RTNS: an NS-2 extension to Simulate Wireless Real-Time Distributed Systems for Structured Topologies

Paolo Pagano
Scuola Superiore Sant'Anna
56127 Pisa, Italy
p.pagano@sssup.it

Mangesh Chitnis
Scuola Superiore Sant'Anna
56127 Pisa, Italy
m.chitnis@sssup.it

Giuseppe Lipari
Scuola Superiore Sant'Anna
56127 Pisa, Italy
g.lipari@sssup.it

ABSTRACT

Wireless Sensor Networks are now being considered for use in industrial automation and process control. These applications present different characteristics with respect to classical WSN application domains. In particular, the nodes may have high computational load due to the high sampling frequencies; moreover, they present real-time constraints, as data must be processed and transmitted with bounded delay.

In this paper, we present RTNS, a simulator for distributed real-time systems that allows to model and simulate the temporal behavior of network protocols, real-time Operating System and distributed applications. The tool has been developed as a plug-in extension of the popular NS-2 simulator, hence it is possible to reuse most of the packages already available for NS-2. The aspects related to real-time Operating System, the overhead of interrupt handlers and protocol management, and the set of concurrent tasks executing on each node, are modeled using the RTSim simulator. With respect to a previously documented version, the package now has an extended scope and can model complex multi-hop scenarios.

After presenting the simulator structure, we show how the tool can be used to model and simulate realistic WSN scenarios. Hereby, three examples are presented with the aim of showing how possible failures in the nodes or a load suddenly appearing in gateways connecting neighbor clusters for structured topologies can cause a worsening in the end-to-end transmission delays. We show that the adoption of a real-time Operating System in the nodes along with a proper scheduling policy for tasks can avoid (or at least keep under control) unpredictable effects in end-to-end delay.

1. INTRODUCTION

The development of wireless sensor networks (WSN) was originally motivated by military applications. However, WSN are now being used in many civilian applications, including environment and habitat monitoring, healthcare applications, agricultural management and control. Usually, these applications do not require tight real-time constraints. The typical frequencies at which data needs to be sampled and processed are quite low, below 1 Hz. As a consequence, the amount of computational load in each node, and the amount of message transmissions, is very small. For these applications, much of the research has been focused on reducing the power consumptions of the systems, by putting nodes in sleep-mode for the longest possible interval of time.

Recently, WSNs are being considered for use in different application domains. A very promising application of WSN is for industrial automation and process control [18, 1, 11]. One urgent need of industrial automation systems that motivates the use of WSNs is to reduce the amount of physical wires in an industrial plant. Reducing cables can significantly reduce the cost for building and maintaining the system. Moreover, with less cables it is easier to dynamically re-configure the machines, and implement plug-and-play components. An example of research effort in this sense is the RI-MACS project [15], whose goal is to increase reconfigurability and adaptiveness of industrial automation platforms by using WSNs in selected cases.

However, the requirements of these systems are substantially different from those of other domains of WSN. In addition to requirements for increased robustness and fault tolerance, each node is expected to perform a substantial amount of computation in real-time. First of all, the rate at which data must be sampled is quite high (up to hundreds of Hz). Second, several 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. Late sensor messages may not be considered acceptable. Many activities, like sampling and actuation, must be triggered periodically and executed with bounded response time, otherwise the system may not work properly. A real-time schedulability analysis 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 WSN has not been deeply investigated until now. TinyOS [19], the most popular Operating System for WSN, does not support predictable real-time scheduling mechanisms although recent works[3] have imported preemption in scheduling mechanisms. At the network level, mechanisms for real-time communication and quality of service control (like the Collision Free Period access included in IEEE 802.15.4 standard) are not implemented or not enabled. 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 important problem is the lack of simulation tools that can model not only the functional but also the temporal behavior of system, both at the network level and at the node level. Simulation plays and important role both for research and for industrial

WICON 2007, October 22-24, 2007, Austin, Texas, USA

practice. In academic research, simulation 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 assessment of performance, for system dimensioning, and to identify potential bottlenecks and problems.

In this paper we propose the *Real Time Network Simulator* (RTNS), a tool that addresses precisely this problem: how to properly simulate the temporal behavior of the application, taking into account not only the delay due to network, but also the delay due to the Operating System overhead and the computational load in the node.

Unfortunately, previous to our proposal, no simulator allowed to model at the same time the temporal behavior of the application tasks and the real-time Operating System in the node, and the temporal behavior of the network protocols. In most simulators, it was possible to analyze the effect of network protocols on message delays (see for example [4, 12] for an overview of the state of the art on simulation tools). However, for the kind of applications we consider in this paper (industrial automation and process control), the effects of the Operating System scheduler, and the overhead due to the protocol stack and due to the interrupts, cannot be neglected.

Also, such effects are highly non-linear, and cannot be easily approximated by simple equations. For example, according to the holistic analysis [14, 5], the worst-case end-to-end delay of a chain of task can be upper-bounded by a complex recursive set of equations. Moreover, in most cases actual worst-case delays are much lower than the estimated upper bound.

The effect that the task scheduling algorithm in the node has on the end-to-end delay of messages has been shown in [12]. The paper compares the FCFS scheduling policy (used by TinyOS 1.x) against fixed priority scheduling used by most Real-Time Operating Systems (RTOSs).

