed Priority (FP) scheduling model; in Section 6 we sketch a possible interpretation of the results.

2. Existing simulation engines

The type of simulation we are concerned with is discrete and event-driven. We are interested in simulating distributed systems, with particular concern on MANETs (Mobile Ad-hoc Networks) and WSN (Wireless Sensor Networks).

In what follows we briefly report the main features of the most popular simulators available for the scientific community and designed to reliably simulate MANETs and WSNs; for a complete survey see [13].

OPNET (Optimized Network Engineering Tools) [3] is a commercial tool from OPNET Technologies Inc. for modeling and simulation of communications networks, devices, and protocols. Although OPNET is rather intended for companies to diagnose or reorganize their network, it is possible to implement one's own algorithm by reusing existing components.

GloMoSim (Global Mobile Information Systems Simulation Library) [8] is a scalable simulation library designed at UCLA Computing Laboratory to support studies of large-scale purely wireless network models. GloMoSim is a library for the C-based parallel discrete-event simulation language PARSEC (Parallel Simulation Environment for Complex Systems) [9]. One of the important distinguishing features of PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. However, the documentation shipped with GloMoSim is quite poor as well as the set of standard tools for scenario generation and post-simulation accessories.

QualNet [4] is a commercial product from Scalable Network Technologies (which is derived from GloMoSim) trying to alleviate most of the GloMoSim's flaws, coming with an extensive suite of faithful implementations of models and protocols for both wired and wireless networks as well as extensive documentation and technical support.

New platforms written modularly and using object oriented techniques are OMNeT++ [5] (written in C++) and J-Sim [1] (written in Java). Both have strong GUI support and flexible architecture and are rapidly becoming popular simulation platforms in the scientific community as well as in industrial settings.

A different role is played by the TOSSIM simulator, coming along with the TinyOS operating system. It compiles directly from TinyOS code using a special target in the Makefile. The simulation runs natively on a desktop or laptop. The simulator is capable to simulate thousands of nodes simultaneously. Every mote in a simulation runs the same TinyOS image. TOSSIM provides run-time configurable debugging output, allowing a user to examine the execution of an application from different perspectives without needing to recompile. TinyViz is a Java-based GUI that allows the user to visualize and control the simulation as it runs, inspecting debug messages, radio and UART packets, and so forth. The simulation provides several mechanisms for interacting with the network; packet traffic can be monitored and packets can be statically or dynamically injected into the network. The transmission is simulated at the bit level.

The TOSSIM and TinyViz simulation capabilities are anyhow constrained to TinyOS based applications (protocols and modules already implemented in TinyOS); moreover they can be seen more as debuggers or emulators, rathen than simulators.

The validity of these packages as well as of others not even mentioned in this paper is doubtless; anyhow a share ranging from 40% to 70% [14] (depending on the network layer) of the existing simulations in the world are run through the NS-2 package which plays the role of a "de facto" standard. The back-end (i.e. the skeleton classes) of the package is written in C++, whereas the OTcl scripting language plays the role of front-end to ease the generation of network scenarios and activities. The transmission is simulated at the packet level and the propagation models are built in the package. More details will be given in Section 3.1. The diffusion in the telecommunication community and the existence of a module [23] for simulating the 802.15.4 IEEE standard MAC layer already included in the official distribution let us decide to adopt NS-2 for simulating the network transmission in a WSN.

In the operating system area, there is not such a widely used simulation package as NS-2. Rather, it looks like every research group uses its own simulator. Many operating systems simulators are available for didactic purposes. Here, we cite MOSS (Modern Operating Systems Simulators) [17], a collection of Java-based simulation programs that is used to illustrate key concepts of operating systems in university courses. Generally, such packages are difficult to re-use in different contexts. In particular, MOSS does not support real-time scheduling policies and interaction with the network.

RTSim [18] is a software package written in C++ for the simulation of real-time operating systems, available as open source [6]. It includes support for many real-time schedul-

ing policies and typical real-time task models (i.e. periodic and event-driven tasks, and interrupt handlers). In this paper, we propose to combine RTSim with NS-2 for the simulation of real-time distributed systems. The structure of RTSim is described in Section 3.2.

3. The simulation framework

3.1. The NS-2 simulator

NS-2 is a tool for simulation and evaluation of network protocols. The tool is distributed as open source, and the size of the source code is about 400 Megabytes. Most of MANET and Ad-Hoc routing protocols are already available in NS-2.

