Reducing WSN Simulation Runtime by using Multiple Simultaneous Instances

Pedro Pinto[a,c], António Pinto[b] and Manuel Ricardo[c]

[a]*ESTG, Instituto Politécnico de Viana do Castelo, Viana do Castelo, Portugal*
[b]*CIICESI, ESTGF, Politécnico do Porto and INESC TEC, Porto, Portugal*
[c]*INESC TEC, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal*

Abstract. WSN can be deployed using widely available hardware and software solutions. The Contiki is an open source operative system compatible with a wide range of WSN hardware. A Contiki development environment named InstantContiki is also available and includes the Cooja simulation tool, useful for the simulation of WSN scenarios, prior to their deployment. This simulation tool can provide realistic results since it uses the full Contiki's source code and some motes can be emulated at the hardware level. However, the Cooja simulator uses one process per simulation, not taking advantage of multiple core processors. In this paper we propose a framework to automate the execution of simulations of multiple scenarios and configurations in Cooja. When a multiple cores processor is available, this framework can run multiple simultaneous Cooja instances, taking advantage of processing resources and contributing to reduce the total simulation runtime.

Keywords: WSN, Contiki, Cooja, Simulation.
PACS: 06, 89

WSN DEVELOPMENT CHALLENGES

WSN can be deployed in real scenarios using hardware products such as the Z1[1], the SeedEye[2], the MICAz[3], or the Tmote Sky[4]. For these hardware products, multiple operative systems are available, such as the TinyOS[5], the RIOT-OS[6], and the Contiki [7].

The InstantContiki is a Contiki development environment available as a ready-to-use Ubuntu Linux VMware virtual machine which includes the Contiki and the Cooja simulator [8], among other tools. Version 2.7 is the current version of InstantContiki and its Cooja simulator uses a single process, i.e., it runs in a single core. If Cooja simulator runs within a virtual machine with multiple virtual cores available, they are underused when simulations are performed. Also, while there may not be a direct correspondence between the number of virtual cores assigned to the 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.

Simulators such as NS-3, assume that motes run simplified versions of the real software and hardware. The Cooja simulator differs because it allows the simulation of full Contiki's source code in a set of emulated hardware nodes. The simulation of full Contiki's source code running in a specific emulated hardware platform has the advantages of obtain close-to-real results and enable the fast deployment of the simulated experiments directly onto the real motes; however, it increases simulation complexity and simulation runtime.

In order to obtain statistically sound simulation results, multiple simulations are usually run, either to test different random topologies or test the same scenario with random seeds, for instance. Such rounds of repeated simulations can sum up to several hours or even days of simulation runtime. The proposed simulation framework automates Cooja simulations and also takes advantage of multiple virtual cores by running multiple simultaneous Cooja instances.

SIMULATION FRAMEWORK

The proposed simulation framework is presented in Fig. 1. It comprises the main functional block, the *Main Launcher*. Here, the user provides 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 the execution of all simulations, launching parallel instances for each simulation, each one with its own set of parameters. Each instance is composed of a *Simulation Setup*, a *Simulation Execution* and a *Simulation Analysis*. According to the *Simulation Scheduler* information, the *Simulation Setup* generates all simulation parameters in one main configuration file and then starts the *Simulation Execution*. The *Simulation Analysis* starts after the end of the simulation and analyses the simulation log to generate graphics of the obtained results.

