Chapter 24
Reducing Simulation Runtime in Wireless Sensor Networks:
A Simulation Framework to Reduce WSN Simulation Runtime by Using Multiple Simultaneous Instances

Pedro Pinto
Instituto Politécnico de Viana do Castelo and INESC TEC, Portugal

António Alberto Pinto
CIICESI, ESTGF, Politécnico do Porto and INESC TEC, Portugal

Manuel Ricardo
INESC TEC, Faculdade de Engenharia, Universidade do Porto, Portugal

ABSTRACT

Wireless Sensor Networks (WSNs) can be deployed using available hardware and software. The Contiki is an operative system compatible with a wide range of WSN hardware. A Contiki development environment named InstantContiki is also available and includes the Cooja simulator, useful to test WSN simulation scenarios prior to their deployment. Cooja can provide realistic results since it uses the full Contiki's source code and some motes can be emulated at the hardware level. However this implies extending the simulation runtime, which is heightened since the Cooja is single threaded, i.e, it makes use of a single core per instant of time, not taking advantage of the current multi-core processors. This chapter presents a framework to automate the configuration and execution of Cooja simulations. When a multi-core processor is available, this framework runs multiple simultaneous Cooja instances to reduce simulations runtime in exchange of higher CPU load and RAM usage.

DOI: 10.4018/978-1-4666-8823-0.ch024

INTRODUCTION

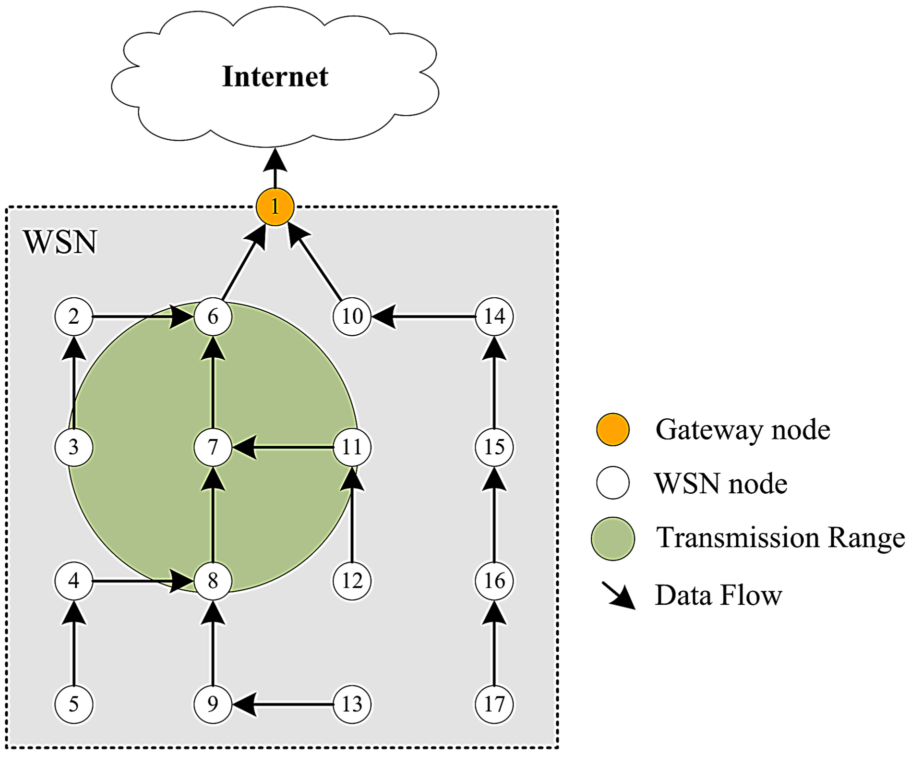
A Wireless Sensor Network (WSN) consists of a large number of sensor nodes communicating with each other, where each node has limited energy and processing resources. Usually, these nodes generate and transport sensed data towards a gateway node, which, in turn, connects these networks to the Internet, as shown in Figure 1.

The WSNs can be deployed in real scenarios using hardware products such as the Z1 ("Z1 mote," n.d.), the SeedEye ("SeedEye," n.d.), the MICAz ("MICAz," n.d.), or the Tmote Sky ("Tmote Sky Project," n.d.). In terms of software, specifically regarding the Operative System (OS), multiple options are available, examples being the Tiny OS ("TinyOS," n.d.), the RIOT OS ("RIOT Operative System," n.d.), and the Contiki ("Contiki OS," n.d.).

To design and test WSN, namely in scenarios using Contiki, the developers may rely on a development environment named InstantContiki which consists of an Ubuntu Linux in a VMware virtual machine with a set of developer tools. Up to date, the latest version of Contiki is 2.7 and it includes the Cooja simulator (Osterlind, Dunkels, Eriksson, Finne, & Voigt, 2006).

The Cooja simulator is a WSN simulator that uses the full Contiki's source code in a set of emulated hardware nodes. While other simulators, such as NS-2 ("NS-2," n.d.) or NS-3 ("NS-3," n.d.), assume that motes are simplified versions of the real hardware, the usage of full Contiki's source code and real hardware emulation allows Cooja to obtain close-to-real results and enables the fast deployment of the simulated experiments directly onto the real motes. However, it also increases simulation complexity

Figure 1. Typical WSN



and simulation runtime. The extension of simulation runtimes can be more expressive when developers need to run multiple simulations in order to obtain statistically sound simulation results, either to test different random topologies or test the same scenario with random seeds. Such rounds of repeated simulations in Monte Carlo experiments can sum up to several hours or even days of simulation runtime.

At the same time, while running simulations, Cooja runs as single-threaded and this means that it uses a single process and a single core at each instant of time. Thus, if Cooja simulator runs within a machine where a multi-core processor is available, these cores will be underused. While there may not be a direct correspondence between the number of cores assigned to the InstantContiki virtual machine, and the number of cores on the real host machine, the underutilization of virtual processing resources leads to the underuse of real processing resources.

The current chapter proposes a simulation framework that automates Cooja simulations and allows Cooja to run multiple simultaneous instances in order to take advantage of multiple virtual cores. The results show that by running multiple simultaneous instances, this simulation framework reduces simulations runtime while requiring a higher CPU load and a higher usage of RAM.

BACKGROUND

Other research efforts have been applied in order to reduce simulation times either by sampling simulation, by dynamically changing simulation detail, or by using multi-threaded workloads in order to take advantage of multi-core processors. In sampling simulations, developers reduce the simulation runtimes by simulating just a small percentage of the overall tested applications, though trying to cover its most relevant portions and obtain close to real results. Using dynamic simulation detail, the developers use different details while the simulation is running, highlighting specific stages of simulations that will be executed with detail (extending simulation runtime) and using other stages executed with low detail (reducing simulation runtime). This technique reduces simulations runtime when compared to constant high detail simulations execution technique.