1.1 Contributions of this paper

The proposed simulation tool allows to model and simulate:

- RTOS and scheduling algorithms;
- applications in terms of concurrent tasks executing in each node;
- messages, protocols, and transmission delays over the network.

The tool is a combination of NS-2 [9, 7] and RTSim [17] simulators. NS-2 is a popular simulation framework for simulating network protocols, both for wired and wireless networks. RTSim is a simulation tool for RTOSs.

The RTNS co-simulator allows to simulate the networking aspects via NS-2 as well as the real-time Operating Systems aspects via RTSim.

A share ranging from 40% to 70% [7] (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.

In the Operating System area, there is not such a widely used simulation package as NS-2. RTSim [13] is a software package written in C++ for the simulation of real-time Operating Systems, available as open source [17]. It includes support for many real-time scheduling 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.

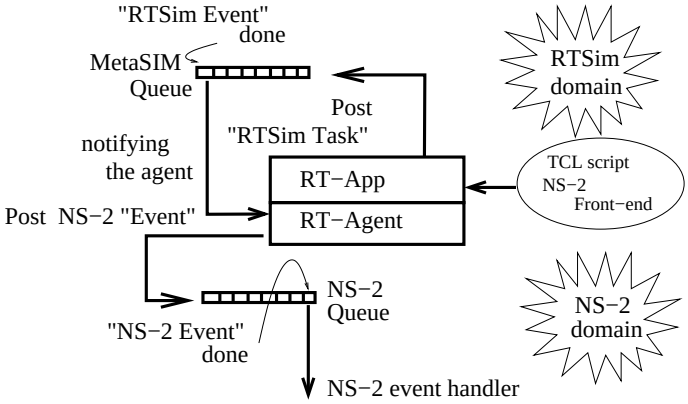

Figure 1: RTNS **co-simulator architecture.**

By combining these two, we could *re-use* the huge amount of existing work on NS-2 and the capabilities of RTSim. With respect to an earlier work [12], we improved the tool by adding the capability to model multi-hop networks and end-to-end message delays. In particular, we can now model the overhead of the network protocols, and specifically of the routing protocol. This extension requires severe modifications in the NS-2 core libraries, as explained in Section 2.2.

The execution time overhead due to the inclusion of OS mechanisms is adding an order of magnitude in execution time if compared with ordinary NS-2 simulations. We proved the scalability property up to 2500 nodes, remarkable especially for highly dense WSNs.

In this paper we show two interesting applications of our tool. By analyzing the results of the simulations, we show that:

- the use of a non-real-time scheduling policy in the nodes can have a tremendous impact on the end-to-end delay of a message transmission in a multi-hop network;
- the effect of an overloaded (or failed) node in the network on the routing protocol depends on the priority at which the routing protocol stack is executed in the RTOS;
- the priority of the different activities in the nodes must be carefully calibrated in order to fulfill the real-time constraints of the application.

2. RTNS

RTNS is obtained linking together NS-2 and RTSim at configure time. The following sections describe the RTNS architecture and the modifications brought to the NS-2 core classes.

2.1 Co-simulator Architecture

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

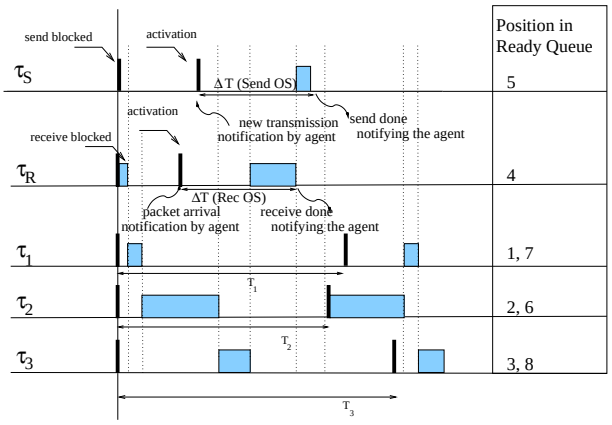

Figure 2: FCFS scheduling of Send, Receive and Dummy Tasks.

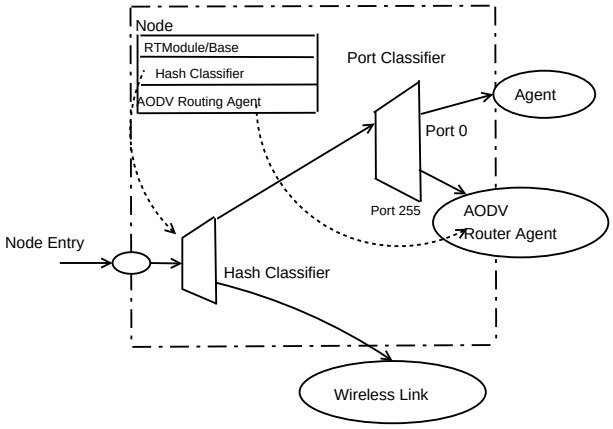

Figure 3: Architecture of a wireless node in NS-2.

