Using Cooja for WSN Simulations: Some New Uses and Limits

Kévin Roussel
INRIA Nancy Grand-Est
615, rue du Jardin Botanique
54600 Villers-lès-Nancy, France
kevin.roussel@inria.fr

Ye-Qiong Song
LORIA/INRIA Nancy Grand-Est
615, rue du Jardin Botanique
54600 Villers-lès-Nancy, France
ye-qiong.song@loria.fr

Olivier Zendra
INRIA Nancy Grand-Est
615, rue du Jardin Botanique
54600 Villers-lès-Nancy, France
olivier.zendra@inria.fr

Abstract

The Cooja/MSPSim network simulation framework is widely used for developing and debugging, but also for performance evaluation of WSN projects.

We show in this paper that Cooja is not limited only to the simulation of the Contiki OS based systems and networks, but can also be extended to perform simulation experiments of other OS based platforms, especially that with RIOT OS.

Moreover, when performing our own simulations with Cooja and MSPSim, we observed timing inconsistencies with identical experimentations made on actual hardware. Such inaccuracies clearly impair the use of the Cooja/MSPSim framework as a performance evaluation tool, at least for time-related performance parameters.

We will present in this paper, as our contributions: On the one hand, how to use Cooja with projects not related to Contiki OS; On the other hand, the detailed results of our investigations on the inaccuracy problems, as well as the consequences of this issue, and give possible leads to fix or avoid it.

Categories and Subject Descriptors

B4.4 [**Performance Analysis and Design Aids**]: Simulation—*Inaccuracies of timings/delays in simulations*; C2.1 [**Network Architecture and Design**]: Wireless Communication—*Wireless Sensor Networks*; D4.m [**Operating Systems**]: Miscellaneous—*WSN-dedicated OS and simulators/emulators*

General Terms

Simulation, Wireless Sensor Networks, WSN-dedicated OS

Keywords

Cooja / MSPSim, inaccuracies, WSN OS, simulation / emulation, validation, performance evaluation

International Conference on Embedded Wireless Systems and Networks (EWSN) 2016
15–17 February, Graz, Austria

ISBN: 978-0-9949886-0-7

1 Introduction

The research on Wireless Sensor Networks (WSN) and Internet of Things (IoT) often resorts to simulation and/or emulation tools. It is indeed often difficult to have enough devices (or "nodes" or "motes" as one might call them) to perform the large-size tests often needed in this domain; especially when these "motes" need to be instrumented to return enough useful information to design, evaluate or debug projects based on this kind of technology.

Thus, the development of innovative, ambitious projects such as those that are the focus of the *NextMote* workshop will surely need such simulation/emulation tools, at least for the first steps of design and test of new technologies, since the number of involved devices is deemed to grow exponentially (for example: smart dusts, seeds or pollens), and thus testing such a lot of miniaturized devices may be more and more difficult.

There are many of such simulation/emulation tools (like, for example, the OpenSim tool from the OpenWSN project [19], or TOSSIM [13] from TinyOS [14]). But currently, one of the most used—if not *the* most used—simulation/emulation tool used in the WSN/IoT domain is the Cooja framework [16], which includes the MSPSim and Avrora software to perform cycle-exact emulation of "motes".

The present paper is based on the two following subjects :

1. The ability, actually tested and used pervasively, to use Cooja to run programs that are not designed with, nor even related to, the Contiki operating system. Any program running on the microcontroller (MCU) architectures supported by the embedded emulators of Cooja—MSPSim and Avrora—can be simulated (with the use of a trivial trick).
2. The result inaccuracies we discovered while using Cooja simulations for testing our own WSN-based projects (note that we used the version of Cooja provided with Contiki release 2.7). Since Cooja is widely used, especially for evaluating performances of WSN or IoT-based projects, such inaccuracies may have strong consequences on the validity of the many publications making use of this framework.