Developers also design and use multi-threaded workloads when multi-core processors are available, in order to take full advantage of processing and memory resources to reduce simulation runtime when compared to single threaded workloads simulation. In (Kihm & Connors, 2005) authors propose statistical procedures to simulate multi-threaded processor architectures and presented experimental results of multi-threading simulations in a context of processor simulation in computer architectures. In (T.E. Carlson, Heirman, & Eeckhout, 2013) authors propose a multi-threaded application sampling methodology that enables workload reduction and reduces simulation runtimes while accurately predicting an application performance. In (Trevor E. Carlson, Heirman, & Eeckhout, 2011) the authors present results and analysis of experimental studies of multi-threaded workloads running in a set of real hardware. In (Goldsby & Pancerella, 2013) authors proposed a set of changes on Java simulation packages to make use of multiple threads to increase simulation performance in the context of systems with autonomous decision-making entities.

Multiple simulators are available to simulate WSNs. The cross-level WSN simulators allow the simulation of the WSN devices and networks in the different levels of abstraction, from physical to application. In this context, we can enumerate J-Sim ("J-Sim Web site," 2015) and Sensor Network Package, SENS (Sundresh, Kim, & Agha, 2004), and Cooja. The Cooja WSN simulator is distributed with Contiki and, in recent years, this simulator has been updated in many extents. The Table 1 presents the changelog

Table 1. Changelog of Cooja Simulator from Contiki version 2.5 to version 2.7 (actual version)

	Contiki 2.5 (09/2011)	**Contiki 2.6 (07/2012)**	**Contiki 2.7 (11/2014)**
Changelog	The MSPSim/Cooja simulation environment has received a significant speed-up.	• Many improvements to the user interface. • Simulation support for the MSP430x architecture and the exp5438, wismote, and z1 platforms.	• Support for link-layer ACKs • Improved stack monitoring and stack overflow triggering • Improved radiologger • Improved Timeline handling • Support for new RF transceivers emulation • Improved MSPsim support for MSP430x instruction set, verified against hardware

related to Cooja simulator from 2.5 (released in September 9th of 2011) to Contiki 2.7 the up-to-date version of Contiki (released November 15th of 2014) – information from ("Contiki OS," n.d.).

Although significant improvements related to the Cooja simulator have been released as shown in Table 1, since Contiki's version 2.5, the updates are related to internal features and enhanced hardware support. Since WSN developers sometimes face the challenge to design and deploy networks with hundreds or thousands of devices, one of the future improvements regarding Cooja that requires particular attention is the support of multi-core processors, i.e. the capability to run multiple threads in each core at each instance of time ("Contiki Developers Mailing List," 2012). By using the full potential of current processor architectures, the simulation runtimes could be reduced as well as the time elapsed from design to deployment stages. In order to tackle this limitation of Cooja, a simulation framework is proposed in order to reduce simulation runtime by using multiple and simultaneous Cooja instances.

SIMULATION FRAMEWORK

The proposed simulation framework intends to reduce simulation runtime of Cooja simulator by running multiple simultaneous instances. The layout of this framework is presented in Figure 2, where each block is written in perl (except the Cooja Simulator block). It comprises the main functional block, the Main Launcher, where the user provides the simulations the parameters to be used on all simulations either in the form of an argument list or through a configuration file. After that, the Simulator Scheduler will schedule and initiate all simulations, launching a parallel instance for each simulation with its own set of parameters. Each instance is composed of a Simulation Setup and Simulation Execution and Analysis. According to the Simulation Scheduler information, the Simulation Setup generates all simulation parameters in one main configuration file and then starts the code compilation. Then, the Simulation Execution and Analysis starts the Cooja simulation and, in the end of this, analyses the simulation log to generate graphics of the obtained results. At the end of each round of simulations, the Simulation Scheduler may perform a general analysis of logs and generate graphics for groups of finished simulations.

In this simulation framework the simulation runtime is assumed to be the time elapsed between the Simulation Setup start and the end of the Simulation Execution and Analysis, as presented in Figure 3.

The simulation framework has a Simultaneous Instances (SI) mode that, when disabled, will result in the sequential execution of all simulations, where each instance will start after the termination of the previous one, as shown in Figure 4.