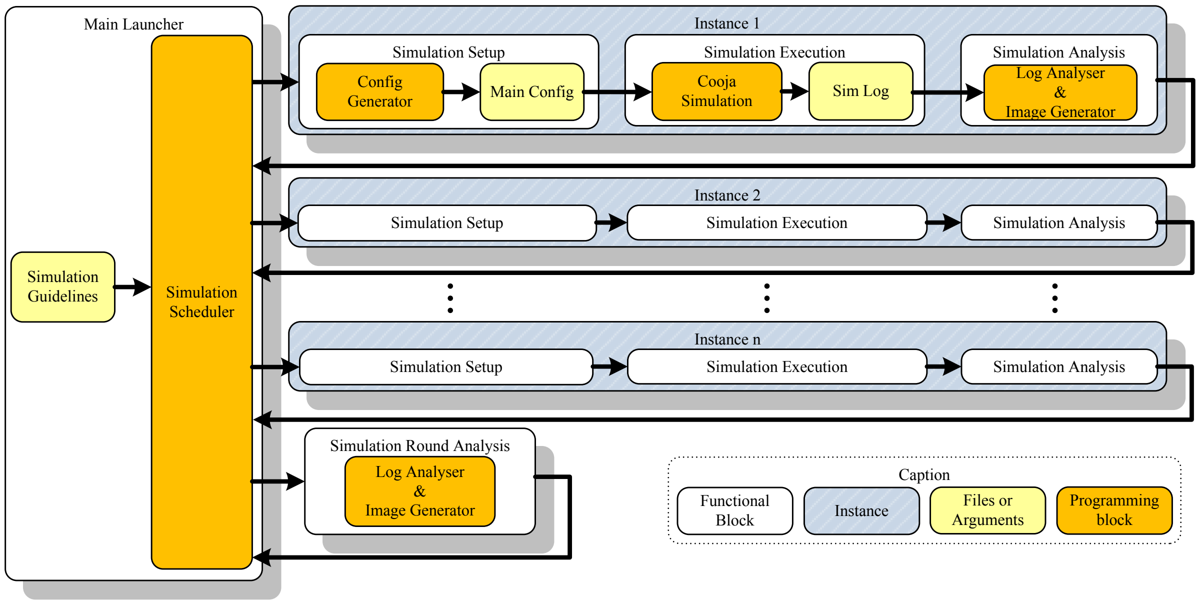

FIGURE 1. Simulation Framework

The simulation framework has a Simultaneous Instances (SI) mode that, when disabled, will result in a sequential execution of all simulations, where each instance will start after the termination of the previous one. 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 number of SI. 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. When the SI mode is enabled, the configuration of a maximum SI number is mandatory and it can be adjusted in order to obtain the maximum reduction of the simulation runtime. Different values for the maximum SI number were tested, and since the expected key impact of enabling SI mode is both the CPU load and RAM usage, the impact on these items was also evaluated.

RESULTS

In order to test the simulation framework presented, a WSN simulation was defined using Contiki 2.5 with 10 generator/forwarder nodes and a sink node, with a simulated time of 60s. The simulation framework was executed in an Ubuntu Linux operative system (12.04 LTS) 32 bits with 2GB of RAM in a VMware Virtual Machine. The virtual machine processor emulated a real Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz processor, with 2 cores and 4 threads.

Two scenarios were created. In the first scenario the *Simulation Scheduler* was configured with the default mode, i.e., with SI disabled. In the second scenario the *Simulation Scheduler* was configured with SI enabled and tested with maximum SI values ranging from 2 to 6 (value 1 is the same as one instance with SI disabled). Both scenarios were repeated 10 times and the values of the average CPU usage, the RAM usage, and the simulation runtime per instance were collected. The scenarios were tested using different number of virtual cores configured in the 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/threads presented in the real CPU; in this case, this limit was 4). The values collected in the default mode assume that there is only one simulation instance and thus, the global runtime was calculated by multiplying the runtime of one instance by the maximum SI number. When SI is enabled, the global simulation runtime presented is the total time for a round of simulations equal to maximum SI number.

The results of global simulation runtime obtained for the maximum SI values tested for each scenario are shown in Fig. 2a). These results show that the global simulation runtime is lower when the SI mode is enabled and the best absolute results are obtained when it is used 4 virtual cores. When using a maximum SI value of 6 and 4 virtual cores, the total runtime reduced from 575s, in default mode, to 325s obtained with SI enabled. Fig. 2b) shows the virtual simulation runtime per instance, obtained by:

$$Virtual\ Runtime\ per\ instance = \frac{Global\ Runtime}{Max.\,Simultaneous\ Instances} \tag{1}$$

The virtual runtime per instance variable was used to evaluate if one should use a value for maximum SI that was lesser, greater or equal to the number of the used virtual cores. The results show that a higher reduction is achieved when using a value for the maximum SI that equals the number of virtual cores. Moreover, the lowest value, 45 seconds approximately, is obtained when the maximum SI is set to 4, which is the number of the virtual cores. This was the expected result since each instance will run in its virtual core and thus, increasing the maximum SI number to a value higher than the number of virtual cores, will result in loss of efficiency due to concurrency within each virtual core.

