Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2010, Article ID 107946, 19 pages
doi:10.1155/2010/107946

Research Article

Simulating Real-Time Aspects of Wireless Sensor Networks

Paolo Pagano,[1] Mangesh Chitnis,[1] Giuseppe Lipari,[1] Christian Nastasi,[1] and Yao Liang[2]

[1] *Real-Time Systems Laboratory (RETIS Lab), Scuola Superiore Sant'Anna, Via G. Moruzzi, 1, 56124 Pisa, Italy*
[2] *Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, 723 W. Michigan Street SL 280, Indianapolis, IN 46202-5132, USA*

Correspondence should be addressed to Paolo Pagano, p.pagano@sssup.it

Received 10 June 2009; Accepted 10 November 2009

Academic Editor: Arnd-Ragnar Rhiemeier

Wireless Sensor Networks (WSNs) technology has been mainly used in the applications with low-frequency sampling and little computational complexity. Recently, new classes of WSN-based applications with different characteristics are being considered, including process control, industrial automation and visual surveillance. Such new applications usually involve relatively heavy computations and also present real-time requirements as bounded end-to- end delay and guaranteed Quality of Service. It becomes then necessary to employ proper resource management policies, not only for communication resources but also jointly for computing resources, in the design and development of such WSN-based applications. In this context, simulation can play a critical role, together with analytical models, for validating a system design against the parameters of Quality of Service demanded for. In this paper, we present RTNS, a publicly available free simulation tool which includes Operating System aspects in wireless distributed applications. RTNS extends the well-known NS-2 simulator with models of the CPU, the Real-Time Operating System and the application tasks, to take into account delays due to the computation in addition to the communication. We demonstrate the benefits of RTNS by presenting our simulation study for a complex WSN-based multi-view vision system for real-time event detection.

1. Introduction

Wireless Sensor Networks (WSNs) were initially proposed in domains where ordinary networks (not necessarily wired) are not convenient, either because of the missing infrastructures, or when numerous nodes (in the order of hundreds) are needed to achieve the assigned task. Examples of such domains are military applications (like quickly monitoring a large un-known area) and environmental monitoring.

The main stream of the research on these early WSN systems has focused on MAC and routing protocol optimization to minimize the communication delay and the energy consumption to improve on the lifetime of the devices between battery replacements. In fact, in these systems energy consumption is generally dominated by communication tasks. On the other hand, research topics related to computing resource management and CPU scheduling have been marginally discussed, as these systems do not present tight real-time requirements and the computational load is usually negligible compared with typical communication delays.

Today, "second generation WSNs" are being considered for industrial automation (e.g., process control in assembly areas), multimedia (e.g., telemedicine, intrusion detection systems), health care (e.g., emergency protocols, disaster response, and stroke patient rehabilitation), and so forth. As these new WSN-based applications usually involve relatively heavier computations, they can be characterized as WSN-based distributed computing systems and take the advantage of the fact that richer devices embedding 16 and 32-bits CPUs and larger sets of programmable and dynamic memories to support more complex applications are available today on the market. Some notable on-going projects in this category are listed below:

(i) SENSE [1]—Smart Embedded Network of Sensing Entities;

(ii) CoBIs [2]—Collaborative Business Items;

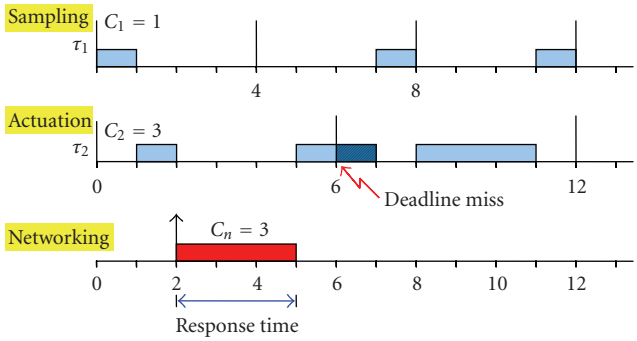


Figure 1: Activities, like sampling and actuation, are triggered periodically and executed within bounded delays. An event-triggered activity may delay them causing a system malfunction.

(iii) WINSOC [3]—Wireless Sensor Network with Self-Organization Capabilities for critical and emergency applications;

(iv) SCIER [4]—A technological simulation platform to managing natural hazards;

(v) SensAction-AAL [5]—SENSing and ACTION to support mobility in Ambient Assisted Living.

The profile of these WSN-based distributed systems is substantially different from those in 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 such that related to data filtering, actuation, diagnostic, logging, communication, and so forth. 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: late sensor messages may not be considered acceptable. Many activities, like sampling and actuation, must be triggered periodically and executed within bounded delays, otherwise the system may not work properly (see Figure 1). Typically, a robust Real-Time Operating System (RTOS) providing customizable scheduling policies (for multitasking operation) and reliable services, ranging from networking to peripheral management, is needed in such applications.

While an off-line schedulability analysis should be performed to guarantee that all timing constraints will be respected and that the system will work properly even under worst-case load conditions, simulation plays a crucial role in this regard, especially for large and complex systems with hundreds or thousands of nodes, where off-line analysis might be impractical. In addition, simulation can help to assess the behavior of the system under average-case conditions.

One unique feature is that in these new WSN systems, it is not always possible to separately assess the behavior of the application messages sent over the network, and of the application tasks scheduled by the RTOS at individual sensor nodes. In other words, both the communication aspect and the computing aspect of such a WSN system usually are tightly coupled. In most cases, the behavior of the network can have a strong impact on the tasks, and vice versa. To the best of our knowledge, there are very few simulation tools that can support a joint modeling of the node (including the Operating System mechanisms) and of the network. An example is given in [6] although especially customized to model control applications.

The IEEE 802.15.4 [7] standard specifying the physical layer and media access control for low-rate Wireless Personal Area Networks (LR-WPANs) seems to fit the requirements of this "second generation WSNs" concerning inter-node R/F communication.

Thus, it is desirable to have a general simulation software package that can jointly simulate the network protocols (e.g., supporting the IEEE 802.15.4 standard) and the RTOS scheduling mechanisms, permitting an early performance assessment and a rapid prototyping of distributed sensor systems [8].

1.1. Contribution of This Work. Aimed to address this new challenge, we have proposed and developed a simulation tool called as Real Time Network Simulator (RTNS) which allows WSN designers to jointly explore the two dimensional design space with respect to computation and communication. This paper systematically presents our work on the RTNS [9–11]. The RTNS is a simulation suite to model operating system mechanisms for distributed networked applications, built by integrating the popular NS-2 (Network Simulator [12]) and RTSim (Real-Time operating system SIMulator [13]) packages. This tool facilitates designers to work in the two dimensions of communication (packet priority) and computation (task priority) in a systematic manner to achieve a more accurate and realistic WSN system's performance evaluation.

In particular, we demonstrate the benefits of the RTNS by presenting a complex WSN-based multi-view vision system for event detection in real time [14–16]. This application concentrates on WSN imaging applications where the efficiency and reliability of the information depends on the real-time support provided by both the kernel and the network stack. Thus a Multi-View Vision application suffers delays and jitters not only due to wireless communication related phenomena but also due to the way the software tasks are scheduled on each sensor node. This WSN-based multi-view vision problem is used as a vehicle to illustrate how the interplays between node computations and network communications can have a significant impact on system performance, and how the better design can be achieved with the help of simulation using the RTNS.

The rest of the paper is organized as follows. Section 2 overviews the state of art in simulation tools. Section 3 presents the architecture of the RTNS. Section 4 details the multihop solution in the RTNS. In Section 5, the RTNS' software performance is discussed in general. In Section 6, modeling WSN in the RTNS is described in detail. Section 7 presents a challenging multi-view vision problem based on WSN, which is used as a vehicle to demonstrate the advantages of the RTNS in the following two sections. Finally, Section 10 gives conclusions and our future work.

2. State of the Art

In discrete event simulation, the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system.

Discrete event simulators are usually implemented making use of a re-sizeable event queue where to post and pop events for appropriate processing. For instance, time-triggered activities regularly post expiration events into the queue to produce a periodic sequence of actions. The queue is reordered at every post to always keep the closest event in front; the physical notion of time is discretized and incrementally elapses by the interval between the two latest expiration events at every pop.

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 [17].

OPNET (Optimized Network Engineering Tools) [18] 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 customized protocols by reusing existing components.

Global Mobile Information Systems Simulation Library (GloMoSim) [19] 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) [20]. 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 [21] 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++ [22] (written in C++) and J-Sim [23] (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.

In the world of control systems, TrueTime [6] is a popular Matlab/Simulink-based simulator; it facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. The tool shares some of the issues addressed by RTNS but it is featured for control systems and has a naive model of the network.

A different role is played by the TOSSIM simulator, coming along with the TinyOS [24] 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, rather 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% [25] (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.

Popular simulators like OPNET, NS-2, and TrueTime support most of the features of the IEEE 802.15.4 standard for WPANs especially those related to the MAC layer mechanisms for network formation and management and contention-based transmissions (via the CSMA/CA scheduling algorithm). The support for Guaranteed Time Slots (GTSs) is absent in TrueTime whereas is provided by external contributions to OPNET [26] and NS-2 (through RTNS).

The authors of [26] motivates the selection of OPNET criticizing the (not native) support of NS-2 for wireless communications. Actually numerical comparisons of the two packages in consistent conditions are rare in literature. Those who tried to perform this comparative analysis sometimes ruled out both of them [27]. Recently [28], many arguments supporting the wireless model of NS-2 have been proposed justifying the unreliable results (sometimes obtained in simulation runs) as driven by a bad setting in some parameters of the wireless modules. The authors show that tuning these parameters permits a strict adherence between real world and simulated data.

Furthermore the remarkable strict adherence to the IEEE 802.15.4 standard of the NS-2 WPAN module [29] and the long debugging stage (from the release 2.26 to 2.31) which has patched the module to fix imprecise behaviors

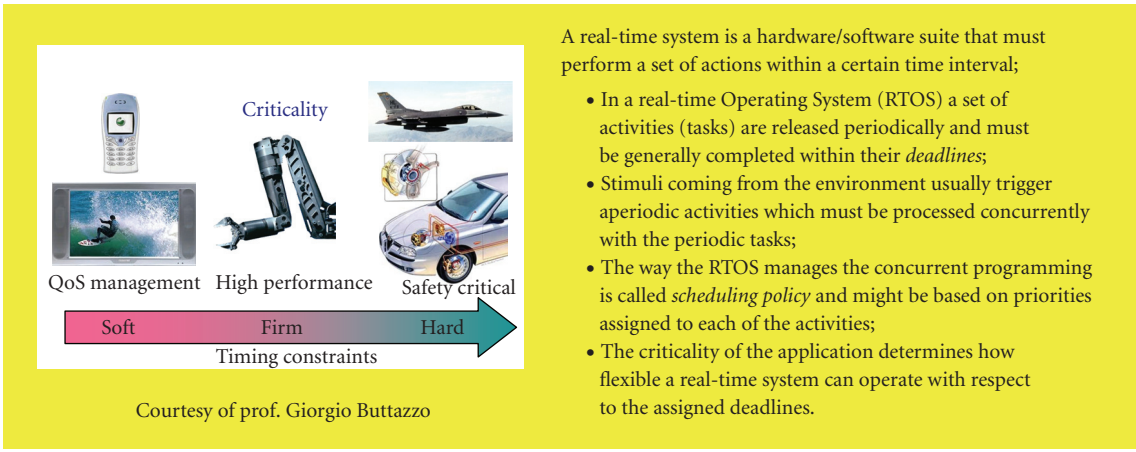


Figure 2: Some concepts related to Real-Time.

