Demo Abstract: MakeSense—Managing Reproducible WSNs Experiments

Rémy Léone, Jérémie Leguay, Paolo Medagliani and Claude Chaudet

Abstract Wireless Sensor Networks (WSN) users often use simulation campaigns before real deployment to evaluate performance and to fine-tune application and network parameters. This process requires repeating the same experiments under similar conditions and to collect, parse and present data efficiently. This chapter introduces MakeSense: a tool that automates this workflow and that allows reproducing simulations easily by defining the whole experiment and post-processing steps in a single JSON configuration file, easy to share and to modify. MakeSense also provides interfaces to interact with a running simulation, allowing to send external stimuli and to collect data in real time. MakeSense currently runs over the COOJA simulator, but has been built to be easily adapted to other architectures, including real testbeds.

1 Introduction

Evaluating applications and protocols through simulation always follow the same workflow from the simulation parameters definition to the creation of graphs that represent performance under different conditions. Yet, no generic tool really provides a way to automate the whole process and people often rely on ad-hoc scripts. Some

R. Léone (✉) · J. Leguay · P. Medagliani
Thales Communications & Security, Gennevilliers, France
e-mail: remy.leone@telecom-paristech.fr; remy.leone@thalesgroup.com

J. Leguay
e-mail: Jeremie.Leguay@thalesgroup.com

P. Medagliani
e-mail: Paolo.Medagliani@thalesgroup.com

R. Léone · C. Chaudet
Institut Mines-Télécom, Télécom ParisTech, CNRS LTCI UMR 5141, Paris, France
e-mail: Claude.Chaudet@telecom-paristech.fr

tools such as NEPI [3] focus on the interaction between testbeds, but do not address wireless sensor network platforms and hence, their architecture may be too complex for constrained devices.

In this chapter, we introduce MakeSense, a tool that automates large-scale WSN experiments and facilitates adaptation and reproducibility. MakeSense relies on a single JSON configuration file described in Sect. 2 from scenario definition to graphs generation. Sharing this lightweight file with the specific source code is sufficient to let others reproduce an experiment, improving the results trustworthiness [1]. MakeSense runs today over the COOJA simulator, but is designed to be generic and to be adapted to real testbeds too. We then describe its workflow and functionalities, including online interaction with simulation and describe a demonstration in Sect. 3.

2 MakeSense Core Functionalities

2.1 JSON Configuration File

All simulation parameters such as random seeds, transmission range are defined in a single JavaScript Object Notation configuration file. An excerpt of such a configuration file is presented on Listing 1.1. The file starts by the definition of some general parameters, such as the wireless interference range and specifies a template for the Makefile that will serve during the building process. It then defines one or several mote types (e.g. routers, end nodes, ...), the firmware they will use and gives them an RPL instance ID. In the `motes` section, it instantiates the different nodes with respect to the templates and specifies individual parameters such as their coordinates, their ID (address) and the UDP port on which they will be reachable during the simulation.

As this configuration file specifies all relevant parameters for a simulation, it is easy to use templates and scripting to run large experimental campaigns. A script can generate a series of such files, referring to the same or to different source codes, run experiments and store the JSON files as scenario descriptors, as they are lightweight.
Listing 7.1 Configuration file excerpt

```
  [...]
"interference_range": 50.0,
"makefile_template":
      "simple_makefile",
"mote_types": {
  "hello": {
      "color": "blue",
      "description": "Router",
      "firmware_address": [
        "nodes",
        "hello",
        "hello-world.sky"
      ],
      "rpl_instance_id": 30
  }
},
"motes": [
  {
      "mote_id": 1,
      "mote_type": "hello",
      "settings": {
        "lambda": 1
      },
      "socket_port": 60001,
      "x": 10,
      "y": 0
},
  [...]
```

2.2 Workflow

MakeSense workflow, represented schematically in Fig. 1, is composed of 6 steps that can be called independently, thanks to their loose coupling, or run in sequence using the `run_all` shortcut.

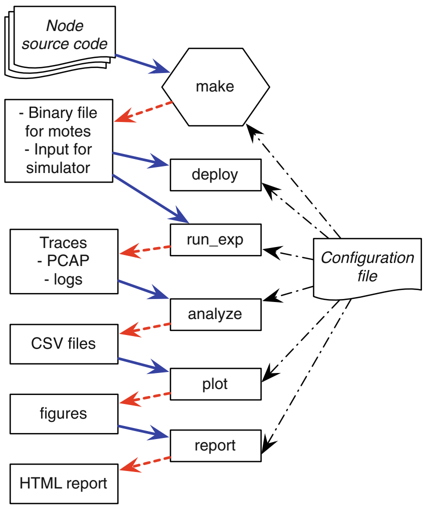


Fig. 1 MakeSense workflow

make	creates the whole environment necessary for the experiment execution. It compiles the source code and creates the configuration files required by the simulation tool from templates.
deploy	uploads the simulation files from a local repository to a remote location to run several simulations in parallel. This step can easily be adapted to upload firmwares to a real testbed.
run_exp	launches the simulation series. MakeSense can launch concurrently, several functions that will fetch the simulation output through the nodes' serial or network interfaces. Each node has its own independent log file.
analyze	parses trace files, e.g. PCAP files or text logs file, by applying a set of filters to produce CSV files that contain only the desired information. Filters are easy to specify and MakeSense includes a set of basic filters.
plot	produces graphs using the CSV files produced by the analyze step and the info in the settings files to select relevant data.
report	gathers all the results into a single HTML report file.