For efficiency reasons NS-2 is written in C++ and OTcl, the latter being an object-oriented scripting language. The C++ part is composed of schedulers and a great variety of network components. Implementation of new protocols will mainly occur in this part. The OTcl part is composed of libraries that gives access to the C++ objects. Definition and configuration of network scenarios is done in OTcl.

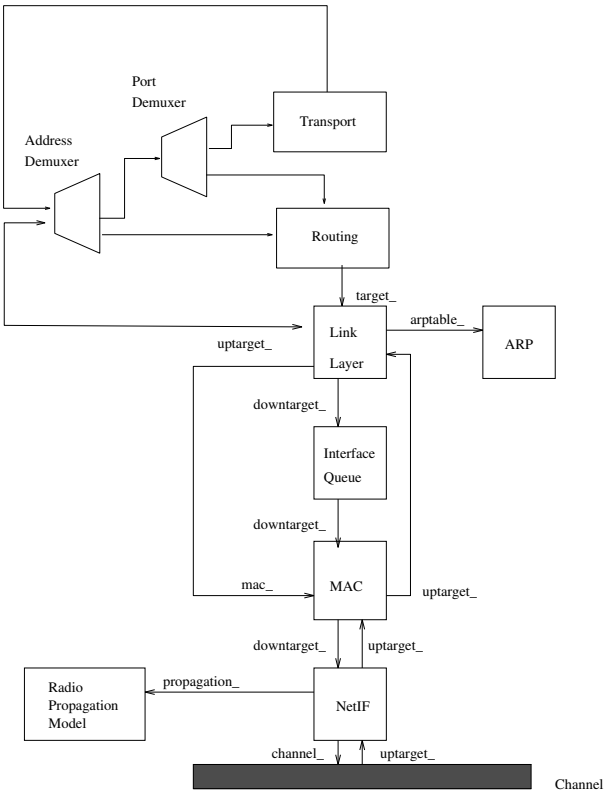

Figure 1. Architecture of a wireless node in NS-2.

The main C++ classes relevant for our simulation are: Event, Scheduler, Packet, and Agent. The scheduler contains a queue of events ready to be executed, ordered by scheduling time. The scheduler runs by selecting the first event in the queue, executing it to its completion, and returning to execute the next event.

Packets are events that are handed to the scheduler before their sending time. A packet models a transmission of a message in the network, but does not model the actual content of the message. Therefore, the packet class need only to contain the header specification of the packet for processing by the respective protocols.

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 end-points of wired and wireless connections. They are identified by INET address and port and are the lowest layer able to pack and insert messages into the network.

Application code is modeled by the Application class. Applications use agents to send and receive messages. If an application runs on top of an agent, NS-2 allows the agent to partially analyze the packet before passing it to the application. This key feature of NS-2 has permitted us to build up a co-simulator based on the RTSim libraries.

Figure 1 illustrates the architecture of a mobile node with all its component types. These components can be substituted with standard components or newly implemented ones.

Let us analyze what happens when a packet has to be sent to the network. First, the agent (through the transport component shown in the top part of Figure 1) creates the packet and hands it to the address demuxer. The address demuxer checks the destination address of the packet. If the destination address is the same as its own, the packet is handed to the port demuxer. Otherwise the packet its handed to the routing component. The routing component will behave depending on the routing protocol used. For example, if the routing protocol is AODV (Ad-hoc On Demand Vector routing), assuming that no route exists to the destination address, the protocol module will start a route request session. When the routing component has a route to the destination, the packet is handed to the LL component, and then handed down to the NetIF component, passing through the IFq and MAC components. From the NetIF, the packet is inserted into the channel. It is also possible to disable routing (adopting the so called "DumbAgent") in case of a simplified mesh scenario.

Definition and configuration of the network nodes has to be done in OTcl. In this step the new protocol is attached to each node. When the OTcl-script is executed, the program outputs a trace file. Graphic representation of trace files is also possible with the Network AniMator (NAM), a visualization tool that shows node movement and communication.

3.2. The RTSim simulator

RTSim is a collection of C++ libraries that allows the user to model and simulate single and multiprocessor embedded systems, with a real-time operating system and a set of concurrent tasks. A distinctive feature of the tool is that it encompasses the state-of-the-art solutions for real-time CPU scheduling (either on single or on multiprocessor boards) and for bounded-delay sharing of resources.