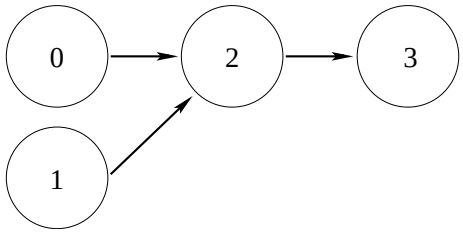

Figure 4: Topology for a wireless scenario consisting of two clusters connected by a gateway station.

all the features of a Real-Time Kernel (see Figure 1). Whenever it's required from the TCL script, the simulator creates an instance of RT-App which is attached to an NS-2 Agent (RT-Agent) to send and receive data packets. The RT-App creates aperiodic tasks for "Send", "Receive", and for "Routing" purposes (see Section 2.2) and schedules them adopting one of the policies implemented in RTSim (as FCFS, EDF, FP, etc.).

These tasks get blocked as soon as they are spawned and remain in the "blocked" state waiting for an event corresponding to a send and receive of a data packet. Whenever they are activated by means of external interrupts they enter the ready queue together with other existing computational (dummy) tasks to create a tunable load within the CPU. The Operating System puts these tasks into the "running" state following the adopted 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 as shown in Figure 2. In case of a priority-based scheduler 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 and the delays are strongly dependent on the load of the node.

In Figure 2 the timing behavior of running tasks scheduled by FCFS policy is shown. The activation order corresponds to the position in the queue of ready tasks (reported in a table at the right side of the figure), thus to the order of execution.

The corresponding case adopting FP scheduling would depend on the priorities assigned to each type of task (Send, Receive, Routing, and Dummy). In our case studies, we will assign to Network tasks (Send, Receive, and Routing) higher priority with respect to the others. This solution works only as matter of example (of a case limit) since in the real world there may be computational tasks more important than networking and a finer tuning to set the priorities is required.

2.2 Modification to Node Structure

As already mentioned in Section 2.1 the NS-2 RT-App class abstracts the operations done by the Operating System. However, in a wireless multi-hop scenario, the intermediate node, in the act of routing a packet, fails to acknowledge the presence of RT-App hence neglecting any delay caused by the concurrent processing of other tasks by the CPU. In this section we will address the problem of RT-App invisibility in intermediate nodes and propose a modification to the NS-2 architecture to solve the problem. The solution described in this paper will help to simulate a wireless multi-hop scenario, which is typically used in WSNs and provide an accurate measure of delay of the packets.

The typical structure of a unicast node in NS-2 is shown in Figure 3. A node has a unique network address and contains an entry point, an address classifier and a port classifier, a set of agents, and a Routing Module.

On reception of a packet, the node examines the destination address field in its header and takes action on its basis.

The possible values of the destination address are mapped to an outgoing interface object to reach the next downstream recipient of the packet. In NS-2, this task is performed by a simple *classifier* object. A classifier provides a way to match a packet against some logical criteria and retrieves a reference to another simulation object based on the matched results. Each classifier contains a table of simulation objects indexed by slot number. The job of a classifier is to determine the slot number associated with a received packet and forward that packet to the object referenced by that particular slot.

Every implementation of the *Routing Module* contained in a wireless node consists of a *Routing Agent*, in charge of exchanging routing packets with neighbors, and a collection of rules, often called *Routing Logic*, used to calculate the actual routes from the information gathered by the agent. To perform packet forwarding the node makes use of the *Routing Tables*.

The default setting for a wireless node adopts the *Base Routing Module* provided by the NS-2 package and uses a Hash Classifier to decide on packet forwarding within the node. Here we describe the working of the wireless node using Hash Classifier and highlight the problem of multi-hop described earlier.

```
r 5.004704067 _2_ MAC  --- 0 AODV 48
                           [0 ffffffff 0 800] --
r 5.004729067 _2_ RTR  --- 0 AODV 48
                           [0 ffffffff 0 800] --
s 5.011660641 _2_ RTR  --- 0 AODV 48
                           [0 ffffffff 0 800] --
s 5.014592000 _2_ MAC  --- 0 AODV 55
                           [0 ffffffff 2 800] --
-----
-----
r 5.057504000 _2_ MAC  --- 0 RT 90
                           [0 2 0 800] --
r 5.057529000 _2_ RTR  --- 0 RT 90
                           [0 2 0 800] --
f 5.057529000 _2_ RTR  --- 0 RT 90
                           [0 2 0 800] --
```

Figure 5: Trace file depicting NS-2 default behavior using Hash classifier.

Consider the wireless scenario as shown in Figure 4 (the same will be considered in a case study in Section 3.2). It consists of 4 wireless nodes, each adopting a Base Routing Module. Node 0 and Node 1 are the source nodes and Node 3 is the sink Node. Node 2 is used as a gateway to route packets to the destination. We use part of the NS-2 trace file as shown in Figure 5 to elaborate on the working of the classifier to forward the packets within the node; an example of a trace for a tcp packet is as follows:

```
r 50.093884945 _6_ RTR  --- 3 udp 170
                            [b2 4 6 800] - [...]
```

Here we see a "udp data packet" being received by a node with id of 6 at time 50.093884945. The unique identifier of this packet is 3 with a common header size of 170. The MAC details shows an IP packet (ETHERTYPE_IP is defined as 0x0800), MAC-id of this receiving node is 4. That of the sending node is 6 and expected time to send this data packet over the wireless channel is b2 (hex2dec conversion: 176+2 sec). The rightmost part of the line of the trace file contains information specific to the module producing the trace file and are not commented because they are not relevant within this context. The format of the trace file is extensively explained in section 16.1.6 of the NS-2 manual [10].

