Using Cooja for WSN Simulations: Some New Uses and Limits

Kévin Roussel Ye-Qiong Song Olivier Zendra

INRIA Nancy Grand-Est — LORIA UMR 7503 — Université de Lorraine

EWSN'2016 NextMote workshop
15 February 2016

Contents

Introduction I

To develop and test large, ambitious WSN-based projects

→ need of powerful and accurate simulation/emulation tools

Many such simulation/emulation tools for WSNs exist:

- OpenSim (from OpenWSN project)
- TOSSIM (from TinyOS project)
- Cooja (from Contiki OS project)

This work focuses on the latter framework: Cooja

→ (one of) the most used WSN simulation/emulation tool

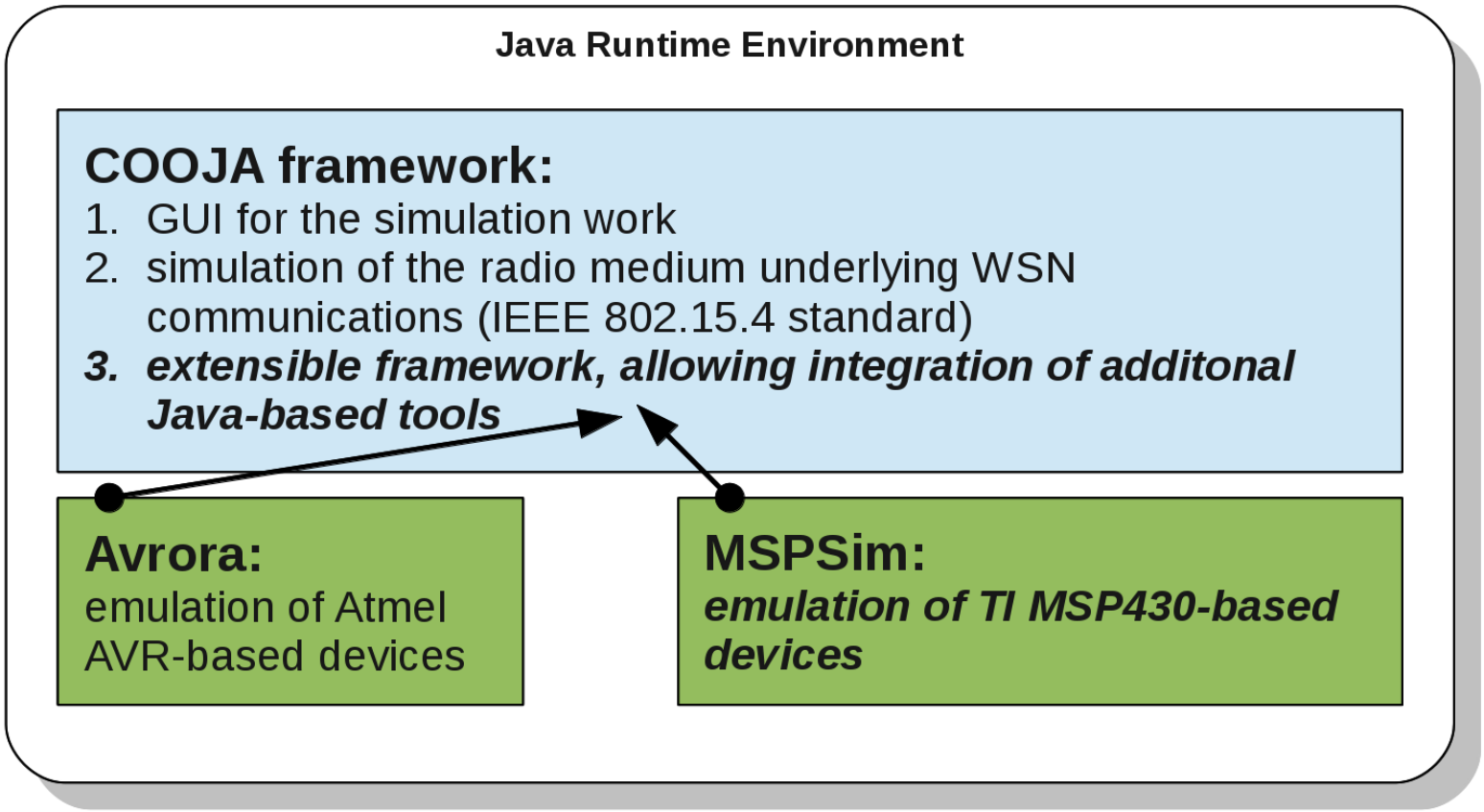
Introduction II

Contributions of our work: we will focus on...