In this paper, after a brief reminder about Cooja and MSPSim in the following section 2, our contributions firstly consist in describing, in section 3, how the Cooja Framework is

not limited to simulation of Contiki-based projects, but can—thanks to its emulation features—run any program built for one of the systems supported by its embedded emulators.
We then provide—in section 4—in a detailed description of this timing inaccuracy problem, using a significant set of comparisons between simulations and experimentations on hardware; we then use these results to provide clues about the origin of this issue, and possible means to fix or avoid it.
Then in section 5 we see this problem's consequences on current WSN/IoT-related literature, in terms of robustness and reliability of the articles relying on such simulations.
Finally, we draw in section 6 some conclusions from all of these contributions as an end to the present article.

2 Cooja and MSPSim

Provided by the Contiki OS project [7], the Cooja network simulator [16] has become a widely used tool in the domain of Wireless Sensor Networks (WSN). The research community especially uses it extensively to perform simulations of small to relatively large wireless networks of connected devices embedding sensors and/or actuators, commonly named *"motes"*, and thus to develop, debug and evaluate projects based in the WSN technology.

The use of simulations is especially useful for performing virtual runs on large sensor networks, comprising a large number of motes that would be difficult, long and costly to operate with actual hardware.

Cooja itself is a Java-based application, providing three main features:

1. a graphical user interface (GUI, based on Java's standard Swing toolkit) to design, run, and analyze WSN simulations;
2. simulation of the radio medium underlying the wireless communications of sensor networks;
3. an extensible framework, which allows integration of external tools to provide additional features to the Cooja application.

This last feature is used to allow Cooja to actually emulate the various motes constituting the WSNs. It indeed embeds and uses dedicated emulator programs that perform cycle-accurate emulation of the chips with which motes are built: microcontroller units (MCUs), radio transceivers, etc.

This emulation mechanism is one of the main strong assets of Cooja: thanks to it, very fine-grained, precise and low-level simulations can be performed; this is why Cooja has become a tool of choice especially for debugging and evaluating WSN-related software (which happens to be often based on Contiki OS).

Current versions of Cooja make use of two different emulator software packages: Avrora [18] for emulation of Atmel AVR-based devices, and MSPSim [8] for emulation of TI MSP430-based devices.

(Also note that in the future, the structure of Cooja would allow it to include more emulators, dedicated to other architectures: one might especially think to the ARM Cortex-M architecture which is more and more used in "motes" and other embedded devices.)

Of these two emulators, MSPSim is currently the most used in literature including Cooja-based simulations, since motes based on MSP430 MCUs are more commonly distributed and used: we can especially think to the pervasive TelosB/SkyMote family, or to Zolertia's Z1 platform. Since the latter devices are the hardware we use, the present paper will principally focus on the MSPSim emulator.

3 Using Cooja and MSPSim: Not Only for Contiki!

While Cooja has been designed and built by the Contiki project to perform simulations of Contiki-based networks of motes, its structure is actually not tied at all to this operating system.

As explained in the previous section 2, the Cooja Java application itself only provides— besides a GUI—the simulation of the radio medium, which by itself is, of course, totally independent of anything but signal transmission: It only cares about signal strength, propagation, diffusion and interferences. In other words, nothing related to the motes themselves, and *a fortiori* the programs they run is of any slightest influence here.

All the handling of the motes and the programs they run are handled by the emulators software embedded with Cooja, the latter only process the radio signals that the emulated devices emit and receive on the virtual radio medium, according to the way the said medium is simulated. That is: All that happens inside the motes themselves is actually ignored by Cooja itself, and is handled by the emulators.

An emulator software being, by definition, a virtual instance of an hardware piece, it is supposed to execute—as much as possible—the same way that the hardware it emulates. Since motes like the TelosB/SkyMote or Zolertia Z1 are in no way tied to the Contiki OS (they are powered by general-purpose MCUs), nor are their emulators like MSPSim.

Thus, like one can run many OSes like TinyOS, Contiki, RIOT OS [12], Nano-RK [9], etc. on these motes, so should we be able to run the same variety of operating systems on these motes' emulators.

Like one will see in the next sections of the present paper, we were able to run RIOT OS applications using Cooja/MSPSim without problem. We are also convinced that running other OSes like, for example, Nano-RK under Cooja simulations would work as well, provided the emulated motes are supported by the versionsof MSPSim and Avrora provided with Cooja.