2.3 Multiple Control Channels

The simulation runs from its beginning to its end without user interaction, or can be run in real-time mode, allowing real-time interaction with traffic generators such as real applications, other simulators, or a user using an interactive command line and visualizing performance evolution through the COOJA scripting engine, as illustrated in Fig. 2. This feature allows to let the WSN run as if it were a real network and to measure its reaction to external, controlled, stimuli.

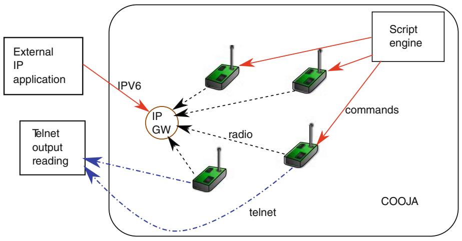


Fig. 2 Real-time interaction with COOJA

3 Demonstration

The COOJA implementation of MakeSense uses the Python programming language and other libraries such as fabric, jinja2 and matplotlib. As a demonstration, we use COOJA [2] in real time mode to simulate a network of 10 nodes connected using an RPL tree. The first node is a border router that connects all the remaining nodes that are CoAP servers, to the hosting operating system. We send traffic to the different sensor nodes from the shell and from the script engine contained inside the simulator. This traffic is composed of CoAP and PING requests.

As specified in the configuration file, MakeSense generates without user interaction various graphs. For example, Fig. 3 represents the theoretical connectivity graph, extracted from the sole configuration file using circular transmission range. Figure 4 is a representation of the real RPL tree, generated by querying nodes for RPL information on their serial interface, without perturbing the network traffic. Figure 5 is a classical per-protocol throughput graph, showing the use of per-protocols filters and Fig. 6 represents delay of Ping requests issued from the hosting operating system, that shows the interaction with the outside world.

Fig. 3 Connectivity graph

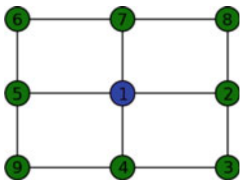
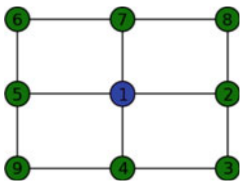


Fig. 4 RPL tree



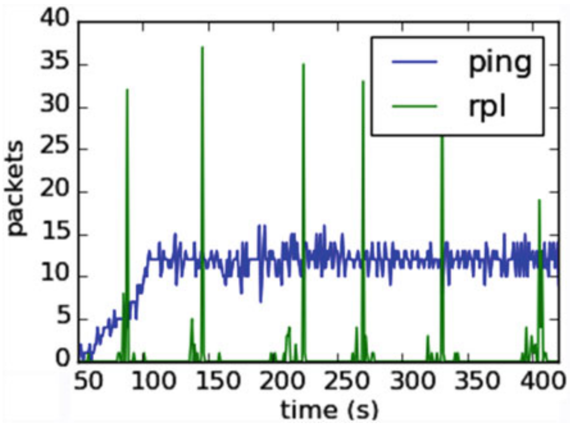


Fig. 5 RPL and ping traffic evolution

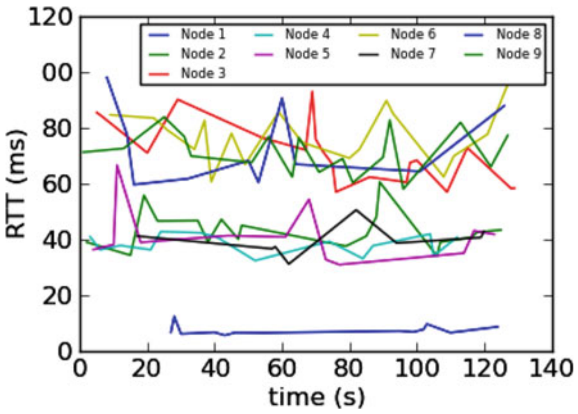


Fig. 6 Ping RTT between host machine and simulated node

4 Conclusion

This chapter describes and shows an example use of MakeSense, a framework for automating series of simulation in order to make them reproducible and reusable. The current implementation is tailored for the Cooja simualtor, and we are working on its adaptation to real testbeds. The MakeSense Python implementation for Cooja is available at: http://github.com/sieben/makesense.

This work takes place in the context of the ANR IRIS project and was partially carried out at the LINCS laboratory.

References

1. Aruliah, D.A., Brown, C.T., Hong, N.P.C., Davis, M., Guy, R.T., Haddock, S.H.D., Huff, K., Mitchell, I., Plumbley, M., Waugh, B., White, E.P., Wilson, G., Wilson, P.: Best practices for scientific computing. Comput. Res. Reposit. abs/1210.0530 (2012)
2. Eriksson, J., Österlind, F., Finne, N., Tsiftes, N., Dunkels, A., Voigt, T., Sauter, R., Marrón, P.J., COOJA/MSPSim: interoperability testing for wireless sensor networks. In: 2nd International Conference on Simulation Tools and Techniques (SIMUTools 2009), p. 27. Rome, Italy (2009)
3. Lacage, M., Ferrari, M., Hansen, M., Turletti, T., Dabbous, W.: NEPI: using independent simulators, emulators, and testbeds for easy experimentation. ACM SIGOPS Oper. Syst. Rev. **43**(4), 60–65 (2010)