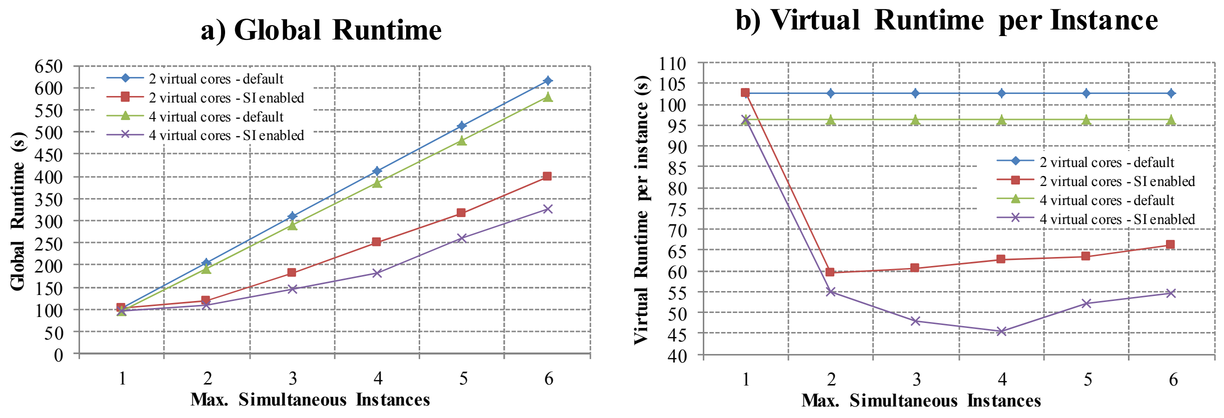

FIGURE 2. a) Global runtime and b) Virtual Runtime per instance

Fig. 3 presents the results for the simulation time ratio obtained using:

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

where the simulated time is constant and equal to 60s in all scenarios. The presented results show that the use of multiple virtual cores increases the simulation time ratio up to around 60%, when using 4 virtual cores and 4 maximum SIs. 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 maximum SI number is equal to the number of used virtual cores.

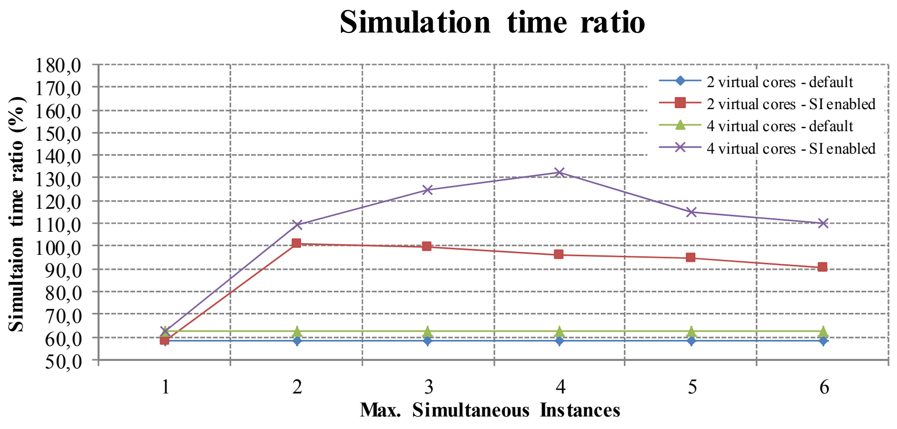

FIGURE 3. Simulation time ratio

The use of SI mode impacts on CPU load and RAM usage. Fig. 4a) shows the average load of the CPU for the tested maximum SI values. The indicated values are expressed as a percentage of total processor resources, assuming values from 0% to 100%, where 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 load will be 25%. The results show that, for scenarios with 2 virtual cores, the CPU load reaches 100% for a maximum SI value of 2 or higher. For scenarios using 4 virtual cores, CPU will reach 100% for a maximum SI value of 4 or higher. As a side effect, these results implied an impact on the temperature of real CPU which increased up to values around 100ºC, during simulation runtime.