To run our RIOT OS applications under Cooja, we only had to perform one very simple trick: MSPSim expects to run executables whose name extension corresponds to the emulated platform; for example: "`executable.sky`" for the SkyMote/TelosB family, or "`executable.z1`" for the Zolertia Z1. The Contiki build system (`Makefiles`) is designed to give to the compiled executables such ad-hoc extensions automatically. RIOT OS build system, on the contrary, always produces executables named like "`executable.elf`" (since the compilation toolchain, which happens to be the standard GNU toolchain for both OSes, produces executables in the well-known ELF executable format). To be able to run

RIOT OS executables under Cooka simulations, we thus had to change their extension accordingly to the target platform to respect the Contiki build system naming scheme—Cooja will refuse to load a file as an executable without the appropriate extension. Once this renaming done, running RIOT OS compiled executables with Cooja simulations just works without any problem: Contiki build system, apart from the specific naming scheme, also produces standard ELF executables, built with the appropriate version of the GCC toolchain (i.e.: `gcc-msp430` or `gcc-avr`, according to the platform). Note that we were able to use the standard Debian-packaged version of these cross-compilers to create executables for both Contiki OS and RIOT OS without any problem.

To sum it up, the Contiki build system produces standard ELF-formatted executables, adapted to the target MCU architecture. Any WSN OS producing such standard ELF executables should thus be able to be simulated/emulated as well under Cooja simulations. The only manipulation needed is a simple renaming trick as explained in the previous paragraph, so as to follow Cooja naming scheme linking executable name extension and platform ("`exe.z1`" for the Zolertia Z1, "`exe.sky`" for SkyMote/TelosB, and so on).

If one needs to know the extension that Cooja expects for a given hardware, one should simply compile a simple Contiki example (like the ever classical "`hello-world`") for the wanted hardware, and see what extension the produced exectable has.

4 Timing Inaccuracy Problem in MSPSim

When performing our own simulations with virtual networks of MSP430-based motes, we noticed timing inaccuracies in comparison to experiences made on actual hardware. More precisely, we noticed that our simulations showed unexplained delays during packet transmission (TX) over the radio medium, that weren't observed during similar experiences on physical motes.

We searched the literature, but we found no article related to any reported inaccuracy in Cooja or MSPSim.

We then investigated the problem, and discovered the following results.

The problem is with the emulation of MSP430-powered, radio-enabled WSN devices (a.k.a. "motes") by the MSPSim software package.

The differences appear on one peculiar operation: when loading packet data into the transmission (TX) buffer of the emulated radio transceiver: MSPSim, when emulating the mote, performs this TX buffer loading at a different speed than the actual hardware.

Consequently, we wrote a simple test program, whose only role is to send data packets of various sizes, chosen amongst:

- *moderate* size, with a payload of 30 bytes;
- *medium* size, with a payload of 60 bytes;
- *large* size, with a payload of 110 bytes (that is: near the maximum size of 127 bytes for IEEE 802.15.4 packets).

The actually transmitted packet have 11 bytes of overhead (headers and checksum) beyond their payload.

This program sends consecutively 50 packets of the chosen size, at the rate of 1 packet per second, for each run: we thus computed the mean and standard deviation for such a group of 50 packets for each setup. The measured value is the duration or "delay" taken to load one of these packets into the TX buffer of the radio transceiver.

In addition, many runs have been executed for each setup, so as to verify the stability of the results.

It has been compiled for, and run on the following hardware platforms:

- the well-known SkyMote/TelosB mote, powered by a MSP430F1611 MCU;
- the more recent Zolertia Z1 mote powered by a MSP430 F2617 MCU.

Both devices have the same CC2420 radio transceiver.

These values are given for different operating systems—the well-known Contiki OS, as well as the more recent RIOT OS [12] which is also specialized in WSN—as well as for two kinds of SPI drivers:

- a standard SPI model, which waits for every transmitted byte to be validated by the hardware SPI interface before sending the next one: we also call it the "safe" SPI access model, since it allows to detect any problem that can occur during transmission on the SPI bus; this is the write method used by default by the SPI driver of RIOT OS;
- a so-called "fast write" SPI model, where a byte is written to the bus every time the SPI hardware TX register is empty, without waiting for the validation signals for the previous byte to be returned; this is the write method used by the SPI driver of Contiki OS. While this allows for faster writes on the SPI bus, it makes transfers—at least in theory—less reliable.