As shown in the trace file, we see that when a packet is received at Node 2 the MAC layer forwards it to RTR (the *Routing Agent*), AODV in this example. RTR is responsible for deciding the outgoing link. The decision to forward the packet from MAC Layer to RTR is done by the classifier. The classifier is unable to interpret the presence of RT-App which is assigned to a well defined port (port # 0 in our implementation) operated by a single agent (RT-Agent). The classifier forwards the packet to the actual recipient, the Routing Agent on port # 255.

Because of this behavior the simulator is unable to address the delay induced by the processing load present in the Node. This delay is quite significant and models the actual processing behavior of a Wireless Sensor Node.

We propose a new classifier which will handle the forwarding of packets in a way that would consider the presence of RT-App and thus take into effect the Operating System behavior in packet forwarding.

As shown in Figure 6 we introduce a new Module called WSN-RoutingModule. This Module is used to incorporate the classifier (RTMobile) which handles the flow of Routing and Data Packets exchanged between the NS-2 agents. RTMobile Classifier extends

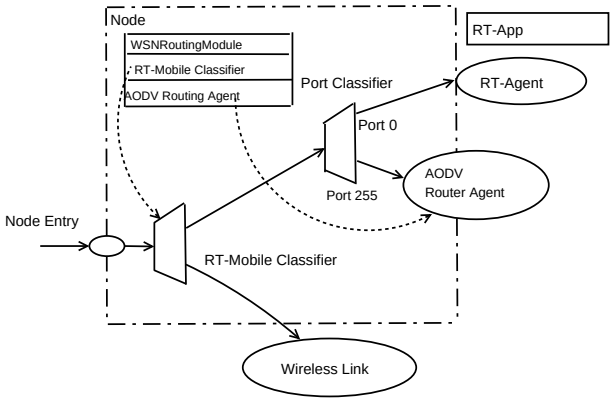

Figure 6: Modified architecture of Wireless Node in NS-2.

```
if (destinationPort = AODV_Agent AND
    destAddr <> NodeAddr AND
    sourceAddr <> NodeAddr){
                          #check if I'm
                          #an intermediate node
    if packetArrived = false;
                          #if RTApp hasn't got it
       destinationPort = Port_RTAgent;
    else
       packetArrived = true;
    forward(packet);      #follow the normal logic
}
```

Figure 7: Pseudocode representing packet flow control logic within RTMobile Classifier.

the functionality of Hash Classifier.

The pseudo-code in Figure 7 explains the logic used by the classifier. Whenever the classifier of a node receives a packet, it checks if it is an intermediate hop along the path, and examines the type of packet (e.g. it might be AODV or some kind of data packets as "RT" in our example).

If the received packet is of type AODV, the classifier changes the port in the packet header to that of RT-Agent for a first "default" classification. In this way the routing packets are examined (following the scheduling policy decided for the node) by the RT-App which notifies the RT-Agent on job done.

Whenever RT-Agent gets this notification it will forward the packet back to the classifier (which in turn will let the packet follow the normal iter), thus introducing delay in packet transmission.

The introduction of RTMobile Classifier makes the simulator aware of the presence of RT-App in each node even for those types of packets (as the routing ones) which are not explicitly addressed to that port. The appropriate task running in the node performs the job of examining the packets and re-transmitting them to the next downstream recipient. This introduces some delay depending on the load present in the node as opposed to the default implementation of NS-2 in which receive and forward of routing messages is instantaneously performed by the *Routing Agent*.

The flow of packets reported in Figure 8 for the case of a RT-Mobile classifier corresponds to the same application as in Figure 5.

The trace file shows the introduction of RT-Agent in the packet flow within the intermediate node: before sending the packet to RTR (i.e AODV Agent), the packet resides for some time within the agent AGT (i.e RT-Agent). As it can be seen from the trace

```
r 5.004704067 _2_ MAC  --- 0 AODV 48
                           [0 ffffffff 0 800] --
r 5.004729067 _2_ AGT  --- 0 AODV 48
                           [0 ffffffff 0 800] --
s 5.181500000 _2_ AGT  --- 0 AODV 48
                           [0 ffffffff 0 800] --
r 5.181500000 _2_ RTR  --- 0 AODV 48
                           [0 ffffffff 0 800] --
s 5.185636857 _2_ RTR  --- 0 AODV 48
                           [0 ffffffff 0 800] --
s 5.187392000 _2_ MAC  --- 0 AODV 55
                           [0 ffffffff 2 800] --
-----
-----
r 5.331744000 _2_ MAC  --- 0 RT 90
                           [0 2 0 800] --
r 5.331769000 _2_ AGT  --- 0 RT 90
                           [0 2 0 800] --
s 5.331800000 _2_ AGT  --- 0 RT 90
                           [0 2 0 800] --
r 5.331800000 _2_ RTR  --- 0 RT 90
                           [0 2 0 800] --
f 5.331800000 _2_ RTR  --- 0 RT 90
                           [0 2 0 800] --
s 5.336512000 _2_ MAC  --- 0 RT 97
                           [0 3 2 800] --
```

Figure 8: Trace file depicting NS-2 behavior using RTMobile classifier.

file there is an additional introduction of delay between MAC layer and RTR layer in case of Data (of type "RT") as well as for routing packets (of type "AODV").