Figure 2. Simulation Framework

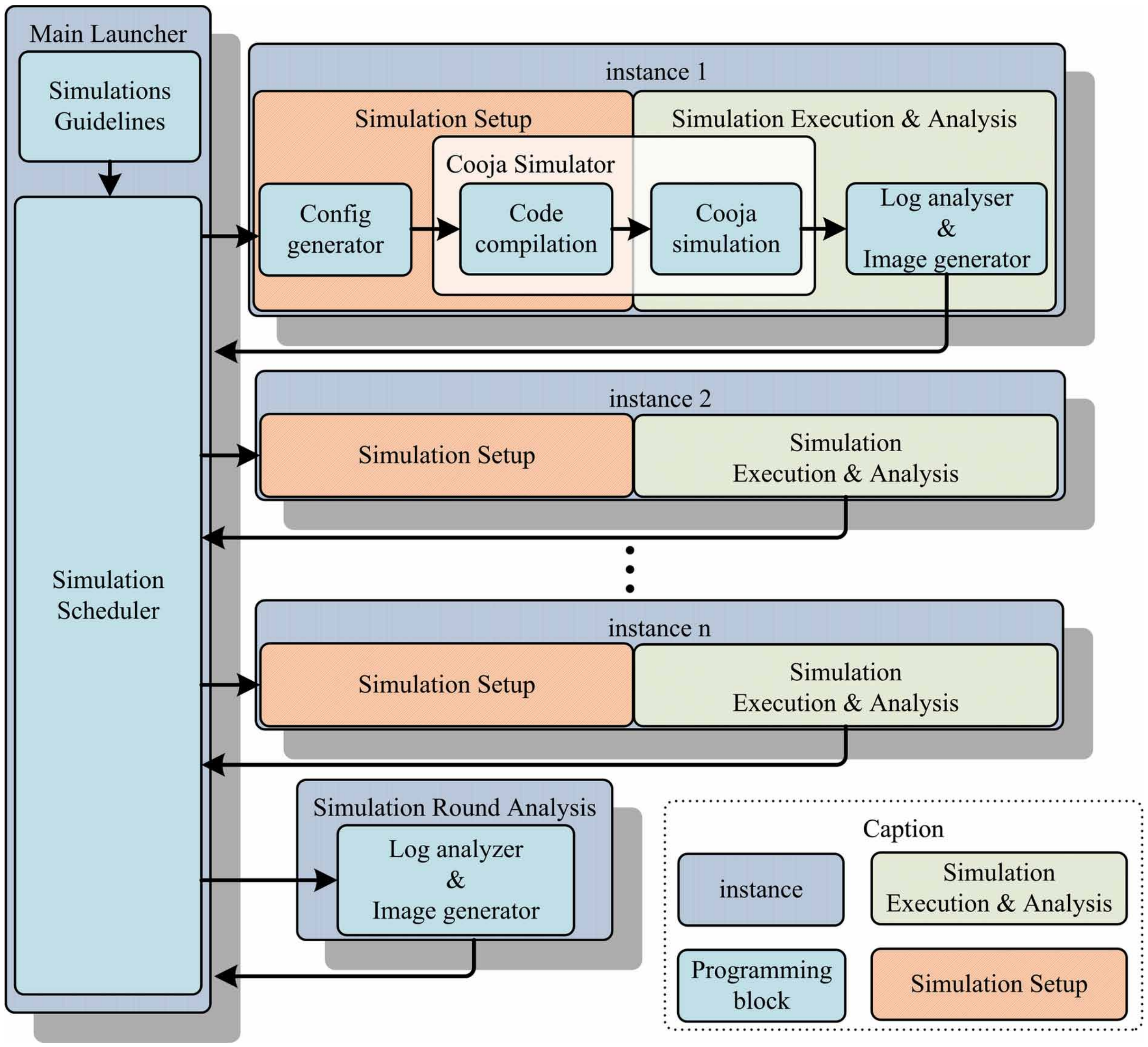


Figure 3. Simulation Runtime

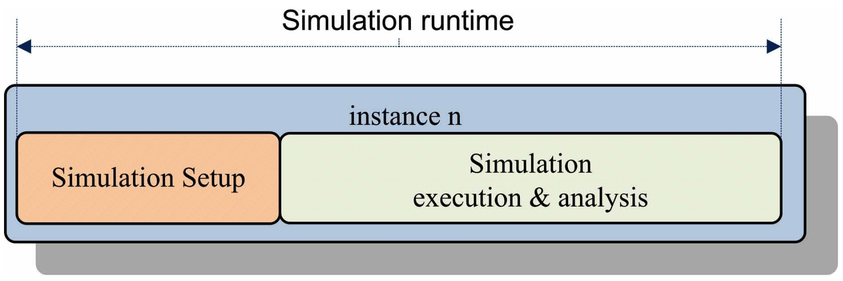
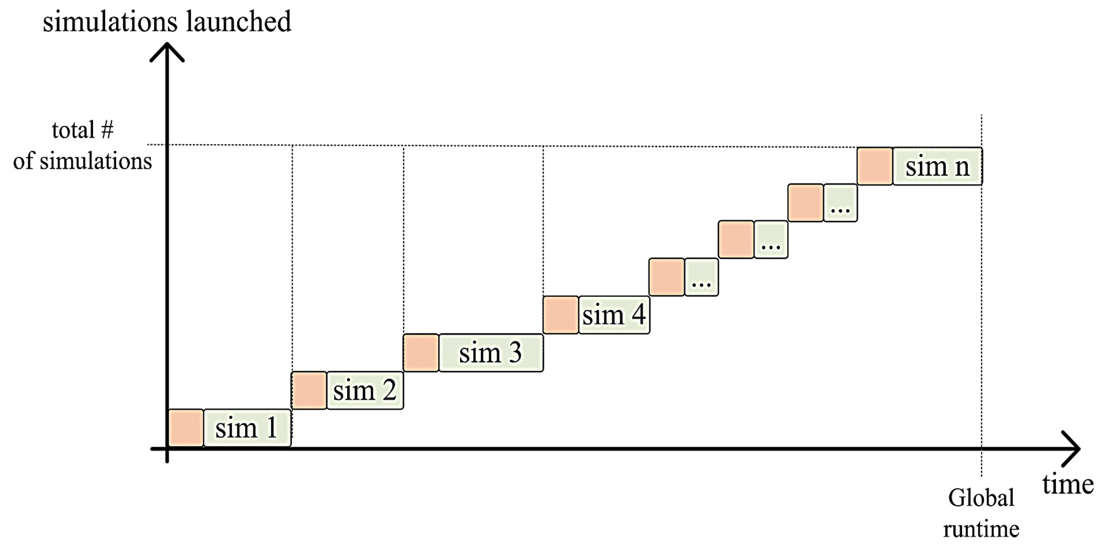


Figure 4. SI disabled



On the other hand, when the SI mode is enabled, the Simulation Scheduler will start each simulation instance independently of the others and up to a maximum SI number (MaxSI), as presented in Figure 5. In this mode, the configuration of a MaxSI value is mandatory and it can be adjusted in order to obtain the maximum reduction of the simulation runtime.

When SI mode is enabled, the simulator scheduler manages the test, initiation and termination of the multiple simulations instances. In this context, a simplified flow diagram of the Simulator Scheduler internal procedure is shown in Figure 6.

Figure 5. SI enabled (MaxSI equals to 4)

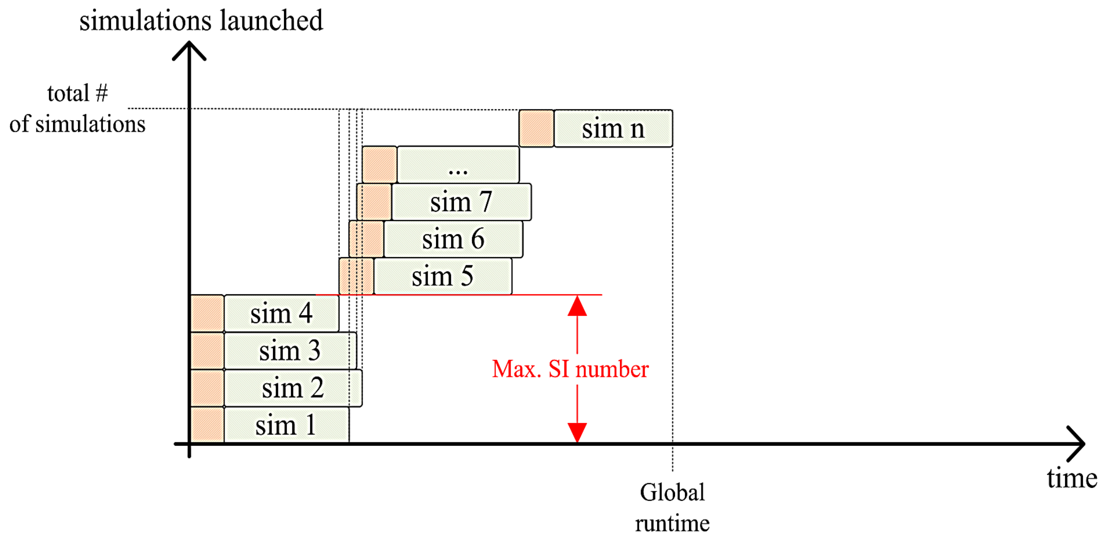
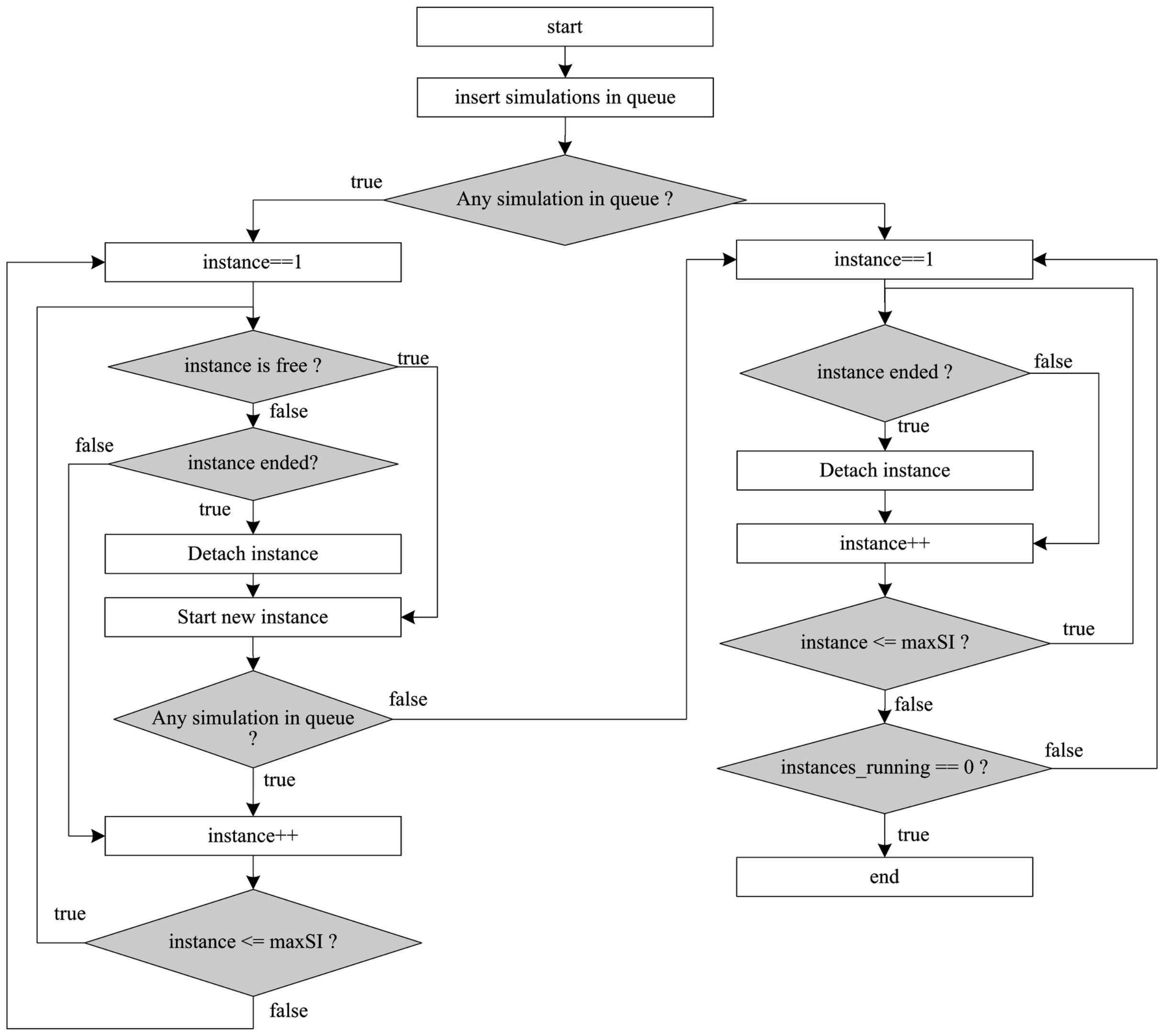