For completing the results obtained with default SPI drivers for both OSes, we modified the RIOT OS SPI driver to make it use the "fast write" SPI model, and thus obtained an additional setup for our tests.

The results of the comparison between execution on simulated motes in Cooja/MSPSim (in the version from Contiki release 2.7) and on physical hardware are shown in Table 1 for SkyMote/TelsoB hardware, and in Table 2 for Zolertia Z1 hardware.

Note that in both tables, delay values are given in "ticks" of timers incrementing at a rate of 32,768 Hz. The unit is thus a fixed period of time equal to about 30.5 microseconds.

The results with "standard" setups show that:

- for the Z1 hardware platform, the results are just catastrophic: the difference between experimental and simulation values represent an overestimation that amounts from 100% to almost 200% of the actual delay! Such overestimated values are clearly unusable for performance evaluation purposes;
- for the SkyMote/Telosb, the differences are much smaller, but still above 10% for Contiki OS, and even 15% for the RIOT OS setup, which is not negligible for an accurate performance evaluation work; moreover, these differences can go in both directions (that

Table 1. Delays Observed for Loading One Packet into CC2420 TX Buffer of a SkyMote/TelosB Mote, Using Various Software Setups.

Results with Contiki OS

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	6.4	0.50	7.2	0.40	–0.8 ticks	($\approx$ 24 μsec.)	$\approx$ 11% exp. value
Medium	10.7	0.46	12.7	0.48	–2.0 ticks	($\approx$ 60 μsec.)	$\approx$ 15% exp. value
Large	18.0	0.00	20.6	0.49	–2.6 ticks	($\approx$ 80 μsec.)	$\approx$ 13% exp. value

Results with RIOT OS (standard "safe" SPI driver)

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	58.0	0.00	50.3	0.46	7.7 ticks	($\approx$ 235 μsec.)	$\approx$ 15% exp. value
Medium	85.2	0.39	73.6	0.50	11.6 ticks	($\approx$ 355 μsec.)	$\approx$ 16% exp. value
Large	131.2	0.39	111.5	0.51	19.7 ticks	($\approx$ 601 μsec.)	$\approx$ 18% exp. value

Results with RIOT OS *and* modified "fast" SPI writes

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	39.2	0.39	38.4	0.49	0.8 ticks	($\approx$ 24 μsec.)	$\approx$ 2% exp. value
Medium	53.2	0.39	52.8	0.40	0.4 ticks	($\approx$ 12 μsec.)	$\approx$ 1% exp. value
Large	76.2	0.39	75.2	0.39	1.0 ticks	($\approx$ 31 μsec.)	$\approx$ 1% exp. value

is: under- and over-estimation), which makes the timing inaccuracy of simulations quite unpredictable and thus hard to estimate and correct without a comparison with experimentation on actual hardware.

With the modified RIOT OS setup ("fast SPI"), we can see that:

- there is absolutely no accuracy improvement for the Z1 platform, the timing results obtained are still overestimated by about 170% of the actual value!
- for the SkyMote/TelosB platform, the situation improves, leading to the obtention of quite accurate results (2% or less of inaccuracy).

This gives us leads about the cause of this inaccuracy problem: since it is largely dependent on the hardware platform simulated *and* the method used to write on the SPI bus, *it is probably not (mainly) due to the emulation of the CC2420 transceiver chip, but rather to the estimation of the timing of SPI bus transfers, made at the MCU level*.

In that case, it is then obvious that:

a.] the emulation of the MSP430F2617 (the MCU powering the Z1 mote) just overestimates largely the SPI delays ;

b.] the emulation of the MSP430F1611 (from SkyMote/TelosB) performs better on that aspect.

We can guess that the MSPSim software has probably been finely tuned for MSP430F1611 emulation — especially for timing calibrations —, while MSP430F2617 emulation has been less thoroughly tested.