Table 1: Comparison among simulation packages. N = Native, A = Absent, EC = External Contribution.

	Kernel imitation	802.15.4 CSMA/CA	802.15.4 GTS
TrueTime	N	N	A
NS-2 (RTNS)	EC	N	EC
Opnet	A	N	EC

(see, e.g., Chapter "Changes made to the IEEE 802.15.4 Implementation in NS-2.31" in the reference manual [30]) legitimate the use of NS-2 as simulator in the WSN context.

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) [31], 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 [13] is a software package written in C++ for the simulation of real-time operating systems, available as open source [32]. 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.

The imitation of kernel mechanisms is natively included in TrueTime (by means of a Kernel block), added by COTS to NS-2 (using the RTNS extension) and totally absent in OPNET. In Table 1 a simple comparison of the simulators shows that RTNS represents the only software solution (yet the only open source), to the best of authors knowledge, for modeling distributed WSN applications with real-time constraints acting both at node and network levels.

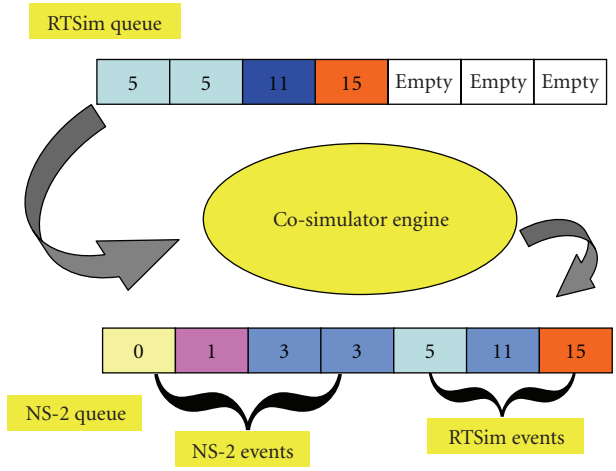


Figure 3: Synchronization of the NS-2 and RTSim schedulers. In the RTSim scheduler there are events at time 5,11,15; the co-simulator engine pushes into the NS-2 scheduler "synchro" events at these time instants to process all corresponding RTSim events at appropriate NS-2 time.

3. RTNS Architecture

In designing our simulator, we decided to reuse existing simulation tools as much as it is possible. We chose [9] the well-known NS-2 simulator for modeling networking protocols and messages, and the RTSim simulator for modeling the RTOS and application tasks (see Figure 2). By reusing existing open-source code, we take advantage of existing contributions of two research communities. Moreover, we can rapidly include future enhancements into our framework. In this section we sketch the structure of both simulation engines and we present the technique we used for the integration.

Both NS-2 and RTSim are discrete event simulators, thus they use their own queues to ensure a chronological sequence of events. We decided to keep the NS-2 event scheduler as the main engine, and make the RTSim engine as its sub-engine.

```
$ns node-config\
-  adhocRouting AODV \
 - llType LL\
 - ifqType Queue/DropTail/PriQueue \
- ifqLen 50 \
- macType Mac/802_15_4 \
- phyType Phy/WirelessPhy/802_15_4 \
- antType Antenna/OmniAntenna \
- channel [new Channel/WirelessChannel]\
- propType Propagation/TwoRayGround
set station [$ns node]
```

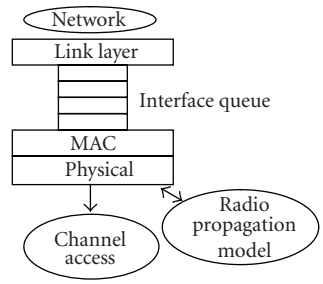


Figure 4: An example for the configuration of a wireless sensor node compliant to the IEEE 802.15.4 standard. The parameters of the NS-2 API (in TCL scripting language) follow the color scheme of the figure on the right: in Orange the Radio Propagation Model, Channel, and Antenna specifications, in Blue the Physical layer, in Green the MAC layer, in Purple the ifQ type and dimensioning, in Red the Routing protocol.

We defined a special event in NS-2 called the "synchro" event, that takes care of processing all events of RTSim that happen at a single point in time as shown in Figure 3.

This design solution

(i) allows to consume time for the "intra-node" activities not related to the network stack operations and ignored by the standalone NS-2 package,

(ii) makes RTNS be seen by the end users as an extension of the NS-2 platform, allowing full compatibility with existing functionalities.

ROOT [33] is a framework for data processing, born at CERN, organized in a collection of C++ classes permitting to save, access, and process data for off-line analysis.

From the code perspective RTNS is obtained linking together NS-2, RTSim, and ROOT at configure time. The data produced on the fly in the simulation are organized in a Tree structure (eased by the on-line availability of the ROOT classes) and saved on disk during the RTNS simulation runs. This structure is then accessed off-line by means of an analysis toolkit for post-processing.

In Figure 4, an example for the usage of the NS-2 API to configure a wireless sensor node is shown, remarking the correspondance with the ISO/OSI model of the network stack.

To enrich the simulation with all the mechanisms induced by the OS, in RTNS the node is equipped by a scheduler that, following a (real-time) policy, interleaves the tasks related to computation ("intranode" activities) with those related to networking ("extranode" activities).

NS-2 front-end (the TCL scripting) has been extended by providing a set of APIs permitting to set the CPU clock, the number of running tasks, and the utilization factor; the tasks execute a customizable instruction set reserving the CPU for a fixed or random (statistically distributed) number of clock ticks.

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 (see Figure 5). Statically, through the TCL front-end, a certain number of tasks are instantiated on the node RAM memory. These tasks can assume a certain semantics (i.e., modeling definite activities of the wireless node like the services offered by the network stack, notably "Send" and "Receive") or blindly consume CPU time (i.e., dummy tasks). RT-App makes use of RT-Agent to inject and retrieve packets to and from the network.

The Operating System puts the tasks into the "running" state following the adopted scheduling policy selected along with the kernel.

The examples discussed in the remaining part of the present and in the forthcoming sections will naively make use of the native NS-2 support for the IEEE 802.15.4 protocol suite. A detailed discussion on the protocol and its implementation in the NS-2 and RTNS simulators will be the subject of Section 6.

3.1. Effect of CPU Load in Transmission End-to-End Delay. In a distributed real-time networked application, the messages exchanged by wireless nodes are in general valid within a certain time window because a late transmission may share wrong readings and a late elaboration may trigger reactions out of time.

RTNS keeps trace of the time overhead spent at each layer of the network stack. For instance in case of periodic single hop transmissions (like CBR traffic), messages are regularly delivered to the Physical layer of the sink node.

To show the impact of the scheduling policy, suppose the sink node is running a task τ as a periodic activity (e.g., like Sampling), and another task wrapping the receive function of the network stack. The arrival of a data frame at the Physical layer activates the receive, but the task goes to the running state whenever τ finishes thus delaying the reading of the packet (see Figure 6).

Having set the CBR interval to 70 ms (slightly more than 75 ms, period of the load task), the packet arrival intersects

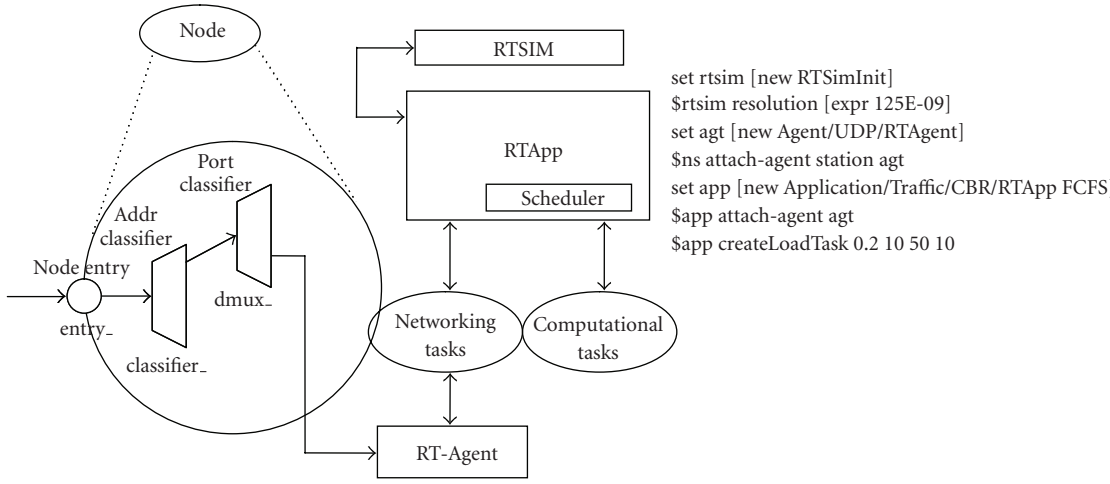


Figure 5: An example for setting up a 8 MHz CPU running an OS with FCFS scheduling policy and a task set generating a computational load of 0.2. The parameters of the RTNS API (in TCL scripting language) follow the color scheme of the figure on the left: in Green the RTSim scheduler instance accustomed for the CPU speed, in Purple the NS-2 networking protocol, in Blue the OS abstraction with the scheduler in Red , in Orange the Task Set generating the load.

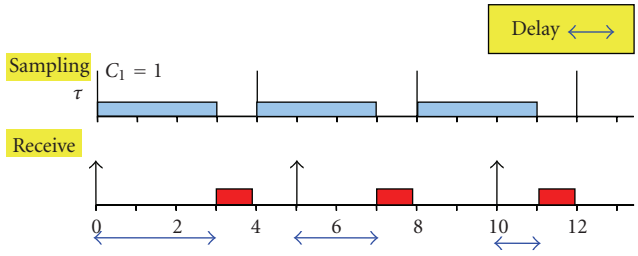


Figure 6: Concurrent activities in the recipient node.

