A simulation execution manager for ns-3

Davide Magrin

1 Personal Information

1.1 Identity Information

- Name: Davide Magrin
- Email: magrinda@dei.unipd.it
- Country: Italy
- Institution: University of Padova
- Advisor: Prof. Michele Zorzi

1.2 Patch Requirement

`lorawan` module for ns-3: `https://github.com/DvdMgr/lorawan`

1.3 Background

1.3.1 Education

I received my BSc in Information Engineering in 2014 and my MSc in Telecommunications Engineering in 2016 from the University of Padova. After receiving a six month study grant to continue my master's thesis work, I entered the PhD program at the University of Padova under the supervision of Prof. Michele Zorzi, in order to pursue research on simulation of large scale IoT networks.

I currently employ both ns-3 and various python libraries to develop my research. Both my bachelor's and my master's theses were centered on ns-3 usage, and I have used both python and C/C++ during my studies to complete class projects, among which machine learning for gesture recognition (using the python `scikit` library), implementation of efficient channel coding schemes and simulation of digital communication architectures (via MATLAB and C++).

1.3.2 Work

No current work experience.

1.3.3 Experience with ns-3

For my bachelor's thesis I used ns-3's `energy` and `lr-wpan` modules to analyze the energy efficiency of prioritized data transmission schemes in a Wireless Sensor Network. My master's thesis consisted in the creation of an ns-3 module to simulate and analyze the performance of a LoRaWAN network. During my six month study grant I experimented with parallel simulation of the wireless channel in ns-3, and further developed the `lorawan` module. Recently, I've supervised the work of two students working with and adding features to the `lorawan` module.

1.3.4 Open-source experience

I currently use and follow the development of some open source projects (mainly Spacemacs and ns-3), but I haven't contributed to any yet.

1.3.5 Research

Publication describing the `lorawan` ns-3 module:

- D. Magrin, M. Centenaro and L. Vangelista, "Performance evaluation of LoRa networks in a smart city scenario," 2017 IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-7. DOI: 10.1109/ICC.2017.7996384

1.3.6 Why you?

Considering my previous experience both as a student learning the ns-3 framework and as a supervisor helping students understand and use the simulator, and considering that my whole PhD will be centered on ns-3 usage, I think I am a good candidate to identify and efficiently implement improvements to the platform. The implementation of the work detailed in this proposal will help both me and my research group get research done more easily, and will be the basis upon which I will build the rest of my large scale IoT simulation research.

2 Project description

2.1 A simulation execution manager for ns-3

ns-3 already provides built-in structures to extract, transform and aggregate data from simulations via the Data Collection Framework (DCF). The software also features robust randomization of simulations by design, allowing users to perform simulator runs of the same scenario using independent random variable streams, in order to increase confidence in the statistical soundness of the obtained results. Despite the premises described above, ns-3 currently lacks a way to automatically perform multiple runs of a simulation script, systematically collect the results of the executions and export them to dedicated frameworks for analysis.

The only available solution to this 'multiple execution management' problem is contained in the statistical framework module, in the form of a bash script. This script, however, seems to be meant more as an example of how to perform multiple runs of a simulation for a certain range of parameters than as a general purpose solution. By contrast, a complete and easily configurable framework that allows users to perform multiple runs automatically could help enforce good practices in multiple simulation executions (e.g., correct usage of the `RngRun` parameter, orderly management of results and simulation reproducibility), help students easily obtain results without having to deal with bash scripts and custom solutions that distract them from the goals of their project, and ultimately avoid the duplication of code that is inevitable when every user needs to re-implement support for such a common and necessary use case.

2.2 Approach

The envisioned solution consists in the creation of a python library that can start, manage and collect results from multiple simulation runs. In essence, the aim of this library is to make obtaining and organizing results from multiple ns-3 script executions straightforward: users will only need to worry about writing the desired simulation script and specifying the range of parameters they desire to run the script with. The framework will then optimize the execution of the needed simulation runs, store the results in a human-readable way and wrap them up in a data structure that can then be analyzed using `numpy`, `MATLAB` or similar tooling.

2.2.1 Other assumptions and integration notes

To remain as general as possible, this framework will try to make as few assumptions as possible about the workflow employed by the user. One possible approach is to perform data collection and reduction directly in-memory inside the simulation, which then only outputs a series of results from the 'single-shot' simulation, for example via the standard output, for the framework to collect and store. Additionally, an output consisting in multiple files that are then processed via user-defined scripts can also be supported via execution of user-specified post-processing scripts.