3. SIMULATION ACTIVITY

In this section, we describe the latency induced by the presence of localized and diffused load in selected Wireless Personal Area Network (WPAN) topologies. Moreover we present the methodology we adopted to gather statistical data in order to organize the simulation results.

We will describe three different case studies of WPAN activities. In each of them we vary some parameters such as load, task features and kernel scheduling policies affecting the end to end delay of the packets.

3.1 Methodology

In each of the case studies a base TCL script is used to generate the network scenario, construct the wireless nodes with the WSN-RoutingModule enabled, and start the data traffic. The application models CBR traffic between one or more source nodes and a single destination in the network. The traffic generators run at a frequency of 5 Hz injecting into the network, packets with payload of 70 bytes.

A set of simulation runs are spawned under different conditions by providing an appropriate value for the following parameters:

- LOAD, to set the simulated CPU load in a node;
- SCHED, to adopt a scheduling policy used in RT-App. In our simulation it can take the value of FCFS (First Come First Served) or FP (Fixed Priority).

To achieve good statistical significance and to properly use the pseudo-random number generation, for each combination, the final value for the selected metrics are estimated averaging over 10 different pairs of seeds (RTSIM-SEED/NS-SEED) provided to NS-2 and RTSim engines. The simulation results under each condition are computed on the basis of 20,000 transmitted packets.

To log the simulation results and generate histograms we used the ROOT [2] Data Analysis Platform developed at CERN. The results provided the information on the total time required for packets to reach the destination. This time depends on the overhead incurred at the source node to send the packet (ΔT_{send}), the network transmission time ($\sum_i \Delta T^i_{routing} + \Delta T^i_{propagation}$), and the overhead incurred at the sink to receive the packet (ΔT_{rec}):

$$\Delta T = \Delta T_{send} + \sum_i \Delta T^i_{routing} + \Delta T_{rec}\,, \qquad (1)$$

where the summation is done upon the intermediate nodes.

The network transmission time is dependent on the load in the intermediate nodes. Any delay in forwarding the packet by intermediate nodes is seen by the destination node as a congestion in the network.

The error for the time overhead is calculated as the variance of the sample obtained in identical simulation conditions (i.e. varying the seeds pair only):

$$s^2(\Delta T) = \frac{1}{9}\sum (\Delta T_i - \langle \Delta T \rangle)^2 \qquad (2)$$

For each simulated transmission we estimate maximum, minimum and mean delays of packet transmissions. The simulation adopts the NS-2 TwoRayGround wave propagation model.

3.2 Simulation 1: Fetch and Forward

In this section we refer to a WSN consisting of four wireless nodes placed as in Figure 4. For Physical and MAC layers we adopted the default WPAN settings of NS-2, namely an order 3, beacon-enabled superframe without Granted Time Slots (GTS).

The topology has been kept as simple as possible to easily extract communication patterns and calculate the most relevant metrics. Obviously the scenario in Figure 4 can be complicated as needed in order to accommodate hundreds of nodes interconnected in tens of clusters.

The communication range is such that Nodes 0, 1, and 2 are connected through single hop paths; Node 2 (acting as PAN coordinator) can hear every other node including Node 3. The network is thus composed by two interconnected clusters, Node 2 being the gateway between them.

We imagine that, for cost minimization in industrial deployments, the number of stations is kept as low as possible and Node 2 is involved in ordinary computational tasks apart from the activity descending from the role of gateway.

The simulated distributed application consists in transporting sensor data from the left to the right cluster: Nodes 0 and 1 periodically send packets to the sink placed in Node 3.

The simulation makes use of AODV routing algorithm to find a multi-hop path from source to destination and route the packets along that.

AODV does not embed any real-time feature; nevertheless, due to the lack of more appropriate protocols (e.g. SPEED[6] and RAP[8]) in NS-2 following the same line as in the examples provided in the NS-2 802.15.4 library, we decided to adopt it for its light overhead enforcing the compatibility with WSN. The need of improving AODV by means of real-time added features is commented in section 3.4.