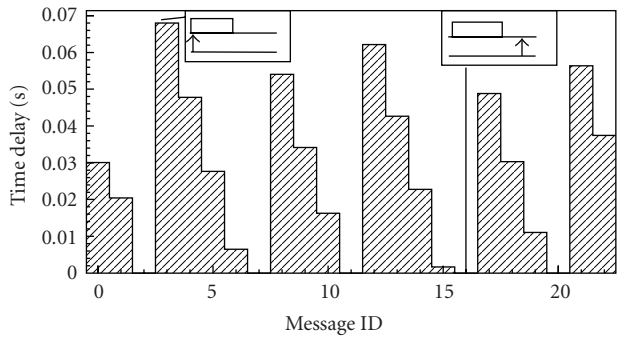


Figure 7: The saw-teeth-like pattern generated by the FCFS scheduling policy adopted for the sink kernel. The plot refers to UF = 0.8.

the load task later and later thus being delayed less and less. In Figure 7, adopting a non-preemptive scheduling policy, the packet reception delay is plotted against a progressive id.

If the data packets are valid say for 50 ms, as it may happen for object localization purposes, video frame transmissions and so on, RTNS shows how this setup is not conform with the QoS the network must respect; the designer is aware that the computational profile assigned to the sink is incompatible with the role it has in networking.

The mapping of functionalities into tasks can eventually be re-evaluated to look for a solution improving on the QoS.

4. Multihop Solution

As explained in the previous section the NS-2 RT-App class is in charge of abstracting the operations performed by the Operating System. The NS-2 RT-App class introduces delays in the packet transmission by scheduling them along with other events of the Operating System. An intermediate NS-2 node which acts as a router in a multihop communication does not forward an incoming packet to the nodes NS-2 RT-App class. Instead, the NS-2 node forwards the packet on an outgoing wireless path after determining the next hop neighbor. In this process, the NS-2 node neglects any delay induced by other tasks running simultaneously on the node.

In this section we will address the problem of scheduling delays in wireless routers. We propose a modification to the NS-2 architecture to solve the problem. The solution described in this paper will help to correctly simulate a wireless multihop scenario, and provide an accurate measure of packets delay.

A unicast node in NS-2 is shown in Figure 8. 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 receiving 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

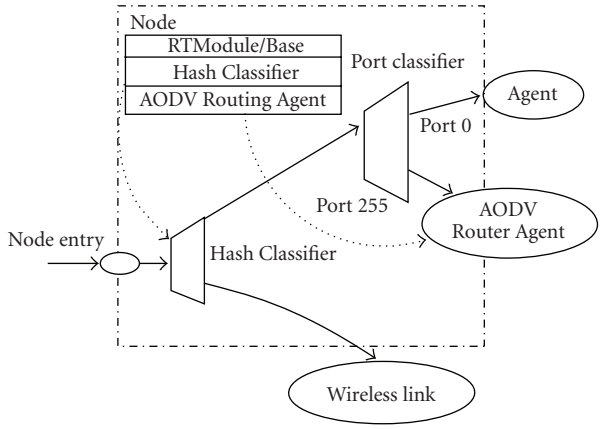


Figure 8: Architecture of a wireless node in NS-2.

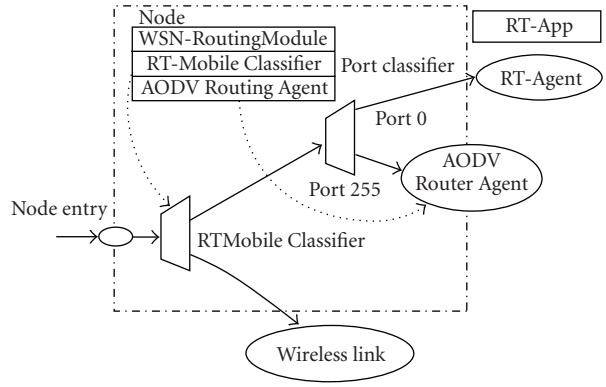


Figure 9: Modified architecture of Wireless Node in NS-2.

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.

Because of this behavior the simulator is unable to include 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 forwarding of packets in a way that will recognize the presence of RT-App. The latter will schedule the routing of an incoming packet along with other tasks running simultaneously on the node, thus delaying the packet transmission depending both on the instantaneous load and the kernel scheduling policy.

As shown in Figure 9 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 the functionality of Hash 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 like "UDP").

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 iteration), 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 retransmitting 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*.

4.1. Effect of CPU Load in Multi-Hop Scenarios. The simulations run through RTNS eases a possible co-design of the kernel structure and the network protocols [34] for specific applications.

The real-world microcontrollers adopted for WSNs are very simplified in hardware (not equipped with co-processors devoted to networking) and MAC layer is purely coded in software. In ad-hoc multi-hop networks, since the "routing" activity is scheduled concurrently with all the others, it is of paramount importance to evaluate the impact of the computational load on the transmissions.

A realistic statistical analysis should start from the model of the load assigning to nodes a certain probability of CPU usage (Utilization Factor, UF). In the simple case of a WSN where the nodes are running the same executable, the nodes may have a constant uniform UF related to the complexity (Execution Time) and the period of the executable (as in Figure 10(a)); another scenario is that of Figure 10(b) where the UF might vary as a function of the space following a Gaussian distribution.

Having modeled the distribution of the load, it's possible to simulate a multi-hop transmission retrieving the network delay as a function of the cumulative load along the routing path. The sink feels the delay in message delivery as

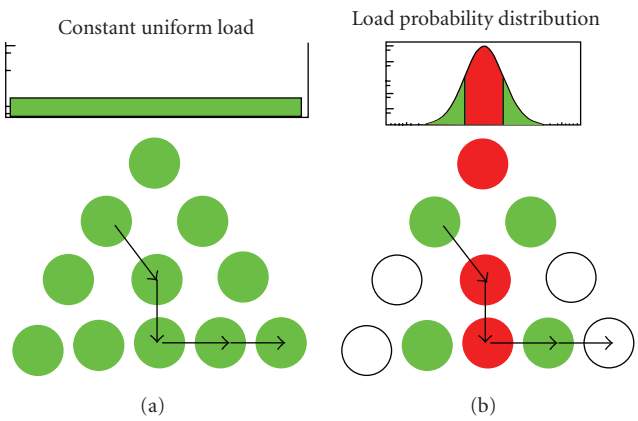


Figure 10: CPU load models for a tree-like topology: (a) uniform and (b) gaussian distributed.

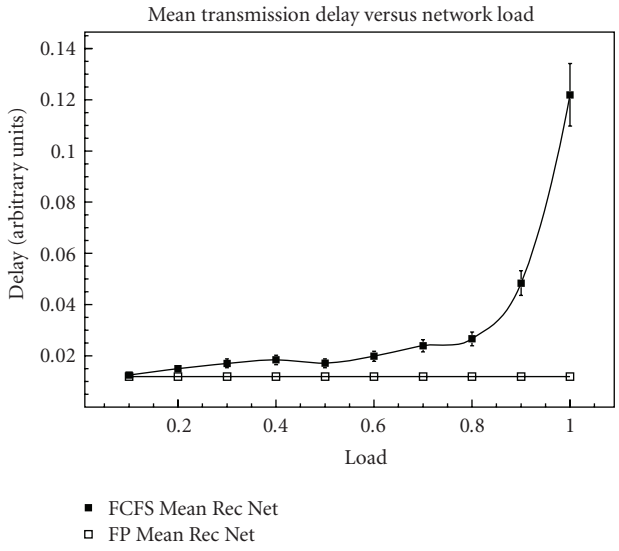


Figure 11: Time delay to read data packets induced by concurrent activities using a FCFS scheduling policy.

a congestion occurrence in the network as it happens when the packets arrive late at destination.

If the network follows a congestion avoidance protocol, the sink may decide to wrongly re-set some of the communication parameters. A reliable simulation permits to filter the effects coming from network congestion from those related to CPU load, permitting an accurate setup of networking protocols.

As a matter of example, in Figure 11, the End-to-End mean transmission time is plotted against the CPU load in the nodes having adopted a model like that of Figure 10(a).

If the load probability distribution is static, it might be preferable to adopt a static routing table (at design time) minimizing the cumulative load on the routing path. If the designer has concerns about the faulty nature of the nodes (which is incompatible with static routing), or if the load distribution happens to change in time, popular on-demand routing protocols can be extended paying some price in terms of over-head but gaining effective predictability in communication.

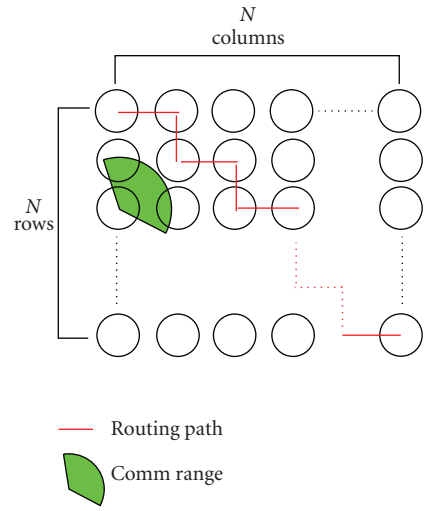


Figure 12: Lattice like scenario with the two opposite nodes on the diagonal willing to exchange data. The total number of nodes, figure relevant for the scalability of the package is a parameter in the simulation.

5. Simulator Performance

The usage of RTNS requires more resources concerning disk space, memory, and time overhead with respect to NS-2. The tarball size is of the order of 150 MBytes and includes the components required by the tool: ROOT [33] and RTSim [32] are external packages (free sources) which must be linked at compile time.

The overhead induced by the Operating System simulation on one hand enlarges the time needed to simulate the network operations, on the other hand may limit in number the nodes composing the network.

Referring to the scenario in Figure 12, where the two farthest nodes along the diagonal in a rectangular lattice are exchanging data, we measured the simulation time supposing a uniform CPU occupancy in the nodes equal to 50%. The network is operated in "non-beacon" mode as defined in the IEEE802.15.4 standard for a peer-to-peer communication paradigm.

As shown in Figure 13, NS-2 and RTNS show a similar trend in the simulation time as a function of the total number of nodes in the lattice, the two curves being displaced along Y by a fixed amount.

The RAM footprint increases linearly as a function of the number of nodes, as it is expected from the linear increase in the number of RTNS objects (kernels, schedulers, tasks, etc.). The plot in Figure 14 demonstrates this linear dependence, showing that the difference with standalone NS-2 resides only in the slope of the two lines.