Figure 6. Simulation Scheduler internal procedure (SI mode enabled)



Initially, the simulations to execute are inserted in a queue and then, two different cycles are performed. When the simulations queue is not empty, one cycle tests constantly if any instance already ended and, in case this is true, it starts a new simulation in this instance. The other cycle is performed when there are no simulations in queue to perform. Here, this cycle tests if the instances ended and is there is no simulations running. When no instances are in the running state, the procedure ends. This procedure uses the perl interpreter-based threads in perl (threads module) to manage instances (e.g. start new instances, check if instance ended or check if an instance is free).

Validation Environment

In order to test the presented simulation framework, a WSN simulation was defined using Contiki 2.5 with 10 generator/forwarder nodes and a gateway node, with a simulated time of 60s. The Simulation

Scheduler was configured with SI enabled and tested with MaxSI values ranging from 1 to 6. When the MaxSI value equals to 1, this means there are no simultaneous instances (similar to having SI disabled) and in order to compare this particular case with the others, where SI is greater than 1, the results obtained for MaxSI value equal to 1 were repeated for every SI tested, and named as the default mode. These tests were repeated 10 times each with a total number of 20 simulations using different seeds and values for the global runtime, the runtime per simulation, the CPU load, and the RAM usage were collected and their average values and standard deviations were calculated.

The simulation framework was tested in an Ubuntu Linux (12.04 LTS) 32 bits system with 2GB of RAM in a VMware virtual machine, VMware® player 6 ("VMware," n.d.). The virtual machine processor emulated an Intel® Core™ i7-2620M CPU @ 2.70GHz processor ("Intel® Core™ i7-2620M Processor," n.d.), with 2 cores, each one with 2 virtual cores, resulting in a total of 4 virtual cores. The simulation framework was tested using different number of virtual cores configured in the VMware virtual machine: initially were used 2 virtual cores and then, 4 virtual cores (the number of virtual cores in a VMware virtual machine can be configured up to the number of cores presented in the real CPU; 4 cores in this case).

Results

The results collected for the average global runtime and the average runtime per simulation are presented in Table 2.

Figure 7 shows the results obtained for the average global runtime for the MaxSI values tested for each scenario. These results show that the average global runtime is lower when the SI mode is enabled and the best absolute results are obtained when 4 virtual cores are used. When using a MaxSI value of 4 and 4 virtual cores, the average global runtime reduced from 2300s, in default mode, to 1200s (approx.).