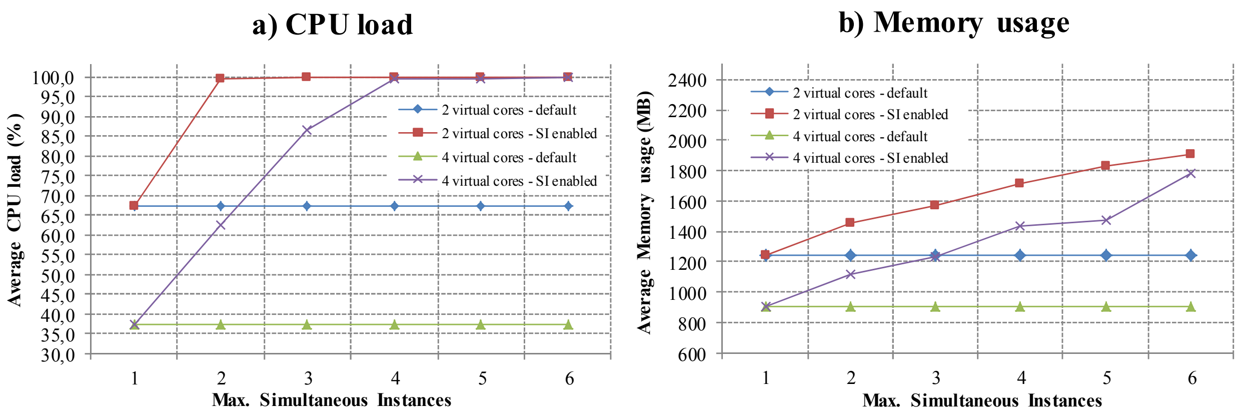

FIGURE 4. a) Average CPU load and b) Average Memory usage

The average RAM usage for each tested scenario is shown in Fig. 4b). The obtained results show the expected behavior: in default mode the simulation instances will have a sequential execution and thus, the memory usage is somewhat stable. On the other hand, with SI enabled the memory usage is expected to increase with the number of SI. The results show that the memory usage increases approximately 200MB per instance.

CONCLUSION

In order to design and test WSNs scenarios Contiki developers may use compatible hardware and a set of available developments tools that include the Cooja simulator. Although Cooja simulator allows close-to-real simulations since it uses the full Contiki's source code and emulates the hardware of some motes, it has a downside: simulations runtimes take longer. Also, the current version of the Cooja simulator uses a single process and does not take full advantage of current multiple core processor architectures.

The proposed simulation framework automates Cooja simulations and can run multiple simultaneous Cooja instances, each one using a different core of the CPU, if available. The proposed framework enables the reduction of simulations runtime by increasing the average CPU load and RAM usage. Moreover, the proposed framework achieves the best results when the maximum SI number is equal to the number of processor virtual cores.

REFERENCES

1. "Z1 mote." [Online]. Available: http://zolertia.com/products/z1.
2. "SeedEye." [Online]. Available: http://www.evidence.eu.com/products/seed-eye.html.
3. "MICAz." [Online]. Available: http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf.
4. "Tmote Sky Project." [Online]. Available: http://www.snm.ethz.ch/Projects/TmoteSky.
5. "TinyOS." [Online]. Available: http://www.tinyos.net/.
6. "RIOT Operative System." [Online]. Available: http://www.riot-os.org/.
7. "Contiki OS." [Online]. Available: http://www.contiki-os.org.
8. F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, 2006, pp. 641–648.