In RTSim, a task (or process) is a finite or infinite sequence of requests for execution, or *jobs*. Each job executes a piece of code (a sequence of instructions) implementing some functional behavior. When a job is activated, we say that it *arrives* and the activation time is called *arrival time*. Depending on the pattern of arrival times, tasks can be classified as *periodic*, if the arrivals are separated by a constant interval of time called "period"; *sporadic*, if the arrivals are separated by variable intervals of time with a lower bound, called *minimum inter-arrival time*; and *aperiodic*, if a lower bound is not known on the inter-arrival times.

In real-time systems, tasks have time constraints, often expressed as *deadlines*: for example, a typical time constraint for a periodic task is that each job must finish before the next activation.

The *instructions* of a task are used to model its timing and functional behavior. Basically, an instruction is modeled by an execution time (which can be deterministic or stochastic) and a behavior (which modifies the state of the task and of other system components). Examples of instructions are:

- delay(*time*): this instruction models a piece of code that takes a certain amount of time to be executed; *time* can be a fixed number or a random variable. When using this instruction we are not interested in modelling the functional behaviour of the code.

- wait(R) and signal(R); these instructions model wait and signal operations on a semaphore R. Instruction wait(R) can block the task.

Tasks are assigned to the computational resources (nodes) of the system. Each node consists of one or more processors and a real-time operating system (kernel) endowed with a scheduling policy and a synchronization protocol. The state of the art algorithms for CPU scheduling (such as Fixed Priority, Rate Monotonic [16], Earliest Deadline First (EDF) [16], Proportional share [21]) are provided as predefined objects, both for single processor and multiprocessor systems. The performance of the schedulers can be enhanced by using aperiodic servers (Polling server [22], Sporadic Server [20], Constant Bandwidth Server [11], etc). Priority inversion in accessing mutually exclusive resources [19] can be avoided by using appropriate synchronization protocols, such as the Priority Ceiling Protocol [19] or the Stack Resource Policy [12].

Each scheduling point in the system corresponds to a discrete Event in the simulation. Events are handled via a discrete event simulation engine called MetaSim. In response to an event in the simulation, the engine calls an event handler that performs the corresponding action, changing the state of the system. For example, when a task is activated, an event invokes a handler that puts the task in the ready queue and invokes the node scheduler. Such event-driven mechanism is very similar to the one used by NS-2. Therefore, by opportunely adapting the MetaSim simulation engine, we were able to integrate RTSim with NS-2 with a moderate effort.

3.3. Integration

At hand we had two simulation frameworks, with their own simulation engines. We decided to keep NS-2 event scheduler as the main engine, and make the RTSim engine (actually MetaSim engine) as its sub-engine. We defined a special event in NS-2, called the rtsim_event that takes care of processing all events of RTSim that happen at a single point in time.

Whenever an object of RTSim posts an event in the Metasim event queue at simulation time t, the rtsim_event is posted in the NS-2 global event queue at the same simulation time t. When this event expires, the corresponding event handler processes all events that have triggering time t in the Metasim queue. In this way, we keep the logical simulation time of NS-2 synchronized with the logical simulation time of RTSim.

To allow the user to model tasks that send and receive packets from the network, we defined two new types of tasks: the Sender and the Receiver tasks, which can be interfaced with Agents of NS-2 which in turn model network end-points. We also defined two new task instructions: the send and the receive instructions. Suppose the user needs to model a task that periodically wakes up, performs some computation and then sends out a set of data. The user will only need to create a periodic real-time Sender task with a delay instruction that models the time spent in the initial computation, and a send instruction that models the transmission of data to the Agent. The task is subject to the scheduling algorithm in the node, so it may be delayed by other tasks. The actual sending of the data is done only when the corresponding send instruction is executed.