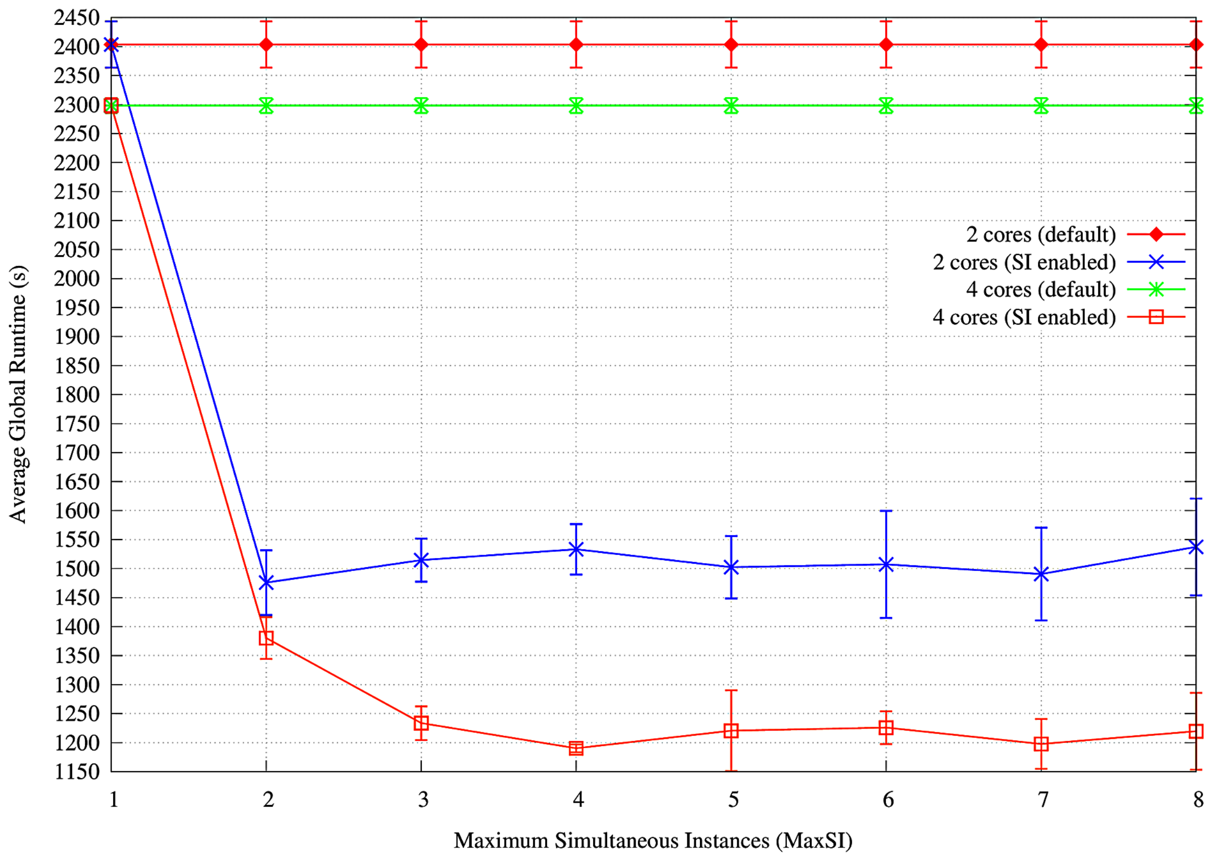
Figure 8 shows the values obtained for the average runtime per simulation and for the tested MaxSI values. These results show that a higher MaxSI value will result in a higher runtime per simulation. Considering the results obtained for a MaxSI value of 2 or more, the real runtime per simulation can be approximated to a linear function. If so, we can see that for 4 cores the linear function presents a slight lower slope than the one of the 2 cores.

A virtual runtime per simulation variable was used to verify the effect of the usage of the simultaneous instances. Table 3 presents the Average Virtual Runtime per simulation and Simulation Time ratio. The Average Virtual Runtime per simulation was obtained by:

Table 2. Average Global Runtime and Average Runtime per simulation

	Average Global Runtime (s)		Average Runtime per Simulation (s)	
MaxSI	2 Cores	4 Cores	2 Cores	4 Cores
1	2403.38	2298.27	119.23	114.01
2	1475.74	1380.29	145.58	136.06
3	1514.61	1233.50	217.77	176.27
4	1533.27	1190.14	303.88	235.20
5	1502.36	1220.52	371.85	301.59
6	1507.13	1225.83	421.29	337.89
7	1490.49	1197.67	492.45	394.28
8	1537.26	1219.50	551.98	429.32

Figure 7. Average Global Runtime



$$Average\ Virtual\ Runtime\ per\ simulation = \frac{Average\ Global\ Runtime}{Total\ number\ of\ simulations} \tag{1}$$

The Simulation Time ratio was obtained by:

$$Simulation\ Time\ ratio = \frac{Simulated\ time}{Virtual\ Runtime\ per\ instance} \tag{2}$$

where the simulated time was constant and equal to 60s in all scenarios.

Figure 9 shows the Average Virtual Runtime per simulation. A greater reduction is achieved when using a MaxSI value that is equal to the number of virtual cores. Moreover, the lowest value, 59 seconds approximately, is obtained when the MaxSI is set to 4, which is the number of virtual cores. This was an expected result since each instance will run in its virtual core and thus, increasing the MaxSI number to a value higher than the number of virtual cores, will result in loss of efficiency due to concurrency within each virtual core. Moreover, the results also show that with the increase of the MaxSI value, there is an increase of the standard deviation of the obtained results.

Figure 8. Average Runtime per Simulation

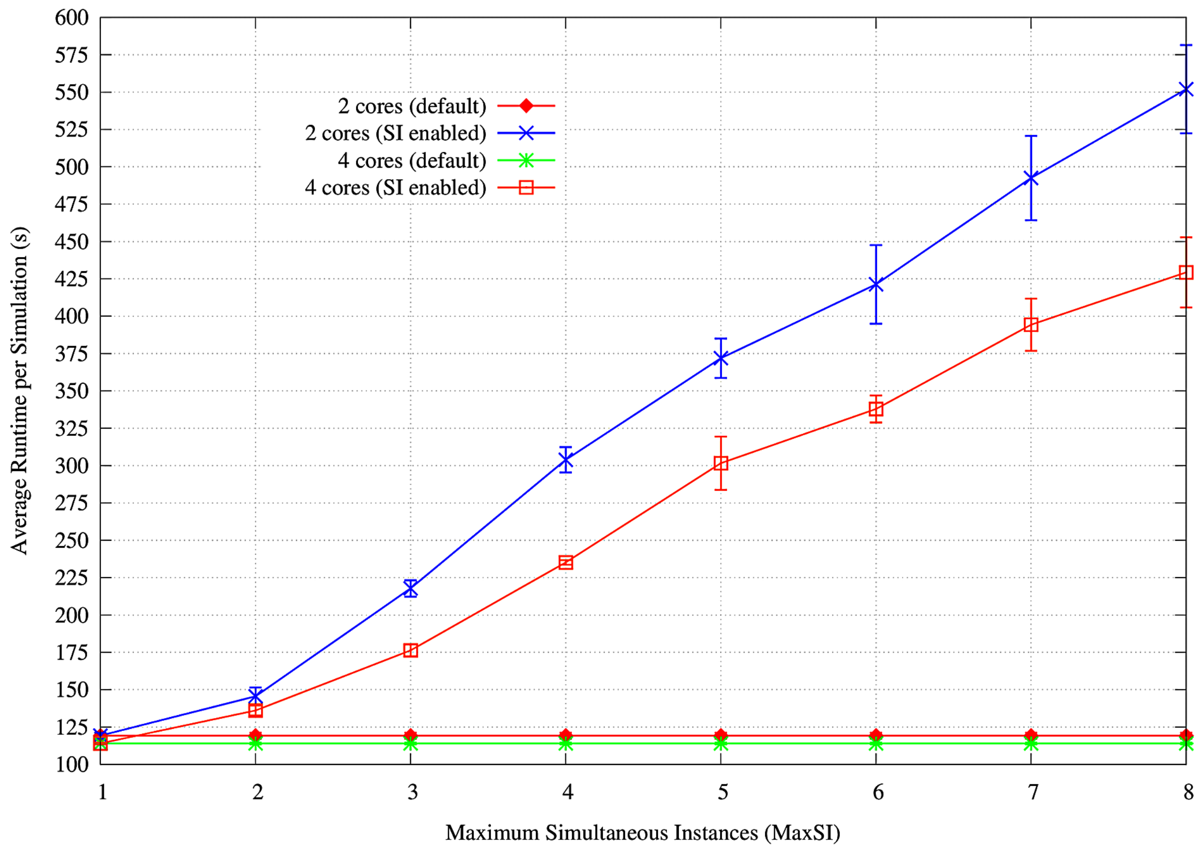


Table 3. Average Virtual Runtime per simulation and Simulation time ratio

	Average Virtual Runtime per Simulation (s)		**Simulation Time Ratio (%)**	
MaxSI	**2 Cores**	**4 Cores**	**2 Cores**	**4 Cores**
1	120.17	114.91	50	52
2	73.79	69.01	81	87
3	75.73	61.68	79	97
4	76.66	59.51	78	101
5	75.12	61.03	80	98
6	75.36	61.29	80	98
7	74.52	59.88	81	100
8	76.86	60.98	78	98