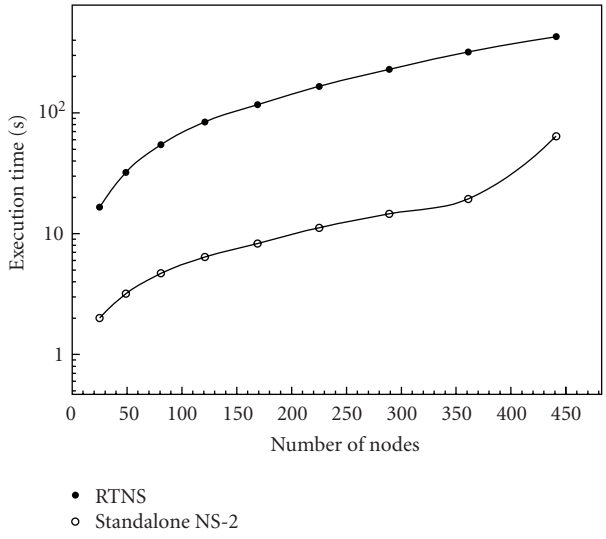


Figure 13: Execution time as a function of the total number of nodes for standalone NS-2 and RTNS.

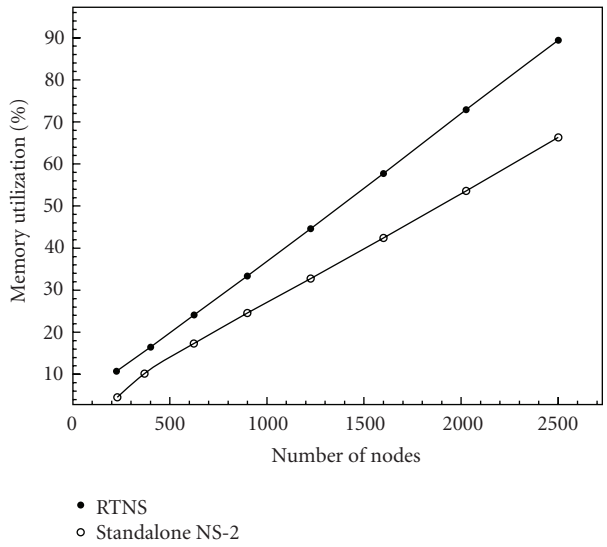


Figure 14: RAM maximum usage of NS-2 and RTNS as a function of the total number of nodes.

6. Modeling WSN in RTNS

6.1. The IEEE 802.15.4 Standard. The IEEE 802.15.4 protocol specifies the Medium Access Control (MAC) sublayer and the Physical Layer of Low-Rate Wireless Personal Area Networks (LR-WPANs). The SSCS (Service Specific Convergence Sublayer) abstracts the Access Point of some services offered by the MAC to upper layers.

The IEEE 802.15.4 Physical Layer uses a 16-ary encoding alphabet (4 bits/symbol) in Direct Sequence Spread Spectrum (DSSS) modulation over three operational frequency bands: 2.4 GHz (16 channels); 915 MHz (10 channels); 868 MHz (1 channel). The IEEE 802.15.4 MAC layer supports two operational modes: (1) the non beacon-enabled mode, in which the MAC is simply ruled by non-slotted CSMA/CA; (2) the beacon-enabled mode, ruled by slotted CSMA/CA, in which beacons are periodically sent by a special node (called network coordinator) to synchronize (in time) nodes that are associated with it, and to carry additional information about the transmission structure. In beacon-enabled mode, the Coordinator defines a SuperFrame structure (Figure 15) which is constructed based on (1) the Beacon Interval (BI), which defines the time between two consecutive beacon frames; (2) the SuperFrame Duration (SD), which defines the active portion in the BI, and is divided into 16 equally-sized time slots, during which frame transmissions are allowed. Optionally, an inactive period is defined if BI > SD. During the inactive period (if it exists), all nodes may enter in a sleep mode (to save energy).

BI and SD are determined by two parameters—the Beacon Order (BO) and the SuperFrame Order (SO):

$$\begin{aligned} \text{BI} &= aBaseSuperFrameDuration \cdot 2^{\text{BO}} \\ \text{SD} &= aBaseSuperFrameDuration \cdot 2^{\text{SO}} \end{aligned} \tag{1}$$

assuming $0 \le \text{SO} \le \text{BO} \le 14$, *aBaseSuperFrameDuration* = 15.36 ms (operating at 250 kbps in the 2.4 GHz band),

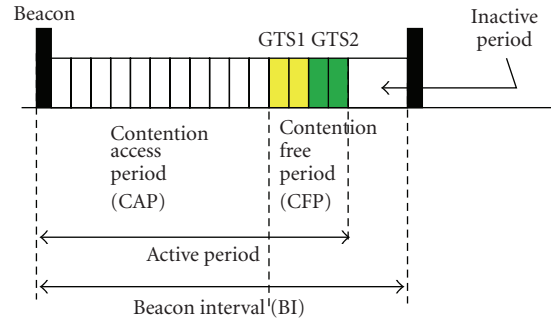
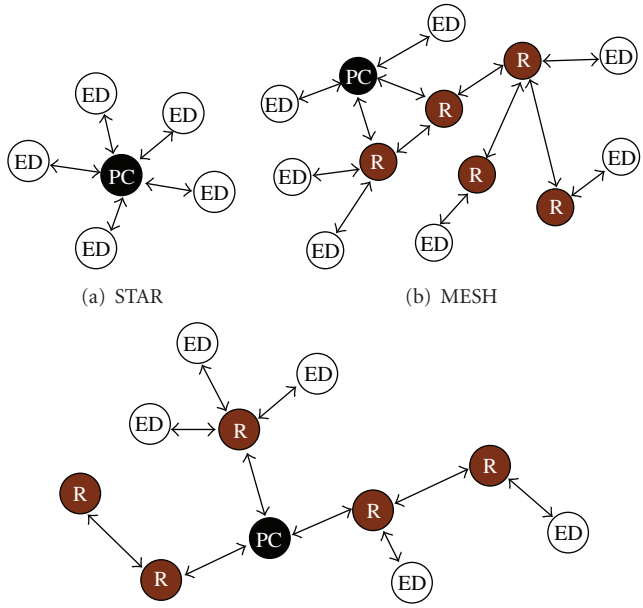


Figure 15: SuperFrame Structure in the IEEE 802.15.4 standard. In beacon enabled mode, the beacon interval can be subdivided into a Contention Access Period (CAP) and eventually into a Contention Free Period (CFP) and Inactive Period.

corresponding to the minimum SuperFrame duration at SO = 0. During the SuperFrame Duration, nodes compete for medium access using slotted CSMA/CA, in the Contention Access Period (CAP). IEEE 802.15.4 also supports a Contention-Free Period (CFP) within the SD, by the allocation of Guaranteed Time Slots (GTS). It can be easily observed in Figure 1 that low duty-cycles can be configured by setting small SO values as compared to BO, resulting in longer sleep (inactive) periods. The standard supports three network topologies: Star, Mesh and Cluster-Tree, illustrated in Figure 16.

In the Star topology (Figure 16(a)) communications must always be relayed through the coordinator; Star networks can operate in both beacon-enabled and non beacon-enabled modes. In the Mesh topology (Figure 16(b)), each node can directly communicate with any other node within its radio range or through multi-hop; Mesh networks must operate in the non beacon-enabled mode. The Cluster-Tree topology (Figure 16(c)) is a special case of a Mesh network where there is a single routing path between any pair of nodes



(a) STAR (b) MESH

(c) CLUSTER-TREE

Figure 16: WPAN star, mesh, and cluster-tree network topologies.

and a distributed synchronization mechanism (operates in beacon-enabled mode).

6.2. The WPAN Module in RTNS. In Figure 17 the network stack standardized in the IEEE 802.15.4 suite is shown. In the superimposed call-outs, the corresponding services implemented in the native WPAN module in NS-2 (release 2.33) and the modifications introduced by this research project are specified: we namely refer to the GTS mechanism and the link with the RTNS framework.

The scarce documentation about the WPAN package in NS-2 refers to the mechanisms exported to the final user interface in some TCL scripting examples [35]: these mechanisms refer for instance to network start-up, node association, network topology and beacon order selection, and so forth.

Following a strict adherence to the standard, the MLME-GTS.request, MLME-GTS.confirm, and MLME-GTS.indication MAC primitives have been implemented for the GTS allocation as shown in Figure 18.

When an (associated) node wants to transmit (or receive) data in real-time, it makes use of the GTSs. A MLME-GTS.request is generated at Network layer; afterwards the node sends a Command frame to the coordinator. The device waits for the coordinator acknowledges receipt and then parses the list of GTS descriptors in the forthcoming beacon to identify the starting slot assigned to it. After the transmission of the ACK frame to the device, the MAC layer of the coordinator calls the MLME-GTS.indication notification procedure to its agent. At the reception of the beacon, the MAC layer of the device notifies the success to its agent by calling the MLME-GTS.confirm procedure.

A GTS can be deallocated whenever the device node formulates an explicit request as shown in the collaboration diagram of Figure 19(a) with a sequence of function calls very similar to the previous case.

Moreover the coordinator can deallocate a GTS (see Figure 19(b)) inserting that GTS descriptor into the list of "removed" GTS in the forthcoming beacon packets whenever one of these conditions happen:

(i) the upper layers of the coordinator require the deallocation;

(ii) the device did not make use of the GTS (in reception/transmission) for 2^n SuperFrames where $n = 2^{8-BO}$ if $0 \leq BO \leq 8$ and $n = 1$ if $9 \leq BO \leq 14$.

To fulfill these functionalities, a GTS DataBase structure has been created. By means of appropriate classes the database is instantiated in both the coordinator and devices memories allowing for

(i) preparing the list of the GTS descriptors to be attached to the beacon frame periodically broadcasted by the coordinator;

(ii) activating the hardware timers for transmitting (receiving) data within the CFP of the MAC SuperFrame;

(iii) enabling the GTS re-location and extending the CAP time interval when GTSs are deallocated.

7. The Multi-View Vision Problem Based on WSN

Hereby we want to discuss the feasibility of image reconstruction making use of wireless sensor devices equipped by photo cameras capturing complementary portions (i.e., views) of a scene pictorially shown in Figure 20: a network controller is building up an event record collecting the data frames coming from device nodes in a definite time window (Protocol Timeout or Detection Window).