We thus see that whatever the software environment used, we can make the following observations about the inaccuracy in timing for the TX buffer loading operation:

- for the Zolertia Z1 hardware platform, this inaccuracy is just huge: the simulated loading delay is always 2 to 3 times larger than the actual delay on hardware, whatever the setup (OS and SPI driver implementation)!
- for the SkyMote/TelosB hardware platform, the inaccuracy is much less important in absolute value, but less predictable, since it can go in both senses: under-estimation or over-estimation of the actual delay on hardware. Moreover, this inaccuracy can go up to 15% of the real value with Contiki OS (and even more when running RIOT OS) which is not a negligible fraction.

This just makes the time-related results obtained with COOJA/ MSPSim simulations unreliable for performance evaluation purposes, particularly on Zolertia Z1, but also on the very popular SkyMote/TelosB hardware platform.

When computing the relative weight of TX buffer loading in total packet transmission duration, we see that the TX load delay always represent at least 10% —up to more than 50%—of the total duration taken to send a packet, especially when running an OS different from Contiki (see Table 3 for detailed results). Such an inaccuracy in the evaluation of packet TX delay is obviously bound to have strong consequences on the reliability of performance evaluations made with Cooja/MSPSim simulations.

Moreover, since the problem seems to be caused by the emulation of MCUs by the MSPSim emulator, we expect the vast majority of the other MSP430-based motes and evaluation boards emulated by the Contiki 2.7 version of Cooja/MSPSim framework (i.e.: latest release at the moment we are writing this paper) to be also similarly impacted by this problem, at various levels.

Consequently, we believe that users of such devices should perform their own tests to evaluate the possible inaccuracy and its impact on their work. To help in that matter, we freely provide the programs we used to perform the present work at the following URL:

`https://github.com/rousselk/tim-inacc-tst-prg`

Table 2. Delays Observed for Loading One Packet into CC2420 TX Buffer of a Zolertia Z1 Mote, Using Various Software Setups.

Results with Contiki OS

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	5.0	0.14	2.3	0.44	2.8 ticks	($\approx$ 84 μsec.)	$\approx$ 122% exp. value
Medium	8.9	0.27	4.2	0.37	4.8 ticks	($\approx$ 145 μsec.)	$\approx$ 114% exp. value
Large	14.0	0.14	7.2	0.39	6.8 ticks	($\approx$ 209 μsec.)	$\approx$ 95% exp. value

Results with RIOT OS (standard "safe" SPI driver)

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	46.0	0.00	16.2	0.39	29.8 ticks	($\approx$ 910 μsec.)	$\approx$ 184% exp. value
Medium	69.0	0.00	24.2	0.39	44.8 ticks	($\approx$ 1368 μsec.)	$\approx$ 185% exp. value
Large	106.8	0.39	38.0	0.00	68.8 ticks	($\approx$ 2100 μsec.)	$\approx$ 181% exp. value

Results with RIOT OS *and* modified "fast" SPI writes

Pkt. Size	Cooja Simulation (ticks)		Hardware Experiment (ticks)		Mean Difference		
	Mean	Std. Dev.	Mean	Std. Dev.			
Moderate	27.0	0.00	10.0	0.00	17.0 ticks	($\approx$ 519 μsec.)	$\approx$ 170% exp. value
Medium	35.0	0.00	13.2	0.39	21.8 ticks	($\approx$ 665 μsec.)	$\approx$ 166% exp. value
Large	49.0	0.00	18.2	0.39	30.8 ticks	($\approx$ 941 μsec.)	$\approx$ 170% exp. value

Table 3. Relative Weight of TX Buffer Loading in Packet Transmission Timings.