Figure 9. Virtual Runtime per Simulation

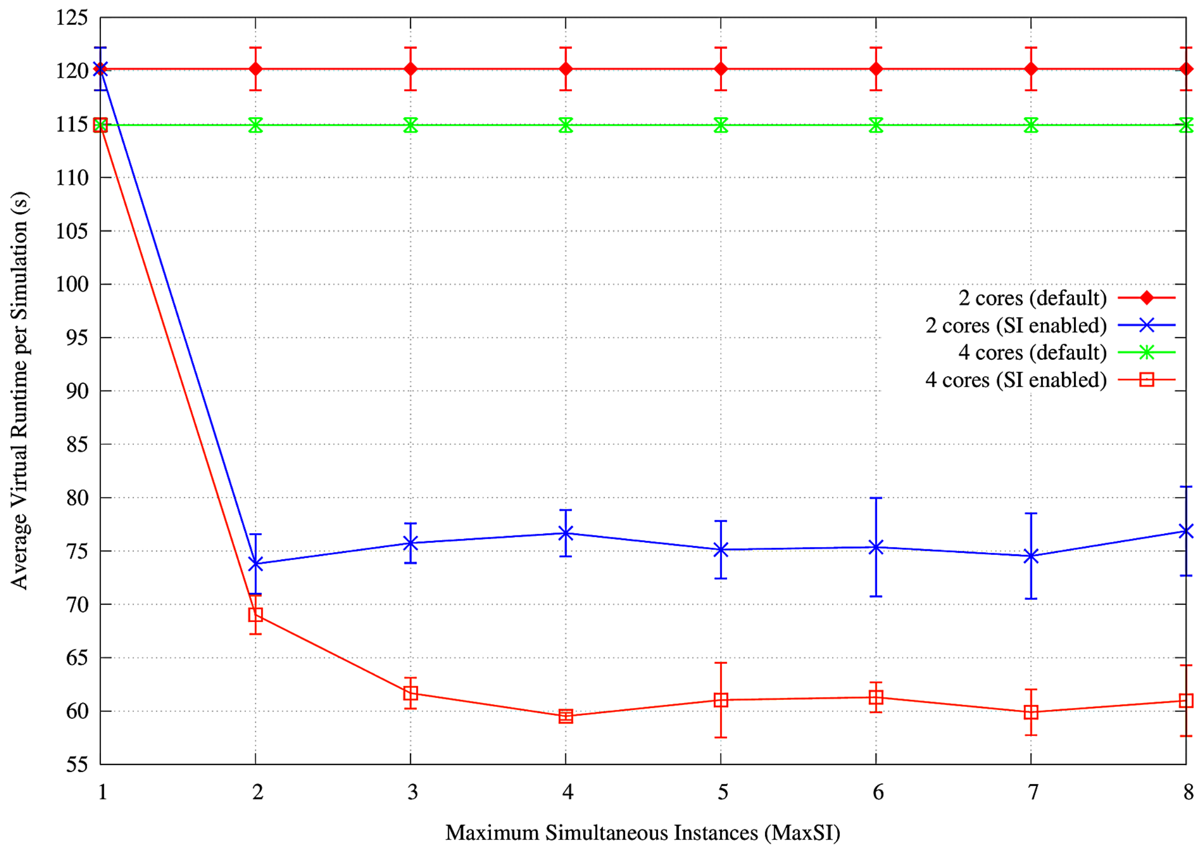


Figure 10 shows the results obtained for the simulation time ratio. The results show that the use of multiple virtual cores increases the simulation time ratio from 52% up to 101%, when using 4 virtual cores and a MaxSI of 4. In this case, the simulation runtime is slower than the simulated time in default mode, but is faster than the simulated time when SI is enabled. The results also reinforce that the highest reduction in simulation runtime is obtained when the maxSI is equal to the number of the used virtual cores.

The use of SI mode impacts on CPU load and RAM usage. Table 4 presents the values collected for CPU load and RAM memory usage. CPU load is expressed as a percentage of total processor resources, ranging from 0% to 100% when all cores of the CPU are in complete use. For instance, if one core is using 100% of its capabilities and the remaining three are using 0%, the represented CPU load will be of 25%.

Figure 11 shows the average load of the CPU for the tested MaxSI values. When using 2 virtual cores, the CPU load reaches 100% (approx.) for a MaxSI value of 2 or higher. When using 4 virtual cores, the CPU load reaches 100% (approx) for a MaxSI value of 4 or higher. As a side effect, an increase of the temperature of the real CPU was detected, reaching 100ºC during simulation runtime.

The average RAM usage for each tested scenario is shown in Figure 12. The obtained results show that, if SI is enabled, the memory usage increases until SI equals 5. For a MaxSI value greater than 5, the memory usage is constant and around 1900MB which is close to the 2000MB memory limit of the system.