7.1. Problem Statement. Due to the resource limitations in Wireless Sensor Networks (WSN), the transmission of the original flat images over the air, in a video streaming fashion, is not simple or feasible at all. Hereby without any desire of completeness we discuss some of them:

(i) the limitation in the network bandwidth (e.g., 250 Kbps maximum in the IEEE 802.15.4 standard) causes long transmission (even with compression) with respect to other wireless protocols like WiFi (a.k.a. IEEE 802.11 b/g, 11/54 Mbps maximum bandwidth), which implies long latency at the network controller and possibly reactions out of time;

(ii) the tiny dimension of the packet frame (e.g., 114 bytes maximum payload at the MAC layer in

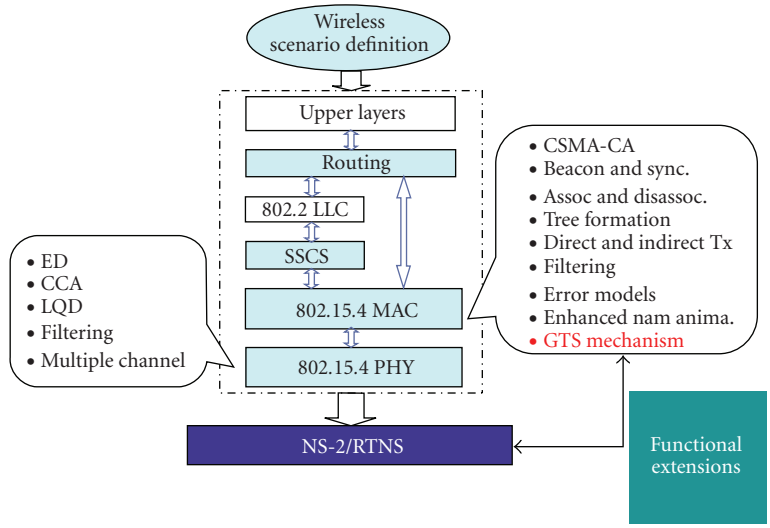


Figure 17: The networking stack as standardized in the IEEE 802.15.4 protocol for WPAN. In the call-outs the services implemented in the NS-2 WPAN module and the functional extensions added within this work.

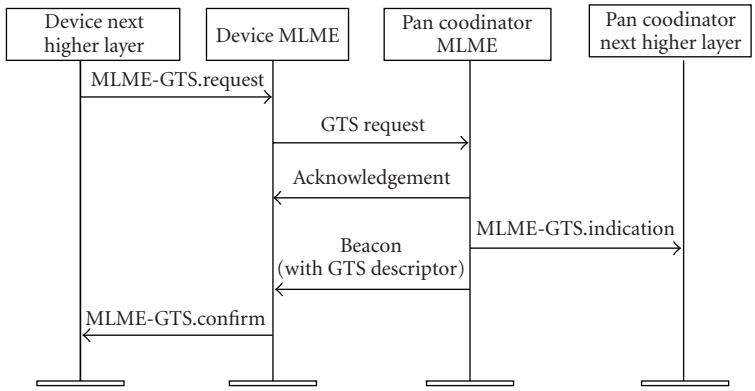


Figure 18: Sequence diagram for a GTS allocation request by a device node.

the 802.15.4 standard) forces the data sources to multiple transmissions; if retransmissions are permitted and/ or data flows are routed through multi-hop paths of different length, mechanisms for packet reordering at the sink/control node are needed for data coherence;

(iii) retransmissions do not fit the power-aware policy of WSNs whose protocols are designed to maximize the lifetime of the batteries and a fortiori to minimize the total number of transmitted symbols. Also, retransmissions result in extra delays which might violate the time constraints for reactions;

(iv) Multi-View Vision systems are definitely real-time distributed systems; local activities at the node level must be supported by real-time kernel and real-time communication techniques must be implemented in the network stack.

Other arguments, hardware dependent, like the resource set reduction in Micro-Controller Unit (MCU) (namely, computing power and memory) complicate even more the envisioning of a video streaming service based on WSNs.

The other option is to consider the microcam nothing else than a sensor providing (not scalar) information to the main unit. Adopting this model, we can deploy a Real-Time Multi-View Vision system by means of WSN technology provided some logic resides at the node level where the images must be (quickly) processed.

The vision operations can be described as 2-tier (hierarchical) process where the local decisions are taken at the device node sending out a *report* towards the sink which is in charge of combining the information coming from all the sources to eventually react. Of course the controller cannot make use of the genuine information and *relies* on devices although the preprocessing stage generates information loss in any case.

All in all this 2-tier process in the end is displacing the complexity towards the device nodes, profiting of the enhanced computing power of modern MCUs recently embedded in some Sensor Boards [36, 37] of the new generation like those provided by Microchip [38], Intel

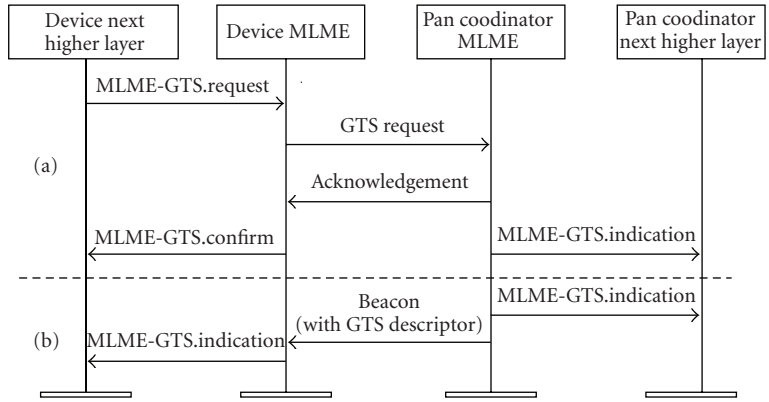


Figure 19: Sequence diagram for a GTS deallocation request by a device node (a); explicit removal by the coordinator (b).

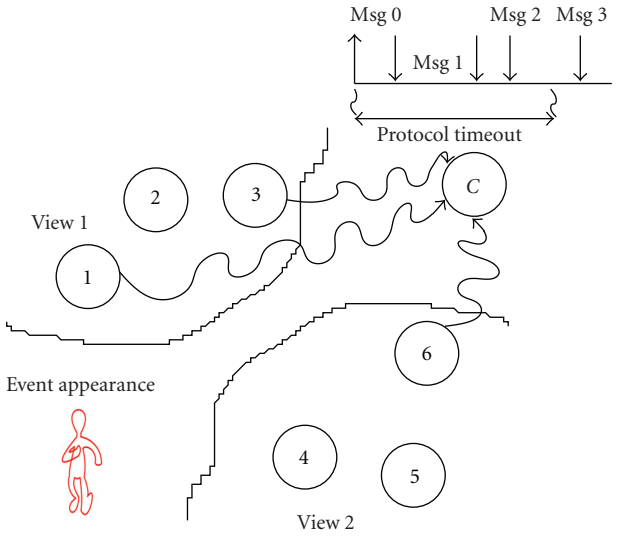


Figure 20: A WSN is detecting the appearance of an event from the integration of 2 not-overlapping views.

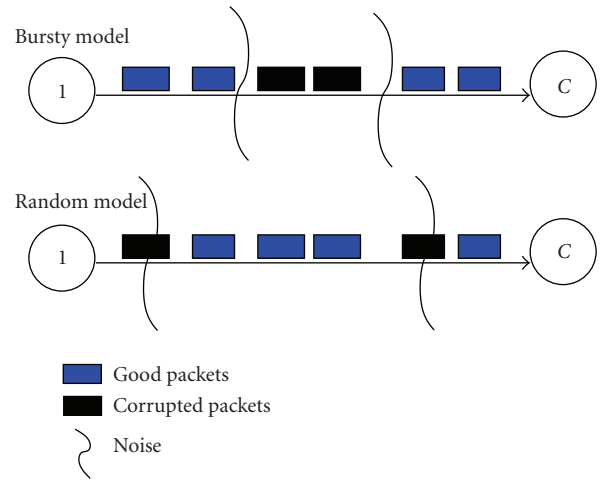


Figure 21: Noise is affecting packet integrity. Interference can occur randomly or burstily.

[39], and others reaching the speed of tens or hundreds of MIPS.

In the following we will first analyze how to achieve reliable communication in Wireless Sensor Networks to further discuss several options for node and network configuration.

7.2. Reliable Event Detection. To build up a reliable system, the network architecture and the transmission protocol must be robust with respect to transmission errors including packet corruption and collisions.

In the following we discuss a strategy to implement reliable communication acting at different levels.

The 802.15.4 standards indirectly discourages long transmissions forcing data fragmentation into tiny (127 bytes long physical payload) packets. The packet length is motivated by the envisioning to deploy WSNs in noisy environments where the Bit Error Rate (BER) is high.

Transmission Errors can occur randomly in time (random noise) or be localized in specific intervals (bursty noise) as depicted in Figure 21.

Both for Bursty and Random error models long transmissions are discouraged. Moreover, in case of random noise, the shortest the packet the lowest the Packet Error Rate (PER) being

$$\mathrm{PER} = 1 - (1 - \mathrm{BER})^n \tag{2}$$

with n being the dimension of the packet in number of bits.

Adopting a random error model, if we send m multiple copies of the same *report*, the combined probability of loosing all packets is PER^m. The actual choice of m, which is ultimately a configuration parameter of the protocol, can be done once we measure the BER figure and we set a threshold in the probability of successful delivery $\mathrm{PER}_{\mathrm{thr}}$ such that

$$m : \mathrm{PER}^m < PER_{\mathrm{thr}}. \tag{3}$$

To overcome random errors, multiple copies of the report can be sent back to back reaching a confidence level equal to

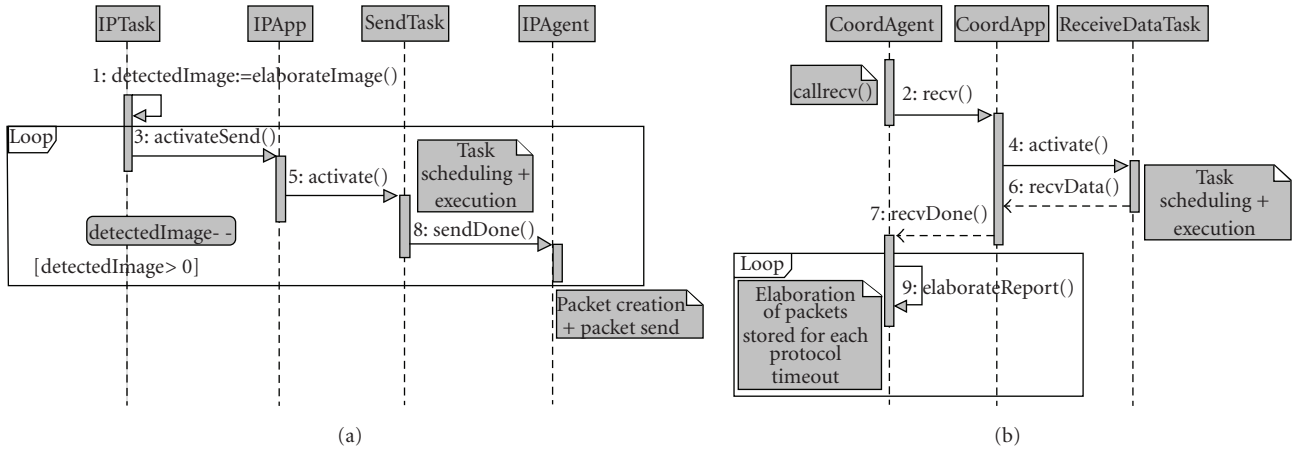


Figure 22: The UML sequence diagrams for sending (a) and receiving (b) reports.