HW Platform	WSN OS	Pkt. Size	Loading delay	TX delay	Total Delay	Loading / Total (percentage)
SkyMote/TelosB	Contiki	Moderate	196	1312	1508	13%
SkyMote/TelosB	Contiki	Medium	327	2272	2599	13%
SkyMote/TelosB	Contiki	Large	549	3872	4421	12%
SkyMote/TelosB	RIOT OS	Moderate	1770	1312	3082	57%
SkyMote/TelosB	RIOT OS	Medium	2599	2272	4871	53%
SkyMote/TelosB	RIOT OS	Large	4003	3872	7875	51%
Zolertia Z1	Contiki	Moderate	153	1312	1465	10%
Zolertia Z1	Contiki	Medium	272	2272	2544	11%
Zolertia Z1	Contiki	Large	428	3872	4300	10%
Zolertia Z1	RIOT OS	Moderate	1404	1312	2716	52%
Zolertia Z1	RIOT OS	Medium	2106	2272	4378	48%
Zolertia Z1	RIOT OS	Large	3260	3872	7132	46%

All delays in this table are in microseconds.
TX delays are computed using standard 802.15.4 rate (32 μsec. per byte), with 11 bytes of overhead per packet.

5 Consequences

We looked in recent literature, and found many recent articles (that is: published in year 2014 or later) relying on Cooja simulations to perform, directly or indirectly, time-related performance evaluation of WSN projects based on 802.15.4 networking.

Among them, we can see that most of these papers use virtual SkyMote/TelosB nodes like [15], [11], [1], [6], [2], [5], [3], and [10]; while some other use different, specific MSP430-based motes (e.g.: EXP5438 for [17], or WisMote for [4]) whose sensibility to the present inaccuracy problem is unknown to us.

We also saw one paper that relies on both simulations and experimental results for timing-related performance evaluations: we especially found [20], which also presents purely numerical, MATLAB-produced results. We expect such articles, (much) less common, to be less sensible to simulation inaccuracies.

Almost all of these papers present work related to higher layers of WSN network stacks, especially routing protocols (like [15], [11], [2], and [3]) or application-level protocols (e.g.: [1], [6], [10], and [17]). Some can even present alternate network stacks ([5]).

Such studies, which rely on Cooja/MSPSim emulation runs to evaluate time-related performance on WSN, have a (potentially high) risk of suffering from inexact results due to the timing inaccuracy described hereabove, which may go from moderate up to critical bias. in the latter case, the issue obviously goes as far as putting the discussions and conclusions of these papers in jeopardy.

Due to the paper size constraints, we can't go into further details about actual consequences on previous works. Future work should be undertaken to go deeper into that matter, as well as to check for potential changes about these problems in the newly released version of Cooja provided with Contiki version 3.0.

6 Conclusion

In the present article, we provided the following contributions:

- We proved that the Cooja Framework is in no way limited to simulations of systems running Contiki OS, but can—thanks to the use of embedded software to perform cycle-exact emulation of devices— run any program designed for one of the emulated architectures, whatever the OS used (or not).
- We clearly showed that MSPSim emulation (at least for the version provided with Contiki release 2.7) suffers from a serious timing inaccuracy issue concerning SPI bus access, which especially impacts communication between MCUs and radio transceivers—and thus, wireless network operation—on WSN motes;
  we described the extent of the problem with detailed experimentation results—extent which happens to be serious, especially for the emulation of the Zolertia Z1 hardware platform;
  we provided serious clues about the cause of the issue, and consequently showed the area to investigate for fixing it.
- We briefly enumerated a list of many recently published articles in the WSN domain whose results are potentially negatively impacted by this problem, because they rely on Cooja/MSPSim simulations to evaluate their work. The validity of such publications may be put in jeopardy by the issue we describe here, especially when time-related results obtained by simulation are involved.

Until a fix is made and published to correct this timing inaccuracy in the MSPSim emulator, we believe the only way to get robust and reliable results concerning timing and time-related performance evaluation is to perform tests on actual hardware, which will eliminate any bias that can be introduced by inaccuracies in simulation.

Of course, such hardware tests are much more difficult, long and costly to prepare, run and analyze.

Note that many published articles in the domain of WSNs, including very recent publications, include simulations made with the Cooja/MSPSim framework. This is clearly a testimony of the usefulness of the this software package. That's why fixing the problem we describe in the present paper by correcting MSPSim code as soon as possible is really important.

Note also that while the issue described here can impair the use of Cooja/MSPSim as a performance evaluation tool, it does not affect its other applications, like the ability to develop and debug WSN-related software much more easily thanks to its emulation features.

7 Acknowledgements

This work has been funded by the French national PIA (*« Programme d'Investissement d'Avenir »*) LAR *(Living Assistant Robot)*.