Figure 10. Simulation time ratio

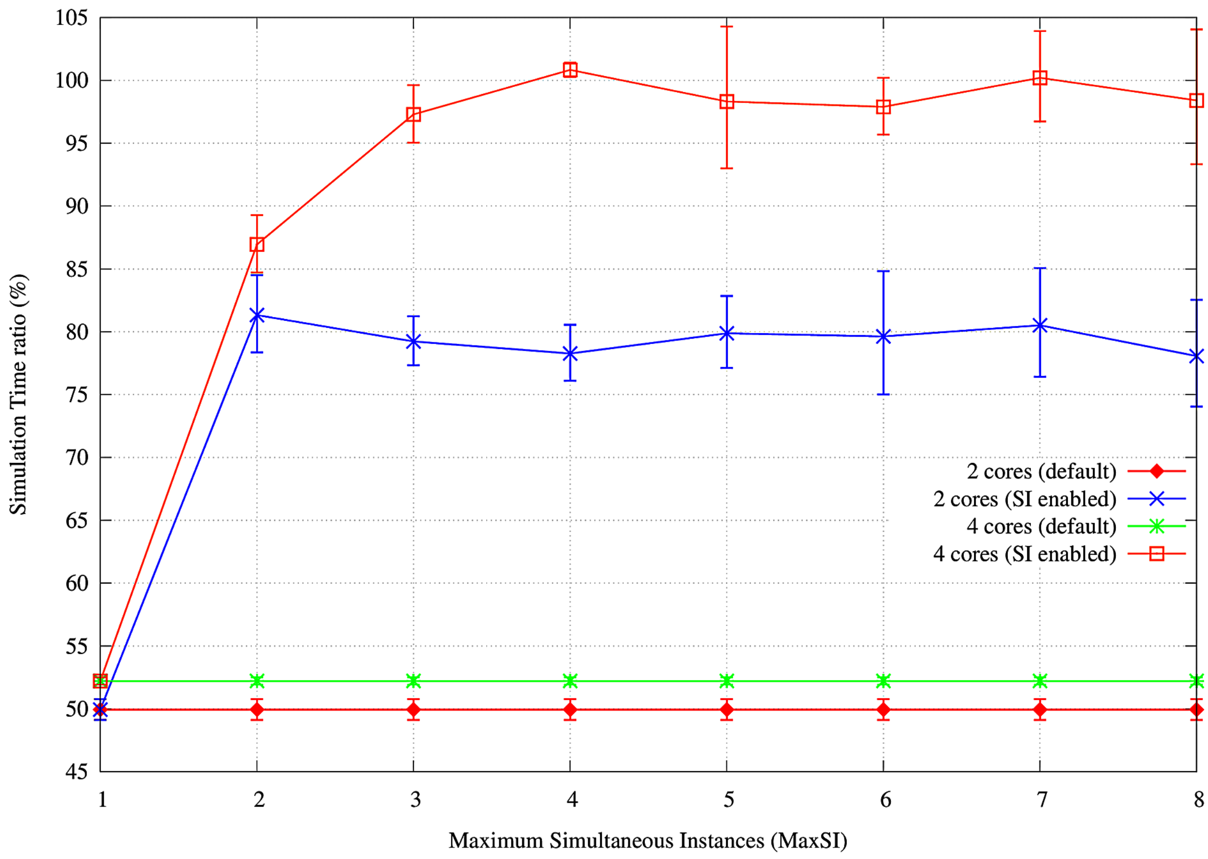
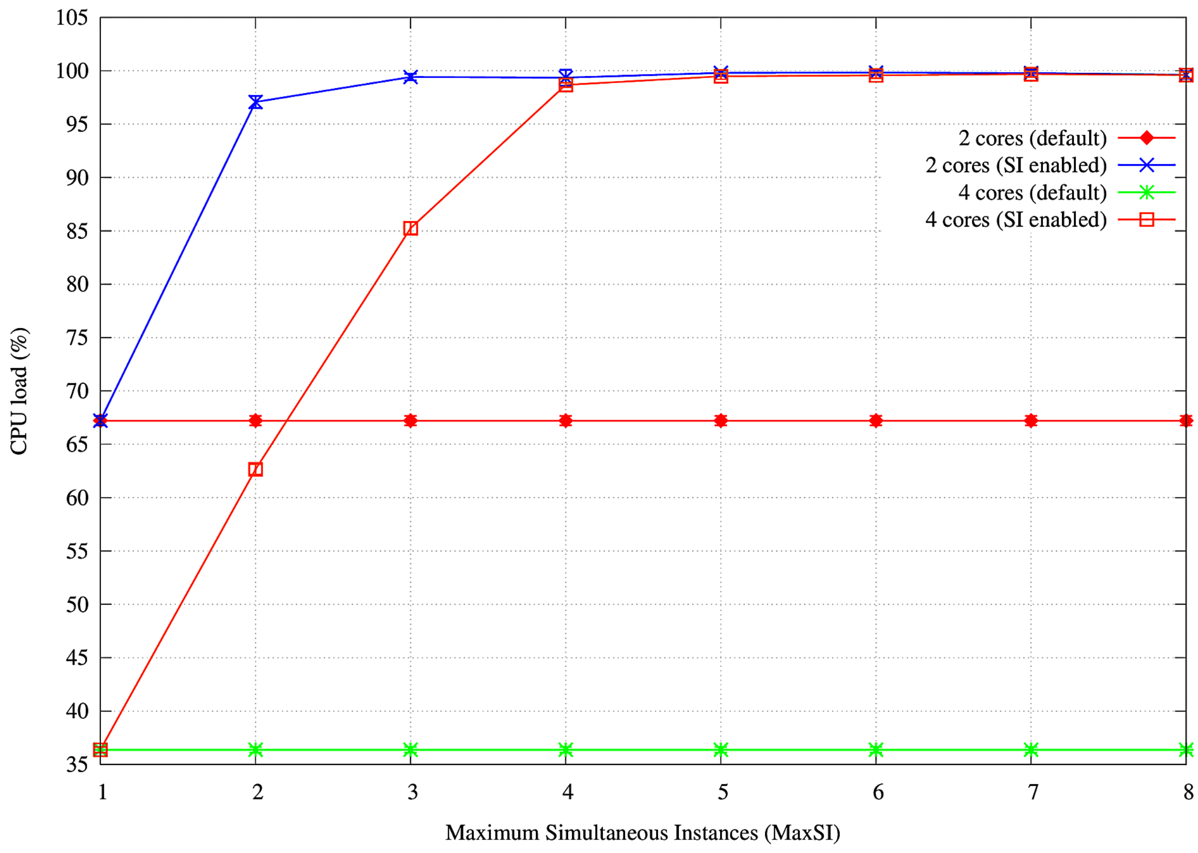


Table 4. Average CPU load and Memory usage

	CPU Load (%)		RAM Usage (MB)	
MaxSI	**2 Cores**	**4 Cores**	**2 Cores**	**4 Cores**
1	67.2	36.3	1453.540	1339.591
2	97.0	62.6	1526.626	1395.175
3	99.4	85.2	1741.389	1676.623
4	99.3	98.6	1897.306	1863.829
5	99.7	99.4	1915.097	1908.103
6	99.8	99.5	1920.312	1913.760
7	99.7	99.6	1916.498	1915.830
8	99.6	99.5	1915.626	1921.616