1. the ability—***actually tested and used***—to use the Cooja framework to simulate/emulate systems not related to Contiki OS
2. the inaccuracies in time-related results we discovered while using Cooja[a] to perform our own simulations to test our own WSN-related projects
3. the possible consequences of these inaccuracies in the literature published in the WSN domain

[a] Note we always used the Cooja version provided with Contiki release 2.7

The Cooja framework and its emulators



The present work focuses on Cooja and the MSPSim emulator only

Using Cooja and MSPSim with any WSN system... (or even without)

What do Cooja's embedded emulators run?

The Contiki build system produces standard (ELF format) executables
$\rightarrow$ the Cooja embedded emulators are designed to run such standard executables

What does it mean?

Any system producing such standard ELF executables (besides Contiki OS itself) will run on virtual motes emulated with the Cooja framework
We tested this with MSPSim, using:

- applications based on RIOT OS
- "bare-metal" applications

Any other OS using `msp430-gcc` as its compiler (e.g.: TinyOS) should run fine on Cooja/MSPSim: a simple trick on executables' file extension is enough

Timing Inaccuracy Problem in MSPSim: Setup

What did we discover during our experiments?

- unexplained delays during simulations/emulations of packet transmissions (TX), not observed on real hardware
- the differences appear on one peculiar operation: ***when loading the TX buffer of the CC2420 radio transceiver***

What did we do to investigate that problem?

- we wrote a simple test program, which only sends data packets of various sizes (resp. 30, 60 and 110 bytes of MAC payload)
- we tried this program on various configurations:
  - two different hardware platforms: Sky/TelosB and Zolertia Z1 motes
  - two different WSN OS: Contiki and RIOT OS

Timing Inaccuracy Problem in MSPSim: Results I

$$\textbf{Inaccuracy} = \frac{\text{Simulated delay} - \text{Actual Hardware test delay}}{\text{Actual Hardware test delay}}$$

Observed TX buffer loading delay inaccuracies of MSPSim emulation versus tests on actual hardware

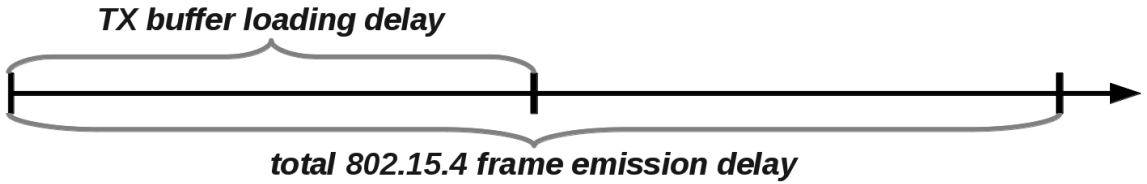
HW platform	OS	Payload size	Inaccuracy
SkyMote/TelosB	Contiki	30 bytes	11%
SkyMote/TelosB	Contiki	60 bytes	15%
SkyMote/TelosB	Contiki	110 bytes	13%
SkyMote/TelosB	RIOT OS	30 bytes	15%
SkyMote/TelosB	RIOT OS	60 bytes	16%
SkyMote/TelosB	RIOT OS	110 bytes	18%
Zolertia Z1	Contiki	30 bytes	122%
Zolertia Z1	Contiki	60 bytes	114%
Zolertia Z1	Contiki	110 bytes	95%
Zolertia Z1	RIOT OS	30 bytes	184%
Zolertia Z1	RIOT OS	60 bytes	185%
Zolertia Z1	RIOT OS	110 bytes	181%

Timing Inaccuracy Problem in MSPSim: Results I

What do our tests show?