8 References

[1] M. Amoretti, O. Alphand, G. Ferrari, F. Rousseau, and A. Duda. DINAS: A Distributed Naming Service for All-IP Wireless Sensor Networks. In *WCNC 2014*, pages 2781–2786, April 2014.

[2] E. Ancillotti, R. Bruno, and M. Conti. Reliable Data Delivery With the IETF Routing Protocol for Low-Power and Lossy Networks. *Industrial Informatics, IEEE Transactions on*, 10(3):1864–1877, August 2014.

[3] E. Ancillotti, R. Bruno, M. Conti, E. Mingozzi, and C. Vallati. Trickle-L^2: Lightweight Link Quality Estimation through Trickle in RPL Networks. In *WoWMoM 2014*, pages 1–9, June 2014.

[4] M. Antonini, S. Cirani, G. Ferrari, P. Medagliani, M. Picone, and L. Veltri. Lightweight Multicast Forwarding for Service Discovery in Low-Power IoT Networks. In *SoftCOM 2014*, pages 133–138, September 2014.

[5] B. T. de Oliveira, C. B. Margi, and L. B. Gabriel. TinySDN: Enabling Multiple Controllers for Software-Defined Wireless Sensor Networks. In *LATINCOM 2014*, pages 1–6, November 2014.

[6] B. Djamaa, M. Richardson, N. Aouf, and B. Walters. Towards Efficient Distributed Service Discovery in Low-Power and Lossy Networks. *Wireless Networks*, 20(8):2437–2453, November 2014.

[7] A. Dunkels, B. Grönvall, and T. Voigt. Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *IEEE 29th Conference on Local Computer Networks*, LCN '04, pages 455–462. IEEE Computer Society, November 2004. `http://www.contiki-os.org/`.

[8] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and J. Marrón. COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, pages 27:1–27:7. ICST, March 2009.

[9] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-RK: an energy-aware resource-centric RTOS for sensor networks. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS'05, pages 265–274. IEEE, December 2005. `http://nanork.org/`.

[10] E. Felemban, A. Sheikh, and M. Manzoor. Improving Response Time in Time Critical Visual Sensor Network Applications. *Ad Hoc Networks*, 23:65–79, December 2014.

[11] O. Gaddour, A. Koubaa, R. Rangarajan, O. Cheikhrouhou, E. Tovar, and M. Abid. Co-RPL: RPL Routing for Mobile Low Power Wireless Sensor Networks Using Corona Mechanism. In *SIES 2014*, pages 200–209, June 2014.

[12] O. Hahm, E. Baccelli, M. Günes, M. Wählisch, and T. C. Schmidt. RIOT OS: Towards an OS for the Internet of Things. In *INFOCOM 2013, Poster Session*, April 2013. `http://www.riot-os.org/`.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137. ACM, November 2003.

[14] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*, pages 115–148. Springer, Berlin Heidelberg, 2005. `http://www.tinyos.net/`.

[15] B. Marques and M. Ricardo. Improving the Energy Efficiency of WSN by Using Application-Layer Topologies to Constrain RPL-Defined Routing Trees. In *MED-HOC-NET 2014*, pages 126–133, June 2014.

[16] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with Cooja. In *IEEE 31st Conference on Local Computer Networks*, LCN '06, pages 641–648. IEEE Computer Society, November 2006.

[17] S.-H. Seo, J. Won, S. Sultana, and E. Bertino. Effective Key Management in Dynamic Wireless Sensor Networks. *Information Forensics and Security, IEEE Transactions on*, 10(2):371–383, February 2015.

[18] B. L. Titzer, D. Lee, and J. Palsberg. Avrora: Scalable Sensor Network Simulation with Precise Timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05. IEEE Press, 2005. Article no. 67.

[19] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister. OpenWSN: a standards-based low-power wireless development environment. *Transactions on Emerging Telecommunications Technologies*, 23(5):480–493, August 2012.

[20] X. Wu, K. Brown, and C. Sreenan. Contact Probing Mechanisms for Opportunistic Sensor Data Collection. *The Computer Journal*, 2015.