RTNS keeps trace of every single transmission in the hypothesis that the nodes must respond to some parameters of QoS. The transmission time is ordinarily simulated by NS-2 and has a random behavior as shown in the top plot of Figure 9. In case of presence of a

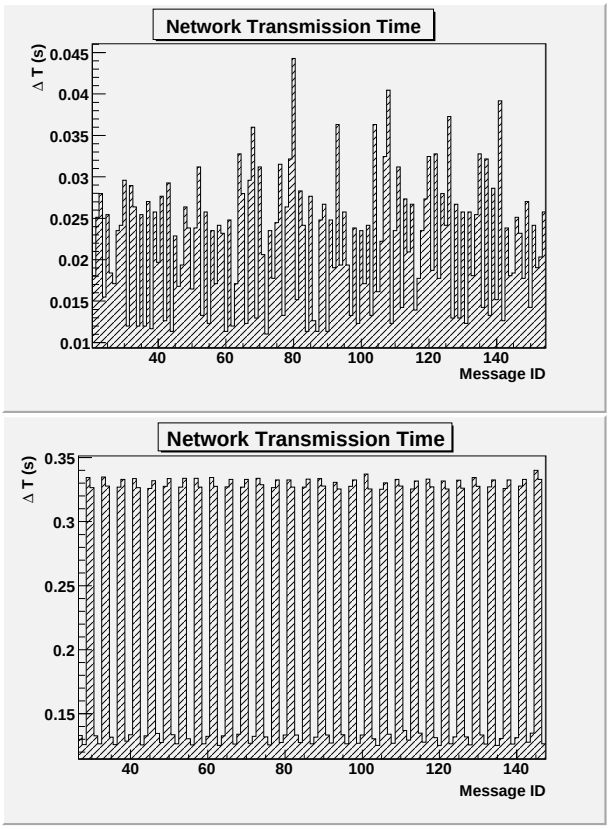

Figure 9: Network transmission time as function of the message ID for the cases of gateway without computational load (top plot) and with a load of 80% (bottom plot). The Y scales differ from plot to plot.

certain load in the gateway, if the kernel adopts a FCFS scheduling policy, a regular pattern in Network Transmission Time shows up as in the bottom plot of Figure 9.

The oscillatory behavior descends from the random time of packet collection at the gateway: if the packet reaches the gateway when its CPU is booked, it will be routed to the sink after the end of the running job and of all the jobs eventually activated at the sink before its arrival. The more the gateway is loaded, the larger is the percentage of the messages forwarded after some delay.

The graph in Figure 10 shows the mean delays in network transmission time as a function of the load in the intermediate node using FCFS and FP scheduling policies. Looking at the graph, as the load in intermediate node increases the network latency increases in case of FCFS. However in case of FP although the load increases, the network latency remains constant. We can't show the results concerning the maximum delays because in this scenario there are two nodes requiring access to the shared medium: the occasional collisions occurring due to these concurrent activities let the maximum delay metric behave in a non-deterministic way thus excluding it from any reliable analysis.

We don't extensively comment upon the latency induced by the Operating System at source (Nodes 0 and 1) and destination (Node 3) because these effects have already been deeply investigated in a previous work [12]. For matter of completeness the delays related to message reception at the sink are reported in Figure 11 and a similar behavior is found at the source nodes.

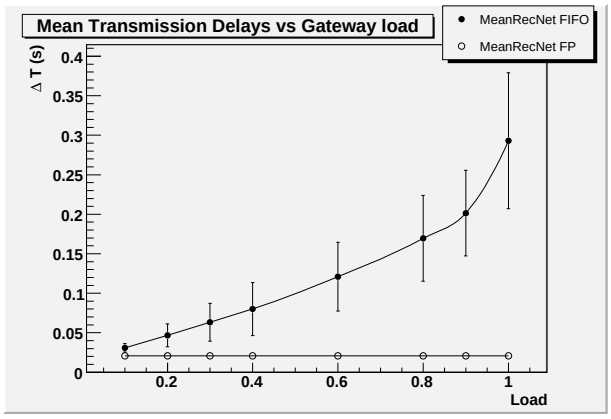

Figure 10: The mean delays in message elaboration due to the CPU activity as a function of the utilization factor of the gateway for FCFS and FP scheduling policies.

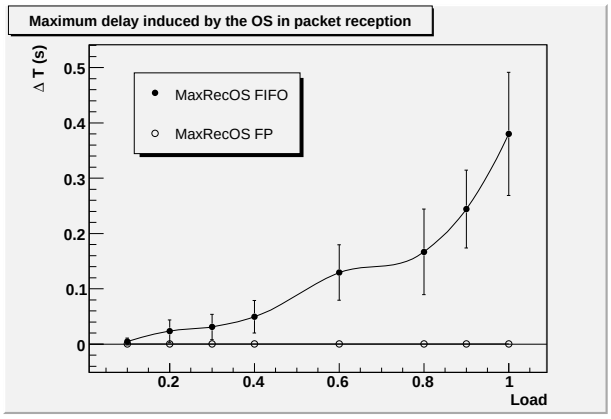

Figure 11: The maximum delays induced by the load at the sink for FCFS and FP scheduling policies.

3.3 Simulation 2: Beacon Enabled Tree

This simulation experiment is an example coming along with the WPAN implementation of NS-2 (*wpan_demo3*). The scenario is structured as a tree topology composed by 11 nodes as shown in Figure 12. This scenario consists of one PAN coordinator, 5 coordinators and 5 devices. The simulation works in a beacon enabled mode with a Beacon Order of 3.

We imagine that in the RI-MACS envisioning of an assembly zone, actuators connected to wireless sensors may need continuous tuning of operational parameters in a tree-shaped network: nodes placed closer to the top of the tree may want to send corrected parameters to the nodes placed on the leaves.

We thus simulate Node 1 as the source and Node 10 the destination. The routing algorithm is AODV, the routing path is sketched as a continuous line in the figure, and the distributed application is modeled as a CBR traffic generator with a constant load distributed over the network.