- the inaccuracy problem is mainly influenced by the emulated hardware platform
  - ***on Zolertia Z1 motes, timing inaccuracies are catastrophic: delays are overestimated by amounts from 95% to 185%***
  - on Sky/TelosB motes, timing inaccuracies are much less important: delays are under- or overestimated by amounts up to 18%
- the used OS has much less influence on the problem: results tend to be more accurate on Contiki than on RIOT OS
- the size of transmitted packets doesn't seem to have any significative impact on this problem

Timing Inaccuracy Problem in MSPSim: Results II



Relative weight of TX buffer loading in packet transmission delays

HW platform	OS	Payload size	Loading delay / Total delay
SkyMote/TelosB	Contiki	30 bytes	13%
SkyMote/TelosB	Contiki	60 bytes	13%
SkyMote/TelosB	Contiki	110 bytes	12%
SkyMote/TelosB	RIOT OS	30 bytes	57%
SkyMote/TelosB	RIOT OS	60 bytes	53%
SkyMote/TelosB	RIOT OS	110 bytes	51%
Zolertia Z1	Contiki	30 bytes	10%
Zolertia Z1	Contiki	60 bytes	11%
Zolertia Z1	Contiki	110 bytes	10%
Zolertia Z1	RIOT OS	30 bytes	52%
Zolertia Z1	RIOT OS	60 bytes	48%
Zolertia Z1	RIOT OS	110 bytes	46%

Timing Inaccuracy Problem in MSPSim: Results II

What do our tests show *(bis)*?

The TX buffer loading operation weight in the whole packet transmission time is anything but negligible:
This TX buffer loading operation amounts to about half of the total packet transmission delay under RIOT OS (much less on Contiki, where it represents 10% to 13%)
$\rightarrow$ *This is probably due to the way these OSes do manage the SPI bus that links the MCU to the radio transceiver on a mote:*

- *"fast mode" (sending burst of bytes) for Contiki OS*
- *"safe mode" (checking every byte sent) for RIOT OS*

Timing Inaccuracy Problem in MSPSim: Discussion

What can we deduce from these results?

- ***Cooja/MSPSim simulations/emulations do not provide accurate enough timing-related results***, especially on Zolertia Z1 motes
  $\rightarrow$ with this inaccuracy issue, ***the simulations/emulations made with this framework cannot be used for timing evaluation purposes of WSN-based projects***
- since the hardware seems to be the main factor influencing this problem, we suppose its cause is—at least partially—linked to wrong timing calibrations for microcontrollers' emulation

Consequences on WSN Literature

What did we find?

- many recent publications rely on Cooja/MSPSim to perform, directly or indirectly, time-related performance evaluations of their WSN-based projects [1] [2] [3] [4] [5] [6] [7] [8]
- most of these publications are focused on higher layers of WSN network stacks, like routing protocols [1] [4] [6] or application-level projects [2] [3] [7] [8]
  $\rightarrow$ ***their authors may probably be unaware of this problem,*** since they are not focused on low-level details

What does that mean?

Many WSN studies rely (at least partially) on Cooja/MSPSim to evaluate their time-related performances
$\rightarrow$ ***This inaccuracy problem could make their results unreliable, and thus put their conclusions in jeopardy!***

Conclusions I

Our contributions:

1. we showed that ***the Cooja framework is not limited to simulations of systems running Contiki OS, but can run any program designed for the emulated architectures*** (especially MSP430, thanks to MSPSim)
2. we showed that ***the MSPSim emulation—at least for the version provided with Contiki release 2.7—suffers from a serious timing inaccuracy issue;*** we described the extent of the problem, and provided serious clues about its cause(s)
3. we briefly enumerated a (non-exhaustive) ***list of recent publications, showing the magnitude of the negative impact*** this problem can cause

Conclusions II

In summary:

Until a fix is made and published to correct this issue in MSPSim, ***we think that all works based on Cooja/MSPSim simulations to evaluate WSN projects for timing matters should be checked (even partially) by tests on actual hardware***

How to test the accuracy of your own works?

We provided the source of our test programs (for Contiki and RIOT OS) on a public GitHub repository:

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

Conclusions III

Strengths of Cooja/MSPSim:

- the many publications in the domain of WSNs, especially very recent ones, including Cooja/MSPSim simulations, is a testimony of the usefulness of such an emulation framework (especially to test large WSNs, which is often difficult and expensive to with actual hardware)
  $\rightarrow$ ***That's why we strongly believe fixing Cooja/MSPSim is really important, and hope for such a fix to appear soon***
- while the issue described can impair the use of Cooja/MSPSim as a performance evaluation tool, ***it does not affect its other valuable uses, like the ability to develop and debug WSN-related software much more easily thanks to its emulation features***

Thanks for Your Attention

Any questions?

Acknowledgements (Funding)

This work has been funded by the French national PIA [a] **LAR** (*"Living Assistant Robot"*).

[a] *PIA : « Programme d'investissement d'avenir »*

References

[1] B.F. Marques and M.P. 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.