In Figure 2, we describe the sequence of actions (and events) that take place when a send instruction is executed. First, the task object of RTSim posts a "send instruction event" in the Metasim queue. When this event expires, the send instruction code is executed, which notifies the corresponding agent. In turn the agent posts a sequence of

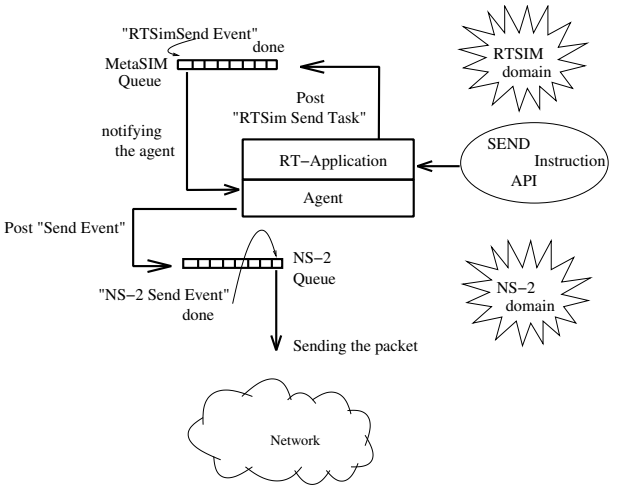

Figure 2. The send instruction API.

events in the NS-2 queue (the exact number and sequence of events depends on the specific network protocol stack) and, at the end, the packet is actually sent on the network.

A similar mechanism happens with a receive instruction. The main difference is that the receive blocks the task if there is no pending data.

4. RT issues in WSNs

The scope of the proposed package is actually very large and is not tightly connected with WSN issues. Wherever any service must be guaranteed in a network (the extent of this guarantee is known as the contract), the context of the OS execution can be taken into account for comparing scheduling policies or to realistically refine the contract.

To some extent, research in the WSNs does not deal with QoS, since many services are given at "best effort" and the single nodes are intended as unreliable. This attitude is somewhat partial and incorrect looking at the variety of existing and proposed applications of sensor networks. In some applications, the impact of the tasks scheduling of the node can be relevant. For example, WSNs for vibrational monitoring and control may have considerable computational load on each node due to the high sampling frequency required. Therefore, efficient scheduling policies in the node may affect the overall performance of the system. Citing H. Karl and A. Willig [15] with respect to the challenges for WSNs we say "[...] there are cases where very high reliability requirements exist. In yet other cases, delay is important when actuators are to be controlled in a real time fashion by the sensor network. The packet delivery ratio is an insufficient metric; what is relevant is the amount and quality of information that can be extracted at given sinks about the observed objects or area. [...]".

These issues invest the communication itself as the access to the shared medium is regulated by the MAC layer. The directives introduced by IEEE 802.15.4, the one adopted by the ZigBee[7] alliance, foresee a TDMA mechanism to access the network. In this case, any tiny transmission latency induced by OS activity may let the node miss to communicate during the GTS (Guaranteed Time Slot) causing longer delays in message delivery.

5. The simulated environment

5.1. The RTOS platform

In this first attempt to simulate RT applications by NS-2, we modeled TinyOS, the most popular Operating System used in WSNs and compared its behavior with a standard model using Fixed Priority (FP) scheduling.

In TinyOS, tasks are scheduled in FCFS order and cannot be preempted by other tasks, thus running to completion. Interrupts handlers, arisen by hardware, can preempt task code.

In Figure 3 the execution of a receive instruction is shown in the context of FCFS and FP scheduling policies. The example refers to a node with 3 periodic tasks with different execution times activated simultaneously at time 0. In case of FCFS the task in charge of processing the incoming message is queued up and the reception delayed; in the case of FP, the processor is preempted at the arrival of the message. The delay in receive marked in the figure is communicated to the NS-2 Agent by the NS-2 RT-Application. The agent takes note of this delay (ΔT (RecOS)) as a function of the message unique ID field in the common header.

The modeled behavior of the Sender task follows the same scheme. In this case the agent saves the delay (ΔT (RecOS)) as a function of the message unique ID.

The extra delay in receive already simulated by ordinary NS-2 timing facilities, i.e. the time spent by the packet to reach the agent of the recipient node is taken into account as well. This delay (ΔT (RecNet)) amounts to the sum of the propagation of the request descending and climbing the network stacks in the two nodes plus the propagation time of the signal in the shared medium as shown in Figure 4. Following the naming schemes used in the figure this delay amounts to $\Delta T(M_1) + \Delta T(P_1) + \Delta TNet + \Delta T(M_2) + \Delta T(P_2)$.