Figure 11. Average CPU load



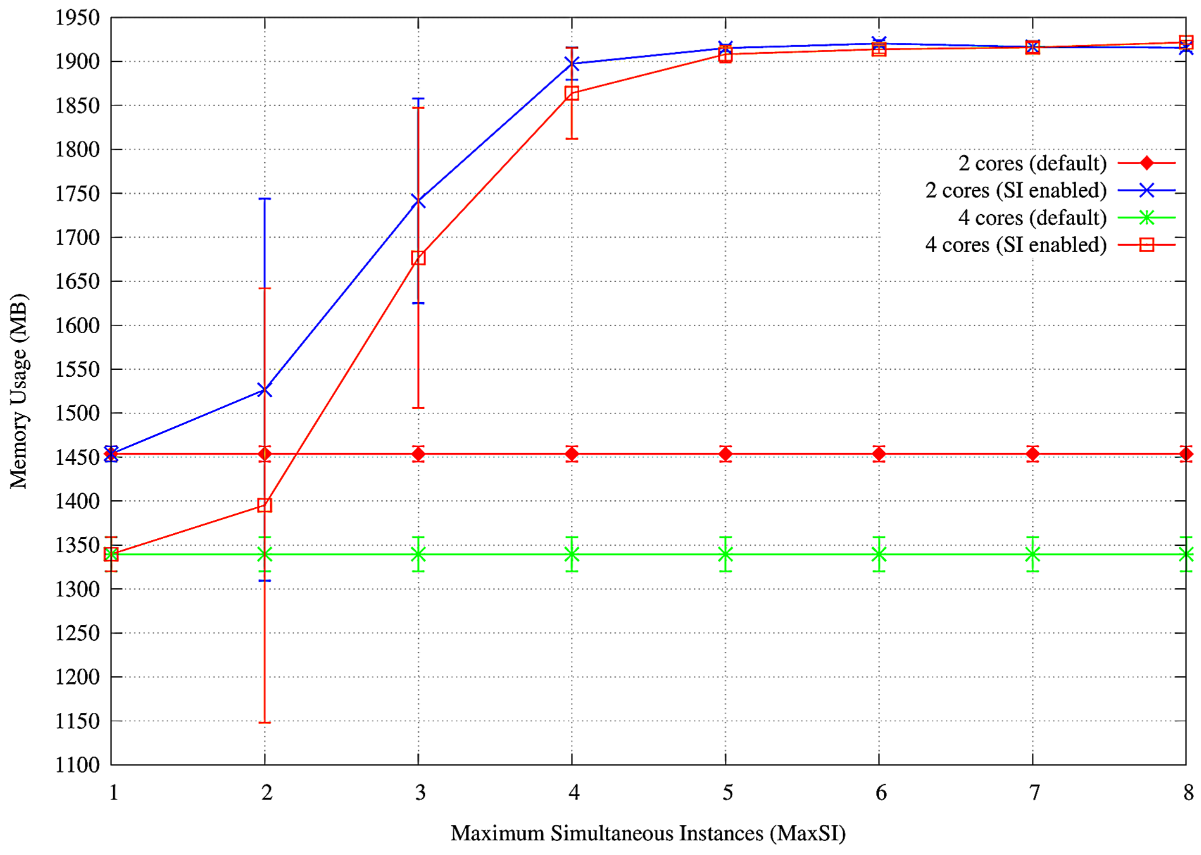
FUTURE RESEARCH DIRECTIONS

Contiki OS includes the Cooja simulator which is a cross-layer simulation environment that allows WSN developers to emulate specific hardware motes, though providing close-to-real results. The Contiki developers' community has been releasing significant improvements, support for new hardware and new features in their releases. Since the WSNs are usually formed by large number of devices, more support for large scale simulations is expected in near future. Any reduction on simulation runtimes on simulation tools will be highly valuable to researchers and developers dealing with large scale scenarios.

At the same time, all the actual hardware capabilities must be used to reduce the simulation runtime in order to enable WSN developers to cross from network design to deployment. Up-to-date benchmarks regarding simulation runtimes in real hardware could be provided to developers and researchers as this is an important item to consider in this context.

The Cooja simulator allows for the simulation of a set of different hardware motes in the same scenario. Regarding this feature, further attention can be devoted to improve the heterogeneous capability of Cooja and other simulators and also to provide interoperability between multiple simulators to perform simulations of complex scenarios involving multiple hardware and software.

Figure 12. Average Memory usage



CONCLUSION

Contiki developers may use compatible hardware and a set of available developments tools that include the Cooja simulator to design and test WSNs scenarios. The Cooja simulator allows for close-to-real simulations by using the full Contiki's source code and by emulating the hardware of some motes. Has a downside, its simulations take longer. The current version of the Cooja simulator uses a single process and does not take full advantage of current multi-core CPU architectures.

In this chapter, a simulation framework that automates Cooja simulations and can run multiple simultaneous Cooja instances, each one using a different core of the CPU, if available, is proposed. It reduces the simulations runtimes while increasing the average CPU load and RAM usage. The proposed simulation framework achieves the best results when the maximum number of simultaneous instances is equal to the number of (virtual) cores.

REFERENCES

Z1 mote. (n. d.). Retrieved from http://zolertia.com/products/z1

Carlson, T. E., Heirman, W., & Eeckhout, L. (2011). *Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-core Simulation.* Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 52:1–52:12). New York, NY, USA: ACM. http://doi.org/ doi:10.1145/2063384.2063454

Carlson, T. E., Heirman, W., & Eeckhout, L. (2013). *Sampled simulation of multi-threaded applications.* Proceedings of 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (pp. 2–12). http://doi.org/ doi:10.1109/ISPASS.2013.6557141

Contiki O. S. (n.d.). Retrieved from http://www.contiki-os.org

Contiki Developers Mailing List. (2012, October 11). Retrieved from http://comments.gmane.org/gmane.os.contiki.devel/15410

Goldsby, M. E., & Pancerella, C. M. (2013). *Multithreaded agent-based simulation.* Proceedings of Simulation Conference (WSC) (pp. 1581–1591). doi.org/ doi:<ALIGNMENT.qj></ALIGNMENT>10.1109/WSC.2013.6721541

Intel® CoreTM i7-2620M Processor. (n. d.). Retrieved from http://ark.intel.com/products/52231/Intel-Core-i7-2620M-Processor-4M-Cache-up-to-3_40-GHz

J-Sim Web site. (2015). Retrieved from https://sites.google.com/site/jsimofficial/

Kihm, J. L., & Connors, D. A. (2005). *Statistical simulation of multithreaded architectures.* Proceedings of 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication *Systems (pp. 67–74).* http://doi.org/ doi:10.1109/MASCOTS.2005.68

MICAz. (n. d.). Retrieved from http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

NS-2. (n. d.). Retrieved from http://www.isi.edu/nsnam/ns/

NS-3. (n. d.). Retrieved from http://www.nsnam.org/

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., & Voigt, T. (2006). *Cross-Level Sensor Network Simulation with COOJA*. Proceedings 2006 31st IEEE Conference on Local Computer Networks (pp. 641–648). http://doi.org/ doi:10.1109/LCN.2006.322172

RIOT Operative System. (n. d.). Retrieved from http://www.riot-os.org/

SeedEye. (n. d.). Retrieved from http://www.evidence.eu.com/products/seed-eye.html

Sundresh, S., Kim, W., & Agha, G. (2004). *SENS: a sensor, environment and network simulator.* Proceedings of Simulation Symposium (pp. 221–228). http://doi.org/ doi:10.1109/SIMSYM.2004.1299486

Tiny, O. S. (n. d.). Retrieved from http://www.tinyos.net/

Tmote Sky Project. (n. d.). Retrieved from http://www.snm.ethz.ch/Projects/TmoteSky

VMware. (n. d.). Retrieved from https://www.vmware.com/

KEY TERMS AND DEFINITIONS

Cooja Simulator: A cross-layer java-based wireless sensor network simulator distributed with Contiki. It allows the simulation of different levels from physical to application layer, and also allows the emulation of the hardware of a set of sensor nodes.

Core: Processing unit that is able to process programmed instructions.

Monte Carlo Experiment: Experiment method that follows an algorithm that rely on a repeated number of the same experiment each using different conditions (e.g. random seeds, random sampling, etc).

MSPSim: A java-based simulator of msp430 sensor network platforms.

Simulation Instance: A simulation executed in an instance or thread that is independent of all other instances or threads in execution.

Thread: Portion of programmed instructions that can be managed (executed) independently.

Wireless Sensor Network: Network of multiple sensor nodes used for monitoring the environment where they are deployed. Usually these sensor nodes are very limited in processing and energy resources.