The methodology used to experiment with this simulation is the same as explained in the previous case. However the results here consider the presence of PAN coordinator and Beacon mode operation.

The graph in figure 13 is a comparison of the end to end maximum and mean delays in packets for Beacon enabled Tree as a

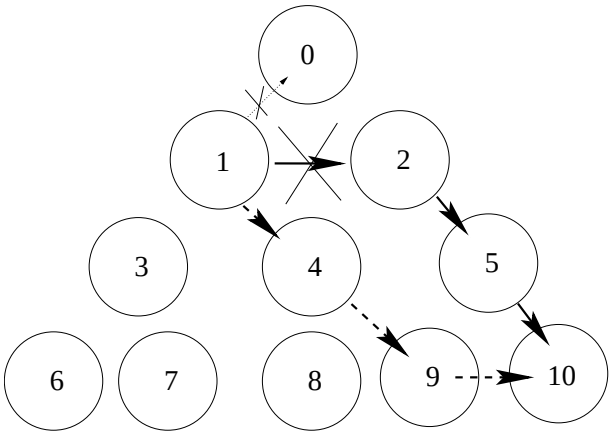

Figure 12: Network topology used in simulation 2 and 3. The original and the alternate routing paths used by AODV protocol are shown on the plot.

function of the network load in the nodes.

3.4 Simulation 3: Effect due to node failure

In the third and last experiment, we used the same simulation settings as in Section 3.3. However, here we compared the effect of node failure on the network latency. Because of the roles of local PAN coordinators in the intermediate branches, the beacon mode has been disabled to avoid the network crash due to the absence of working routes connecting the coordinator to ordinary devices; for the rest, the simulation proceeds in a similar way as in Section 3.3.

The end-to-end transmission time is influenced by the number of hops in the two routing paths followed by the packets before and after the link between node 1 and nodes 0 and 2 break down (see Figure 12).

In Figure 14 we show how with FCFS scheduling policy, the rise in end to end transmission delay is strongly dependent on the load the packet finds along the alternate route. The spike connecting the two portions of the histogram is due to path discovery algorithm used by AODV to find an alternate path. A superimposition of histograms related to a *load-jump* of 0.5, 0.6, 0.9, 0.99 is shown.

A failure in the node may cause a worsening in the parameter of QoS during the execution of a distributed application which is not expected and strongly dependent on the instantaneous load conditions of the network. Of course, as usual, setting the priority for routing packets higher than the other computational tasks lets this rise be independent from the network conditions.

If it's not possible to act on the kernel policies, it is probably possible to customize the routing protocol.

AODV-Aware is a modified implementation of AODV routing protocol being developed at our institute. The modified protocol builds the routes up in order to minimize the end to end delay suffered in packet transmission in multi-hop scenarios. In order to take into account the computational load in the network, the protocol makes use of such information stored in a distributed DB available through a middleware service. By means of AODV-Aware, we can minimize the step-like behavior induced by the failure and shown in Figure 14, by selecting the path with minimum cumulated load in the intermediate nodes. AODV-Aware will be distributed with the forthcoming releases of RTNS.

3.5 Comments

The case studies we presented in previous subsections try to

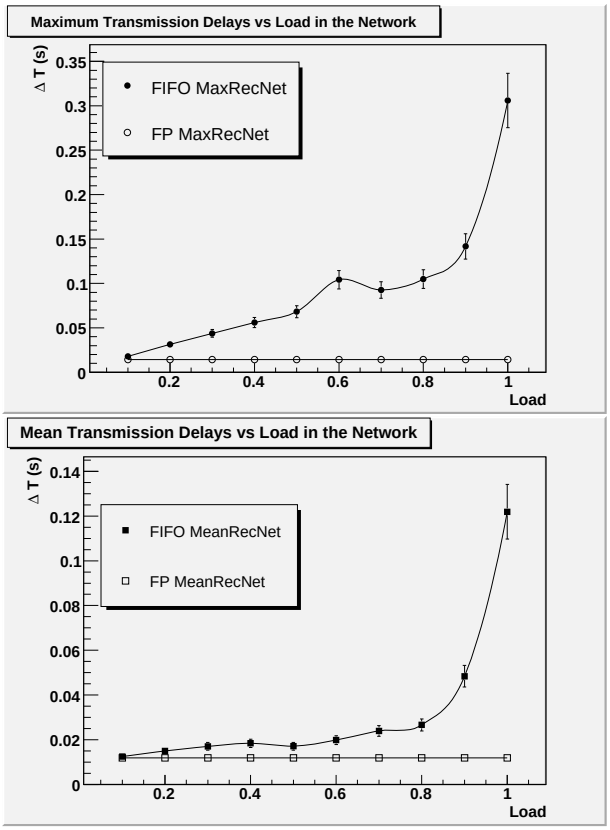

Figure 13: Dependency of Network Latency (maximum in the plot at top, mean in the plot at bottom) on load values.

replicate the real world application behavior considering various topologies compliant with IEEE 802.15.4 standard and different operational modes (beacon enabled, peer to peer).