5.2. System and network models

The simulated scenario has been kept as simple as possible to permit a reliable interpretation of the results: 4 nodes in shape of a square and a sink located in its center of mass (see Figure 5). All the simulated effects which might give

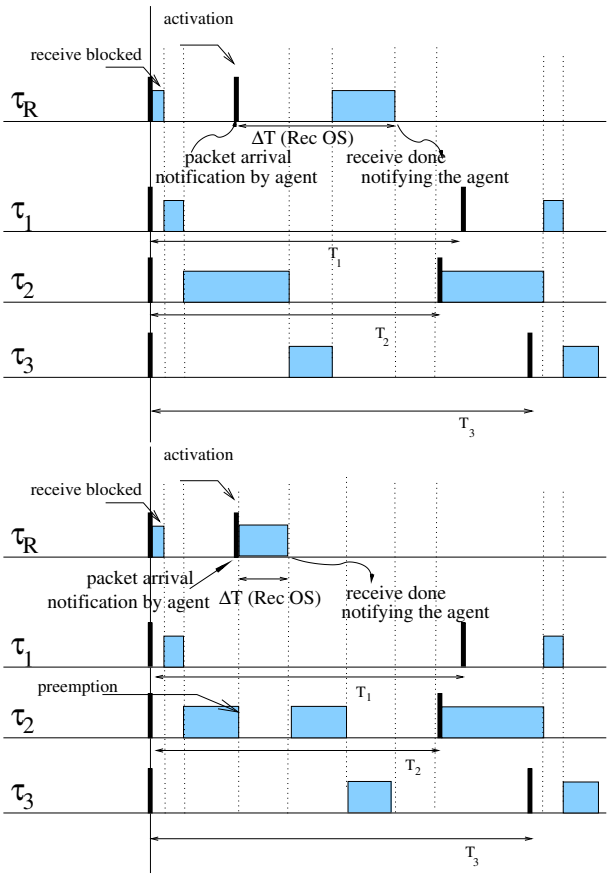

Figure 3. The model for the receive task in the TinyOS environment and with a FP scheduling.

rise to disturbance comparable with the OS induced delays have been turned off; namely:

- the network has been simulated as a single cluster where all nodes can be connected through single hop routes (no ad hoc routing);
- each node is within from every other as results from setting the power thresholds in receive and carrier sense according to the maximum distance between the nodes; moreover the problem of collisions with hidden terminals is avoided;
- all the nodes are aware of the INET coordinates of the sink.

The simulation adopts the TwoRayGround wave propagation model embedded in NS-2. For Physical and MAC OSI layers we adopted the NS-2 implementation as a black box, namely an order 3, beacon-enabled superframe without GTS. The sink played the role of PAN coordinator as well.

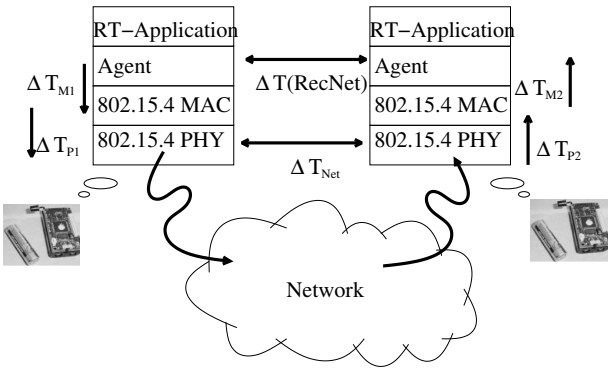

Figure 4. The agent-agent delay as computed using the ordinary NS-2 timing.

In order to simulate the kernel activity under different conditions, an API has been written to permit the creation through the OTcl interface of a certain number of periodic tasks and a given system load. These tasks are called the Dummy Task Set (DTS) in the following. DTS tasks have periods ranging from 0.1 s to0.5 s and generate a cumulative load ranging from 0 to 0.95. In the same API, we added the possibility of selecting the OS scheduling policy (as at present FCFS or FP).

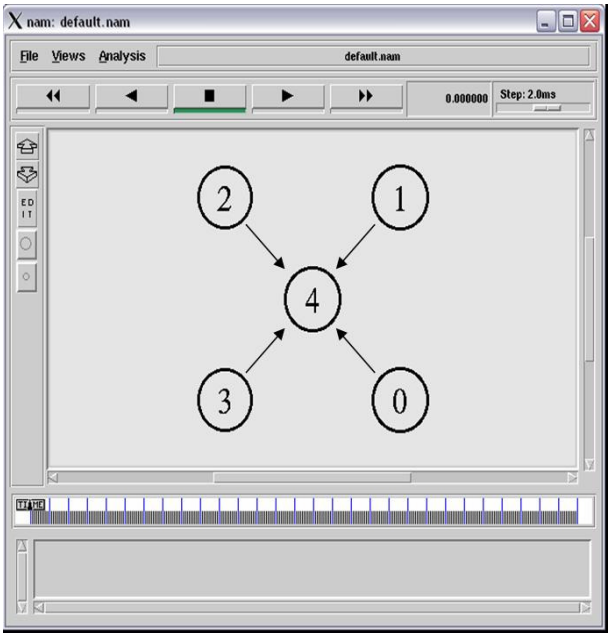

Figure 5. Elaborated screen shot of the Network AniMator for a simple WSN scenario.