$(1 - \mathrm{PER}^m)$ for at least one packet reaching destination without corruption. This can be further enhanced to consider adding redundant sensor nodes for each view to overcome bursty errors.

7.3. The MAC Level. Within the framework of a Multi-View Vision system, retransmissions should be avoided: the evident motivation coming from the need of reducing the total duration of network transactions.

In general the software packages implementing the Network stack for WSNs do not include Transport Layer mechanisms like packet flow reordering and data retransmissions although the latter is optionally provided as a MAC service in the IEEE 802.15.4 standard.

Other mechanisms must be adopted to guarantee successful report deliveries.

Following the multiple transmission techniques here proposed we can be rather confident that the needed information are in the end carried through the medium without corruption. Anyhow packet losses may occur because of collisions, and late data transfers can be caused by missed bandwidth (awaiting for the medium being idle in case of high volume of injected traffic from different sources).

A MAC module implementing the transmission algorithm like CSMA/CA (used for data transmission in every commercial stack for Sensor Boards) does not respond to the requirements of bounding the communication delay (which is in principle infinite) and avoiding packet collisions.

Bandwidth allocation techniques providing TDMA-based access solves the unpredictability in the total duration of network transactions and guarantees timing constraints to the networked application. Moreover if time synchronization among the nodes is enforced at the MAC layer occasional collisions should be in principle avoided and successful reports guaranteed.

The existing technology standardized in the IEEE 802.15.4 protocol actually fits the requirements discussed hereby allowing Collision Free transmissions through the usage of the Guaranteed Time Slots (GTS) and time synchronization through the beacon-enabled mode of the MAC operations.

7.4. The Kernel Level. In so far we implicitly assumed that *reports* can be easily packed and transmitted over the air within a specific GTS or similia. This is actually not straightforward for many working solutions of network stacks implemented on top of non real-time kernels. Local processing is required and the Operating System must be able to perform time synchronization over the tasks devoted to computation and networking.

In our hypothetical system the image frames must be timely processed to fit the transmission of the *reports* in the allocated band. Late Operating System responses ultimately cause loss of Quality of Service and system malfunction.

7.5. Performance Metrics. Our distributed application makes use of the device nodes which apart from image capturing can perform image processing and send *reports* towards a defined location (say the sink). For simplicity (and considering statistical effects only although systematics is expected to play a relevant role in this context) we intend the Multi-View Vision Application as a generalization of a detector providing binary results (0: track detected, 1: track missed).

In a real-time perspective, the system overhead for event detection is very relevant. We imagine that controller reacts whenever the Detection Window is closed, so that its *response time* for event i happening at time T^i_{event} is

$$R^i = T^i_{\mathrm{detect}} - T^i_{\mathrm{event}}\,. \tag{4}$$

Its average value is given by

$$\overline{R} = \frac{\sum_{i=0}^{k} R^i}{k}, \tag{5}$$

k being the number of detected events when n is the true number of events in the WSN scope.

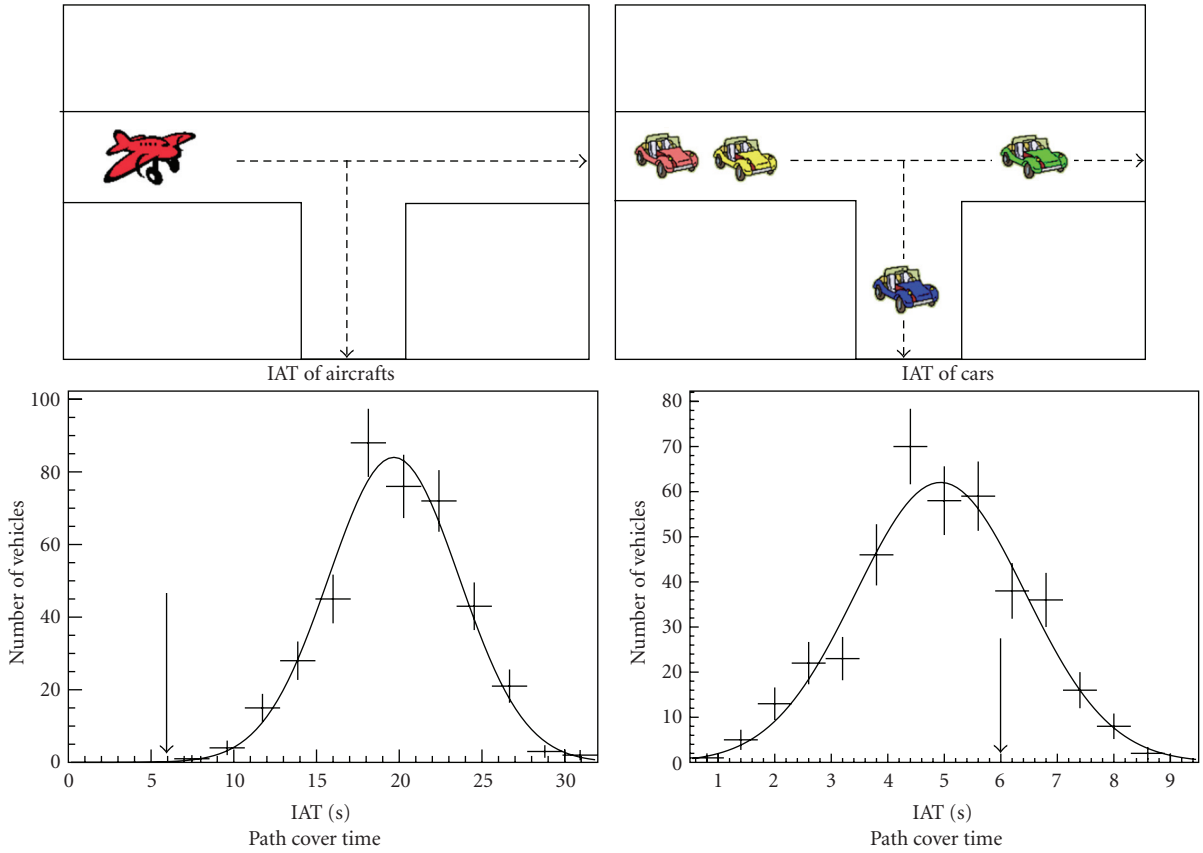


Figure 23: The Inter Arrival Time (IAT) distribution of vehicles in the taxiway and parking scenarios. The (fixed) path cover time is superimposed in the graphs.

To assess the performance of such system we should make use of global metrics such as the *efficiency* (ϵ) defined as

$$\epsilon = \frac{k}{n}. \tag{6}$$

Of course, an ideal system detects all the events ($\epsilon = 1$), and minimizes the response time ($\overline{R} \to 0$).

Referring to Figure 20, a network is composed by 6 nodes getting two independent views of the same scene. *Redundant information* are usually injected into the network if the number of nodes is larger than the number of views.

Of course the matter of efficiency cannot be considered independently from networking issues: the report proliferation induced by information redundancy can cause traffic congestion or, in TDMA based accesses, bandwidth lack since GTSs (maximum 7 in case of IEEE 802.15.4) represent scarce resources allocated on the basis of the node MAC addresses.

8. Multiview Vision Support in RTNS

We introduced into RTNS the abstraction of external stimuli modeled outside the WSN (by means of entities not directly participating in communication like vehicles) and generating event-driven transmissions. We believe this simulation paradigm fits very well the essentials of the domains where WSNs can effectively be deployed.

On one hand we included intra-node mechanisms and external stimuli (generating event-driven transmissions) into the model; on the other hand we adopt global level metrics like system latency and efficiency which serve for system validation from a holistic perspective. These objects can be detected by cameras according to the computational activity in the device CPU.

The device nodes are equipped with detection peripherals like pin-hole cameras: each camera captures the portion of the background scene covered by its solid angle. The topology support offered by NS-2 has been extended to introduce self-moving entities like vehicles not involved in communication. The moving targets act as external stimuli inducing transmissions by device nodes: network activity is therefore event-driven differently from the time-driven traffic generation which is generally adopted in NS-2 based simulations.

Each microcontroller in an individual device node runs the same firmware encoding the activities related to networking and Image Processing (IP). As discussed in Section 3, the RTNS Kernel prototype (implemented in the RT-App class) abstracts the services related to the scheduling policy and resource access. We define task the computational unit corresponding to one activity. A task consists of a set of

instructions that, when executed, reserve the CPU for a finite amount of processor ticks. The tasks can be run concurrently at a node.

The IPApp class is the specialization of the RT-App base class used in this work for instantiating the node kernel. IPApp handles the I/O from the peripherals and executes a S/W task customized for Image Processing (IPTask). Together with IPTask, the SendTask and ReceiveTask implementing the Network layer functionalities for data exchange are spawned at the start-up of the device nodes.

The IPTask task is a periodic activity, parameterized with a likely number of lines of code (Execution Time) and a Period set to the inverse of the maximum frame rate ($T = 1/\sharp fps$) of the camera. When the permanence of an object inside the view of a camera is such that the processing task intercepts it, a notification in the shape of unicasted message ("a report") is sent to the network coordinator. The report transmission is handled by the IPAgent class which represents the UDP-like endpoint of the network stack.

The coordinator is required to collect all the reports in a Detection Window (DW) to take action. The CoordApp class (abstracting its kernel) handles the I/O from the transceiver. The UDP-like agent CoordAgent correlates all the received reports ending in the same DW, and, from their unique signature, predicts the target track on the topological grid. These mechanisms are described by means of UML sequence diagrams in Figure 22. The data coming from the reports are organized in a Tree structure (eased by the on-line availability of the ROOT [33] package classes in RTNS) which is saved on disk during the RTNS simulation runs.

9. Real-Time Multiview Vision Case Study

In this section, we present the effectiveness of the RTNS simulator with the help of a distributed imaging application in Multi-View Vision applications. In this application, the distributed system is in charge of detecting a vehicle and sending the results of image processing tasks to the co-ordinator. The co-ordinator is responsible to take appropriate action based on the number of events arrived within the DW. RTNS provides the necessary means to analyze such a system in terms of OS scheduling of imaging and networking tasks, bandwidth allocation using GTS which helps to determine the event detection efficiency of the overall Multi-View Vision system. The metrics used to assess the system performance have been defined in Section 7.5.