Our simulation results give an accurate measure of these effects and provide the means for application and network analysis. In each of these case studies we see that even a small amount of processing load within the node can lead to an unpredictable behavior in terms of data transmission. Looking at maximum and mean delays in Network transmissions induced by the Operating System running on the wireless nodes, these are found to increase as a function of the load in the network.

Although our analysis shows that using a FP scheduling for packet transmission limits the network delay, this might not always be desirable. In the first case study, we showed a scenario where communication among the nodes makes use of a gateway between two clusters. In such application the load within the gateway strongly depends on the amount of traffic it handles. As the traffic increases the computational load due to packet forwarding increases as well.

If the routing task is given higher priority, as done in our case study, the gateway node might miss deadlines for the tasks associated to data processing.

Setting the task priorities is a crucial design activity and requires a trade off between the tolerance in network transmission delays and the predictability for the computational tasks running on the sensor nodes.

4. THE PUBLISHED RELEASE

RTNS can be downloaded from the homepage at [16].

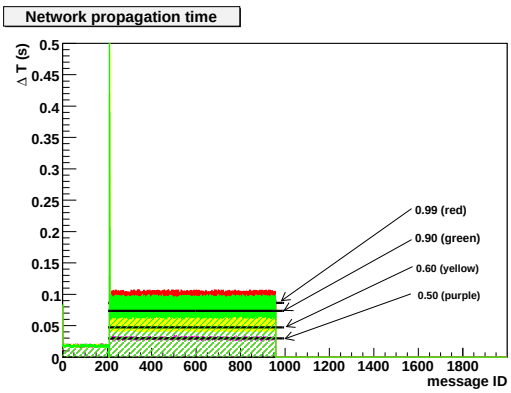

Figure 14: Temporal display of breakdown effect on network transmission time. The superimposed plots refer to different cumulative loads in the AODV alternate path.

The package contains the following components:

- NS-2 version 2.29 (slightly modified);
- RTNS realtime package in NS-2;
- RTSim version 1.0 (slightly modified);
- ROOT version 5.11 (for Cygwin) or 5.12 (for Linux).

The web version also includes all case studies described in this paper along with the scripts useful to generate the commented results.

5. CONCLUSIONS AND OUTLOOK

In this paper we presented a software package simulating Operating System aspects in wireless telecommunication. The tool is particularly suited to model wireless distributed real-time applications. The extension with respect to a previous release deals mainly with modeling of protocol stack overhead in the real-time Operating System running on the node. Therefore, we can now realistically evaluate the impact that a computational load in the node has on the end-to-end delay of critical messages in multi-hop networks with a structured topology.

As extensively explained in section 3, the packet delivery is affected by the computing load along the route if the routing task (activated at the packet reception) is scheduled following a FCFS policy as it happens within the popular TinyOS 1.x software environment. A possible solution is to assign to routing a higher priority with respect to the computational tasks and to adopt a FP scheduling policy (not implemented in TinyOS). This solution (as commented in section 3.5) is not always applicable and an effective solution has to be found from case to case.

The package performing these simulations is organized as a plug-in of the popular NS-2 under the name of Real-Time Network Simulator (RTNS). We decided to freely distribute the code as explained in Section 4.

We are now progressing on AODV-Aware, the routing algorithm extending AODV protocol to real-time functionalities.

In the next future, we plan to run an extensive comparison between data collected from real experiments and simulations. RTNS will cover a key role for testing new communication protocols and designing distributed systems especially for large and complex systems with hundreds of nodes.

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

6. REFERENCES

[1] N. Aakvaag, M. Mathiesen, and G. Thonet. Timing and power issues in wireless sensor networks, an industrial test case. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 2005.

[2] R. Brun and F. Rademakers. ROOT. An Object Oriented Data Analysis Framework. `http://root.cern.ch`.

[3] C. Duffy, U. Roedig, J. Herbert, and C. J. Sreenan. Adding Preemption to TinyOS. In *Proceedings of the The Fourth Workshop on Embedded Networked Sensors (EmNets2007), Cork, Ireland*. ACM Press, June 2007.

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

[5] J. Gutierrez and M. Harbour. Schedulability analysis for tasks with static and dynamic offsets. *Proc. of RTSS*, 1998.

[6] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher. SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks. In *ICDCS*, May 2003.

[7] 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`.

[8] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. In *IEEE Real Time Technology and Applications Symposium*, June 2002.

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

[10] The ns Manual (formerly known as ns Notes and Documentation). `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

[11] N. Ota and P. Wright. Trends in wireless sensor networks for manufacturing. *International Journal of Manufacturing Research*, 1(1):3–17, 2006.

[12] P. Pagano, P. Batra, and G. Lipari. A Framework for Modeling Operating System Mechanisms in the Simulation of Network Protocols for Real-Time Distributed Systems. In *"Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)"*, Long Beach (CA), USA, 2007.

[13] 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.

[14] R. Pellizzoni, G. Lipari, S. Anna, and I. Pisa. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 66–75, 2005.

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

[16] The RTNS simulation suite. `http://rtns.sssup.it`.

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

[18] X. Shen and Y. Sun. Wireless sensor networks for industrial control. In *Proceedings of the 5th IEEE World Congress on Intelligent Control And Automation*, Hangzhou, P.R. China, June 2004. IEEE.

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