The network activity in the node is simulated through the Sender and Receiver tasks; in case of FP scheduling they

have higher priority than the DTS.

In this scenario, after a start-up time t_0 (the initial phase to permit PAN initialization), the node sends packets to the sink at a certain rate ν. This means that a customable number N of messages will be scheduled to be sent from node A_0 to the sink S at instants:

$$t_i^0 = t_0 + i \cdot \frac{1}{\nu} \quad i = 0, 1, ..., N$$

The nodes A_k (k = 1,2,3) will start transmitting to the sink after a fixed delay $k \cdot t_{step}$ and their messages will be scheduled at instants:

$$t_i^k = t_0 + k \cdot t_{step} + i \cdot \frac{1}{\nu} \quad i = 0, 1, ..., N;\ k = 1, 2, 3$$

The kernel load will delay the send and receive by a certain amount of time dependent on the number of tasks and the load present in the node, and on the repetition time $\frac{1}{\nu}$ which may cause congestion in the network. The $k \cdot t_{step}$ factor prevents initial congestion and systematic collisions.

5.3. Results

The delays have been evaluated as a function of the dimension of the DTS, the CPU load in the node and the repetition time. Fixing such a set of parameters we evaluate:

- the maximum, mean and minimum latency observed in sending and receiving the packets;
- the time propagation through the sender and recipient network stacks (including re-transmissions done at lower layers);
- the packet probability of being delivered.

Though the applications can be sensitive to one, more, or all these metrics, some general considerations can be given as follows:

- as expected, using the FP scheduling policy with higher priority to Sender and Receiver, the delay is insensitive to CPU activity;
- the FCFS scheduling policy is conformant only to moderate CPU loads in presence of RT issues; in presence of medium-to-high CPU loads a real-time scheduler, as FP, must be preferred;
- the delays are inversely proportional to the repetition time of the packet transmission by the node; for sufficiently sporadic transmissions ($\frac{1}{\nu} \geq 0.4s$) this effect is reasonably small;
- the dependence on the number of tasks is moderate;

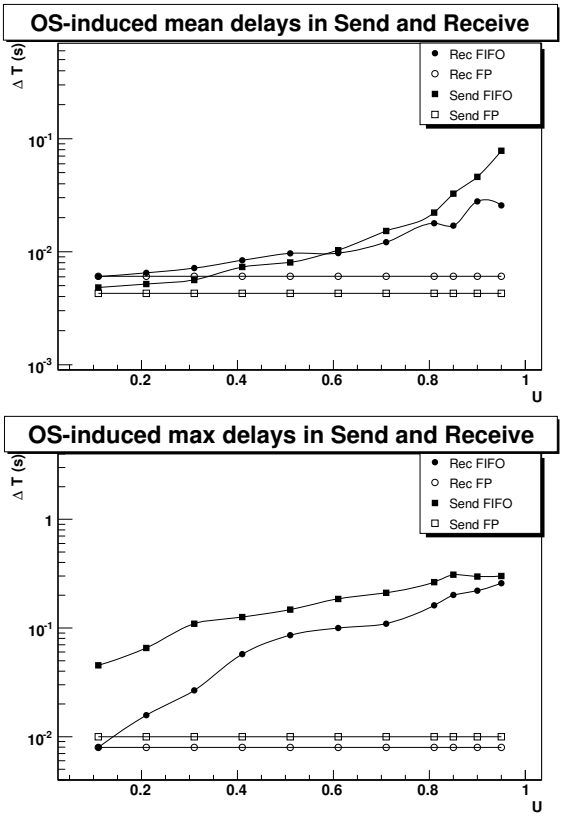

Figure 6. The mean and maximum delays in message elaboration due to the CPU activity as a function of the utilization factor for FCFS and FP scheduling policies.

- with standard PHY, MAC, and LL settings, the packet loss is negligible.