9.1. Event Distribution Models. In Figure 24, five nodes build up a WSN and are in charge of tracking a target along two possible directions. The camera views do not overlap and the object provides a unique signature in case of going straight (reports from nodes 0,1,2) or turning right (reports from nodes 0,1,3).

We simulated two scenarios with different statistical distributions of event InterArrival Times (IAT) having fixed the time needed by the target to travel across the camera views (path covered time): a taxiway in a big airport, and a parking area (see Figure 23).

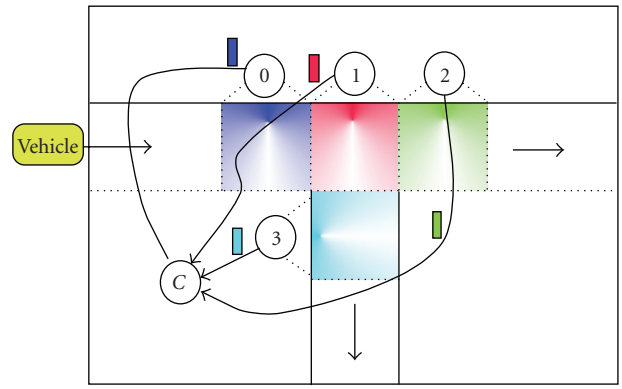


Figure 24: The scenario used for RTNS simulation. Nodes 0,1,2,3 track a vehicle using embedded pin-hole cameras covering complementary views. They send detection reports to the coordinator whenever they recognize the target entered their camera view.

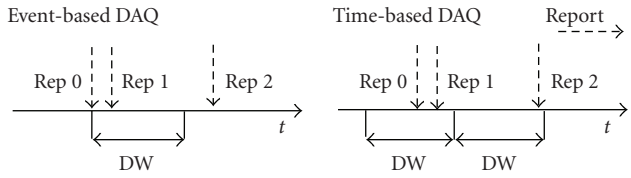


Figure 25: The DAQ logic: the event-based opens the DW at the arrival of a report from Node 0; the time-based opens the DW regularly as a function of the time.

In the first case the distributed system provides a critical service where events expectations are rare with respect to path covered time. If the IAT are Gaussian distributed, this analytically translates to the constraint:

$$\Delta T^{\min}(\text{events}) = \langle \Delta T \rangle - 3 \cdot \sigma > \frac{S}{V}, \tag{7}$$

S, V, $\langle \Delta T \rangle$, and σ being, respectively, the path length, the target speed, the average value and the standard deviation of the IAT probability distribution. In this simplistic model, all aircrafts move at the same speed thus path covered times are deterministically computable.

If we drop the constraint 3, we can have a superposition of arrival events in the scope of the cameras ending in the generation of reports related to independent detections.

The distributed system we keep unchanged from the code perspective responds differently in the two cases as we will see in the remaining part of this section.

We model two different types of Data AcQuisition (DAQ), the time-based and event-based (see Figure 25). Since the expected signatures are $\{0, 1, 2\}$ or $\{0, 1, 3\}$, if the chain of the events is truncated and the reports split into subsequent DWs, the event is discarded.

The time based DAQ is purely hardware, based on a timer activating at constant intervals the detection window. The reports related to the same event arrive in random order being the DW totally uncorrelated with the report arrivals.

Alternatively we can elect Node 0 as the DAQ trigger since all incoming vehicles pass through its hardware view.

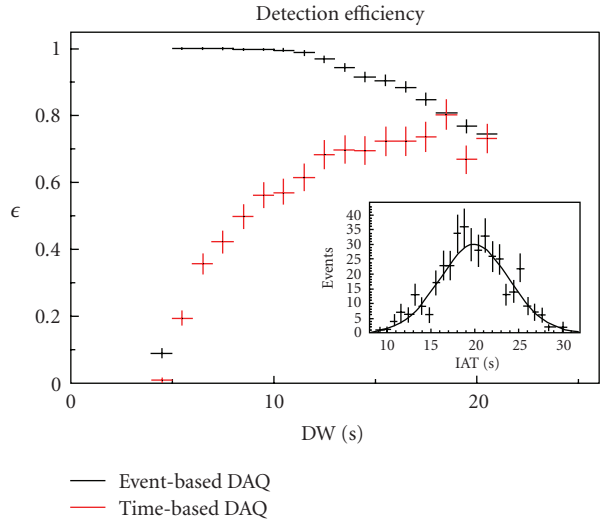


Figure 26: Efficiency behavior as a function of the DW width for the two DAQ models. The embedded plot is a reminder of vehicle IAT.

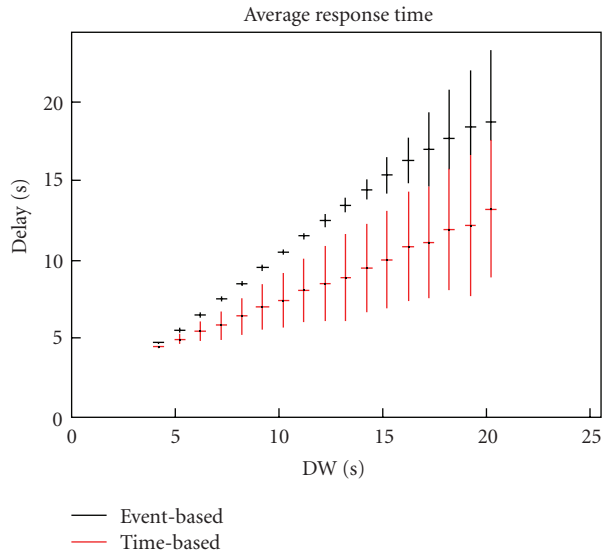


Figure 27: Average response time distribution for Event-based and Time-based DAQ models as a function of the DW width.

Following this argument we model an event-based DAQ where the DW is opened at the arrival of a report from Node 0.

9.2. Simulation Results

9.2.1. The Effects from the Detection Window. In this simulation study we focus on the system performances as a function of the detection window width and the adopted DAQ scheme. We consider CSMA/CA for medium access and First Come First Served (FCFS) for task scheduling at the nodes kernels.

In Figure 26 the system efficiency is plotted against the DW size. For event-based DAQ, the efficiency is maximum when DW is of the order of the path cover time (S/V = 6 seconds). As expected the performances degrade as the DW size increase and thus reports coming from independent detections are mixed up.

If we adopt a time-based DAQ, reports arrivals and DW are uncorrelated. As the DW increases we collect more events reaching the ratio of 70% with DW = 20 seconds.

Depending on the type of DAQ adopted at the coordinator, two different set-points are suggested in this analysis.

As a side effect of the uncorrelation between reports arrivals and DW, the average response time is strictly smaller in the case of time-based DAQ with respect to event-based because for the sub-sample of detected events, the DW results already open at the arrival of the first report thus reducing the response time of the system as shown in Figure 27.

Of course from the design requirements of the visual tracking system (in terms of efficiency and latency) it is possible to select the appropriate DAQ profile privileging either the efficiency or the latency.

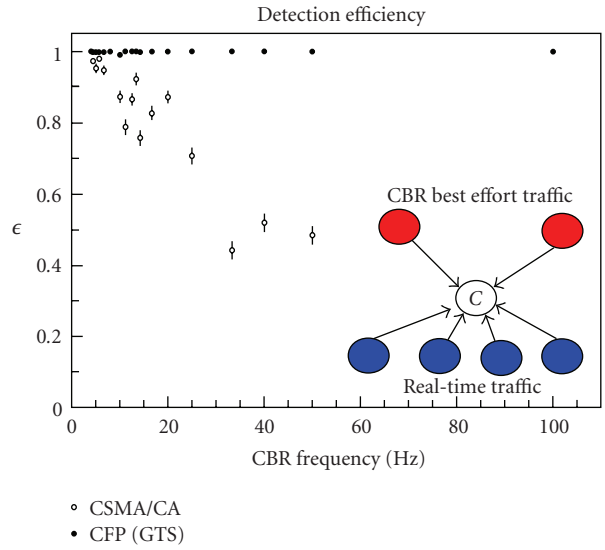


Figure 28: Efficiency as a function of the CBR rate of two disturbing nodes for the case of CSMA/CA (empty markers) and the GTS (filled markers).

9.2.2. The Effects from the Transmission Schedule. Making use of the GTS mechanism support in RTNS we can differentiate real-time and best-effort traffic sources in simulation and assess the benefits of communication over guaranteed band.

In Figure 28 the efficiency (obtained making use of event-based DAQ) is plotted against the rate of disturbance introduced by two nodes generating CBR traffic with tunable frequency. On the x-axis the nominal rate for one node is reported, thus the disturbance rate felt by the controller is actually the double.

The disturbing nodes attempt to inject 100 bytes packet frames into the network at regular intervals. For example, at

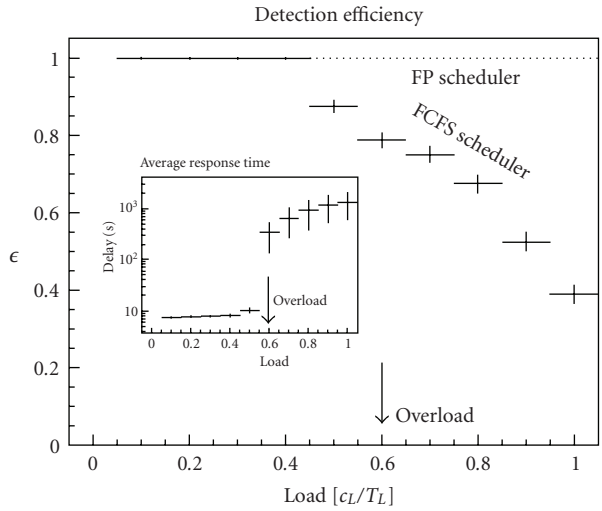


Figure 29: Efficiency as a function of CPU load factor in device nodes for Fixed Priority (FP) and First Come First Served (FCFS). The embedded frame contains the distribution of the Average Response Time in semi-log scale for FCFS. Overload condition occurs when $c_L/T_L = 60\% > 50\%$.