Being interested in WSN typical operations, we show the results for the relevant metrics having reasonably fixed the following parameters in operational plateaus. Namely we set:

$$\frac{1}{\nu} = 1\ s; \qquad t_0 = 5\ s; \qquad t_{step} = 1.1\ s;$$
$$N = 500; \qquad \text{dimension of DTS} = 3.$$

In Figure 6, the OS-induced delay as a function of the load in the sink is shown. The mean (top canvas) and the maximum (bottom canvas) values for sending and receive are displayed as computed from a sample composed by 20,000 exchanged messages. Notice that, in case of FCFS scheduler, the delay grows exponentially with the load, while by using a FP scheduler, the delay is independent of the load.

6. Conclusions and outlook

To simulate real-time distributed embedded systems, we integrated NS-2 and RTSim in a single framework. We have used this tool to model RT issues in wireless telecommunications. In WSNs these issues play a role whenever any QoS must be guaranteed by the nodes.

Ordinary WSN activity has been tested in a simple scenario to evaluate some metrics relevant in RT sensitive applications. Modules implementing FCFS and FP scheduling policies have been tested within certain network and CPU load conditions.

Apart from quantitative comparisons which may be affected by the embedded system features (e.g. the Instruction Set Architecture – ISA and CPU speed of the sensor node), the trend in the simulation shows that FP scheduling policy has to be preferred to FCFS whenever the computational load in the nodes increases. OS adopting FCFS scheduling policy like TinyOS may be not suitable for such operations.

As on-going work, more complicated scenarios are being simulated introducing routing paths and data streams; in a tree-shaped network topology, nodes connecting different clusters may fetch and forward the readings coming from other clusters. The overall RT metrics strongly depend on promptness of such nodes. These simulations are closer to realistic WSN operations and will be tested with CPU-intensive protocols.

Acknowledgements

This work has been financially supported in part by the European Union in the framework of the RI-MACS project (NMP2-CT-2005-016938).

We would like to thank Claudio Cicconetti for his exceptional expertise on NS-2 and Antonio Romano for his support on hardware modeling of WSN nodes.

References

[1] Illinois Network Design and EXperimentation (INDEX) Group (University of Illinois at Urbana-Champaign, IL, USA). The J-Sim Simulator. http://www.j-sim.org/.

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

[3] OPNET Technologies, Inc., Bethesda, MD, USA. The OPNET Simulator. http://www.opnet.com/.

[4] Scalable Network Technologies, Inc., Culver City, CA, USA. The QualNet Simulator. http://www.scalable-networks.com.

[5] The OMNeT++ Discrete Event Simulation System. http://www.omnetpp.org/.

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

[7] The ZigBee alliance. http://www.zigbee.org.

[8] UCLA Computing Laboratory (University of California, Los Angeles CA, USA). The GloMoSim simulator. http://pcl.cs.ucla.edu/projects/glomosim/.

[9] UCLA Computing Laboratory (University of California, Los Angeles CA, USA). The PARSEC environment. http://pcl.cs.ucla.edu/projects/parsec/.

[10] University of California, Berkeley CA, USA). The TinyOS operating system. http://www.tinyos.net/.

[11] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[12] T. P. Baker. Stack-based scheduling of real-time processes. *Real-Time Systems*, (3), 1991.

[13] G. A. Di Caro. Analysis of simulation environments for mobile ad hoc networks. Technical report, Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2003.

[14] T. Henderson. NS-3 Project Goals. Talk given during the "Workshop on NS-2: The IP Network Simulator". http://www.wns2.org/slides/henderson.pdf.

[15] H. Karl and A. Willig. *Protocols and Architecture for Wireless Sensor Networks*. John Wiley and sons, 2005.

[16] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

[17] R. Ontko and A. Reeder. http://www.ontko.com/moss/.

[18] L. Palopoli, G. Lipari, G. Lamastra, L. Abeni, G. Bolognini, and P. Ancilotti. An object oriented tool for simulating distributed real-time control systems. *Software: Practice and Experience*, 2002.

[19] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9), September 1990.

[20] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1989.

[21] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[22] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard-real-time environments. *IEEE Transactions on Computers*, 4(1), January 1995.

[23] J. Zheng and M. J. Lee. A comprehensive performance study of ieee 802.15.4. In *Sensor Network Operations*, pages 218–237. IEEE Press, Wiley Interscience, 2006.