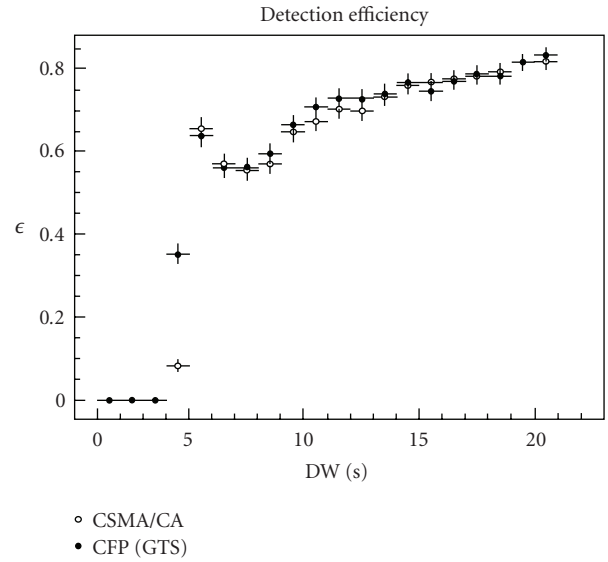


Figure 30: Efficiency behavior as a function of the DW width for the case of CSMA/CA (empty markers) and the GTS (filled markers).

a disturbing frequency of 40 Hz, this translates into 64 Kbps demand having the network 250 Kbps as total capacity.

The nodes are associated to the coordinator and transmissions are done at the same frequency as regulated by the IEEE 802.15.4 standard for the star-shaped networks. As it can be seen (empty markers in the plot), although the working conditions have been selected to produce a fully efficient set-up, this is true only in absence of parallel data flows. Already at a disturbance frequency of 40 Hz, using the CSMA/CA schedule for message transmission, the actual value of the system efficiency is about half of the nominal.

If we schedule the traffic related to visual tracking during the CFP, and the concurrent best effort traffic during the CAP, we permit parallel flows in the network without worsening the performances of the guaranteed services.

In this simple case study, we statically allocate a GTS, 2 slots long (to let the transmission report fit into it), to each device equipped with camera. The system is found insensitive to disturbances and the nominal value for the efficiency (filled markers in the plot) is achieved regardless of any non-real time activity present in the network.

9.2.3. The Effect of the CPU Load. So far the multitasking capabilities of the kernels did not play any role because the "optional" transmission of the report on IPTask completion can be easily coded as an ordinary code branch.

Suppose the device nodes run other activities concurrently with visual tracking (e.g., self-diagnosis, error reporting, etc.). The effects on global metrics depend on the scheduling policy adopted in the kernels, notably upon their preemptive capabilities.

In Figure 29 the system efficiency and latency are tracked against the computational load introduced by a background task scheduled concurrently with the IPTask. The IPTask provides a standalone load of $c_{\mathrm{IP}}/T_{\mathrm{IP}} = 50\%$, so that the overload condition is reached whenever additional load exceeding 50% is scheduled on the node.

If the kernel has no real-time functionality as in the case of TinyOS [24], as the extra load increases, the absolute response time and its jitter increase. Moreover, as the overload condition is met, the system starts missing events and malfunctions show up like that of events being detected after the appearance of many others: in the plot of Latency versus Background load, the trend is discontinuous at the overload condition where response time jumps by two orders of magnitude meaning that the aircraft track is recognized after 300 seconds (5 minutes) from its appearance.

The visual tracking system deployed on top of non real-time kernels does not respect safety critical constraints with respect to background task. This result encourages to adopt real-time kernels like ERIKA [40] and Nano-RK [41] supporting Fixed Priority scheduling whenever the nominal performances must be guaranteed in variable (or even unpredictable) conditions of CPU load.

9.2.4. The Effects from Bandwidth Limitations. When the vehicle inter-arrival times are smaller, events are more frequent and the average network traffic generated by report transmission gets larger. The effect is stronger if the inter-arrival time is comparable with path cover time; in such a case reports related to different vehicles are injected concurrently into the network by different device nodes.

The limitation in bandwidth for the low rate nature of the IEEE 802.15.4 standard prevents the system from the full efficiency. Selecting the most favorable options for the DAQ (the event-based), the system achieves 80% efficiency with a DW equal to 20 s (empty markers in Figure 30). Higher values for the DW have not been explored for reasons related to latency matters.

As it can be seen from the plot, the GTS mechanism does not improve the system performances (filled markers in the plot) and the two curves are statistically compatible. The non-monotonic behavior is explained by the fact that the system starts working properly when the detection window is larger than the path covered time.

10. Conclusions

Next generation WSN applications will present designers and developers additional challenges, as relevant high computational load and timing constraints. Therefore, it is necessary to provide analysis and simulation tools to do early performance estimation.

In this paper, we have presented RTNS, a simulation tool that allows to jointly simulate the effects of network protocols and of RTOS mechanisms in real-time distributed applications. The tool is a combination of two existing open source packages, NS-2 and RTSim, hence it includes models of a wide range of network protocols and RTOS scheduling algorithms.

The tool has been demonstrated on realistic case-studies. We also tested its scalability in terms of simulation time and memory occupancy by increasing the number of simulated nodes.

We are currently working on the integration of RTNS with existing RTOS, such as ERIKA, so to close the gap between simulation model and real application development [8]. In the future, we envision a methodology that can allow the designer of WSNs to automatically generate application code from simulation models into real hardware platforms.

Acknowledgments

The authors would like to thank Claudio Cicconetti, Prashant Batra, Cesare Bartolini, and Francesco Piga for their invaluable support in developing the RTNS framework.

References

[1] "SENSE—Smart Embedded Network of Sensing Entities," http://www.sense-ist.org/.

[2] "CoBIs—Collaborative Business Items," http://www.cobis-online.de/.

[3] "WINSOC—Wireless Sensor Network with Self-Organization Capabilities for critical and emergency applications," http://www.winsoc.org/.

[4] "SCIER—a technological simulation platform to managing natural hazards," http://www.scier.eu/.

[5] "SensAction-AAL—SENSing and ACTION to support mobility in Ambient Assisted Living," http://www.sensaction-aal.eu/.

[6] A. Cervin, M. Ohlin, and D. Henriksson, "Simulation of networked control systems using TrueTime," in *Proceedings of the 3rd International Workshop on Networked Control Systems: Tolerant to Faults*, Nancy, France, June 2007, invited talk.

[7] LAN-MAN Standards Committee of the IEEE Computer Society, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Press, New York, NY, USA, 2003.

[8] M. Chitnis, P. Gai, G. Lipari, P. Pagano, and A. Romano, "Rapid prototyping suite of IEEE 802.15.4-compliant sensor networks," in *Proceedings of the IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS '07)*, pp. 1–3, Pisa, Italy, October 2007.

[9] 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 International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–8, Long Beach, Callif, USA, March 2007.

[10] P. Pagano, M. Chitnis, and G. Lipari, "RTNS: an NS-2 extension to simulate wireless real-time distributed systems for structured topologies," in *Proceedings of the 3rd International Conference on Wireless Internet (WICON '07)*, ACM Press, Austin, Tex, USA, October 2007.

[11] "The RTNS simulation suite," http://rtns.sssup.it.

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

[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*, vol. 32, no. 9, pp. 907–932, 2002.

[14] P. Pagano, F. Piga, G. Lipari, and Y. Liang, "Visual tracking using sensor networks," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)*, p. 28, Rome, Italy, March 2009.

[15] P. Pagano, F. Piga, and Y. Liang, "Real-time multi-view vision systems using wsns," in *Proceedings of the ACM Symposium on Applied Computing (SAC '09)*, pp. 2191–2196, ACM, Honolulu, Hawaii, USA, 2009.

[16] P. Pagano, C. Nastasi, and Y. Liang, "The multivision problem for wireless sensor networks: a discussion about node and network architecture," in *Proceedings of the International Workshop on Cyber-Physical Systems Challenges and Applications in Conjunction with the 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '08)*, Santorini Island, Greece, June 2008, invited talk.

[17] M. Korkalainen, M. Sallinen, N. Kärkkäinen, and P. Tukeva, "Survey of wireless sensor networks simulation tools for demanding applications," in *Proceedings of the 5th International Conference on Networking and Services (ICNS '09)*, pp. 102–106, Valencia, Spain, April 2009.

[18] "The OPNET Simulator," OPNET Technologies, Bethesda, Md, USA, http://www.opnet.com/.

[19] "The GloMoSim simulator," UCLA Computing Laboratory, University of California, Los Angeles, Calif, USA, http://pcl.cs.ucla.edu/projects/glomosim/.

[20] "The PARSEC environment," UCLA Computing Laboratory, University of California, Los Angeles, Calif, USA, http://pcl.cs.ucla.edu/projects/parsec/.

[21] "The QualNet Simulator," Scalable Network Technologies, Culver City, Calif, USA, http://www.scalable-networks.com/.

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

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

[24] "The TinyOS operating," University of California, Berkeley Calif, USA, http://www.tinyos.net/.

[25] T. Henderson, "NS-3 project Goals. Talk given during the "Workshop on NS-2: The IP Network Simulator","

http://www.nsnam.org/tutorials/simutools08/ns-3-tutorial-handouts.pdf.

[26] P. Jurčík, A. Koubâa, M. Alves, E. Tovar, and Z. Hanzálek, "A simulation model for the IEEE 802.15.4 protocol: delay/throughput evaluation of the GTS mechanism," in *Proceedings of the 15th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '07)*, pp. 109–116, IEEE, Istanbul, Turkey, October 2007.

[27] G. F. Lucio, M. Paredes-Farrera, E. F. Jammeh, and M. J. Reed, "Opnet modeler and ns-2—comparing the accuracy of network simulators for packet-level analysis using a network testbed," *WSEAS Transactions on Computers*, vol. 2, no. 3, pp. 700–707, 2003.

[28] S. Ivanov, A. Herms, and G. Lukas, "Experimental validation of the ns-2 wireless model using simulation, emulation, and real network," in *Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN '07)*, pp. 433–444, VDE, Bern, Switzerland, February-March 2007.

[29] J. Zheng and M. J. Lee, "A comprehensive performance study of IEEE 802.15.4," in *Sensor Network Operations*, pp. 218–237, John Wiley & Sons, New York, NY, USA, 2006.

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

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

[32] "The RTSim simulator," http://rtsim.sf.net/.

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

[34] L. F. Friedrich, J. Stankovic, M. Humphrey, M. Marley, and J. Haskins Jr., "A survey of configurable, component-based operating systems for embedded applications," *IEEE Micro*, vol. 21, no. 3, pp. 54–68, 2001.

[35] "802.15.4 and ZigBee Routing Simulation at Samsung/CUNY," http://www-ee.ccny.cuny.edu/zheng/pub/file/WPAN_ZBR_pub.pdf.

[36] "The Flex board," http://www.evidence.eu.com/.

[37] "The Imote2 wireless module," http://www.xbow.com/.

[38] "The Microchip 16-bit dsPIC Digital Signal Controllers," http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=75.

[39] "The Intel Xscale chipset family," http://www.intel.com/design/intelxscale/.

[40] "E.R.I.K.A.," http://erika.sssup.it/.

[41] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-rk: an energy-aware resource-centric operating system for sensor networks," in *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS '05)*, pp. 256–265, Miami, Fla, USA, December 2005.