In any case, this library is meant only as a support to the collection of results of multiple simulations, and it will not provide any experiment control facilities for stopping experiments early or managing result parsing and storage directly from ns-3. This choice is motivated primarily by the current incompleteness of the statistical framework: a full integration would require a non-negligible amount of work on the currently available code, which would subtract time from the creation of a robust multiple execution framework, which is the priority of this proposal. Additionally, the availability of user friendly python libraries for thread pool management and database communications further push the choice towards a python framework.

2.2.2 Testing and documentation

Code will be tested throughout development using the facilities provided by the `pytest` library according to the test driven development paradigm, giving a way to perform regression testing upon implementation of new features and new releases of ns-3. The three test suites that will be provided, each focusing on one of the three main components of the projects, will be discussed in the next section.

The `sphinx` library will be used to generate API documentation directly from the code. A detailed ns-3 manual entry will also be provided, with the aim of guiding a user with a beginner python experience through the configuration and usage of the library.

Examples on how to get from simulation script to plots through the execution framework will be provided for commonly obtained result types, like the time series evolution of a parameter, the statistical distribution of a random variable of interest and the study of a metric across multiple runs using different parameter choices. This, however, is a feature that will be given lower priority, since the main focus of this proposal remains more on the creation of a multiple execution framework, and less on providing users

guidance in result analysis and plotting.

2.2.3 Example workflow

This section describes the three steps of the envisioned workflow: simulation campaign creation/loading, script execution and result management. Usage examples are given along the way to convey an idea of how the framework APIs will be used.

1. **Configuration**. Initially, the user needs to specify some information for the framework to correctly execute and handle the simulations. This information, consisting mainly in the script name and in the accepted parameter list, can be provided both by creating a dictionary structure or by specifying a file to load. After creation, the campaign object's configuration can be saved to a file.

   The following API calls illustrate the process described above:

   ```
   # Create a dictionary structure to specify configuration
   config = {
       'campaign-name': 'example-campaign',
       'script-name': 'example-script',
       'parameters': ['NumPackets', 'Distance']
   }

   # Initialize a new campaign starting from the config dictionary
   campaign = SimulationCampaign(config)

   # Load a previous campaign, whose configuration was saved
   # in a file
   # campaign = SimulationCampaign('config-name.json')

   # Current campaign info can be inspected at any time
   print (campaign)

   # Save current campaign configuration to file
   campaign.save ('ExampleCampaign.json')
   ```

   For each campaign, a corresponding document-oriented database is created by the framework. This database contains an entry for each simulation run, and each entry is tagged with metadata about the parameters that were used to perform the specified run. The technology that will be used to create this database can be either MongoDB[1] or

[1] https://www.mongodb.com/

TinyDB[2], depending on the project's preference for a more complete and powerful or a more portable solution. In order to ensure repeatability of the simulation, the simulation campaign manager also performs a diff of the code against a stable release of the ns-3 repository and saves it as part of the configuration, to make sure that no two simulations in the same campaign are performed starting using different source code. This component will also check the existence of the specified script in the ns-3 script database, and abort creation of the campaign if either the script does not exist or if the specified parameter list is not a subset of the one that is available from the program's command line. Tests checking for correct rejection of incorrect input will be incorporated in this component's test suite, additionally to tests making sure that the saving/loading cycle yields valid simulation campaign objects.

2. **Running simulations**. In order to run a script in a campaign, the framework needs to know the desired range of parameters to be simulated. After this information is set, the runner component of the `SimulationCampaign` object checks which parameter combinations are already available in the database, and performs the missing runs adding them to the database.

```
# Specify the desired parameter range and number of runs
# for each parameter combination
params = {
    'NumPackets': np.arange(1, 20), # Ranges as numpy arrays
    'Distance': 100,                # Constant parameters
    'runs': 20                      # Runs per combination
}

# Return a representation of the runs that are required
# by params and that aren't already available in the database
print("Missing runs: %s" % campaign.get_missing_runs (params))

# Run missing simulations
campaign.perform_missing_runs (params)
```

   Needed simulations can be run either sequentially or using a thread pool[3], which spawns python subprocesses[4] directly running the C++ executable. Calls to `perform_missing_runs` print information about

[2] `http://tinydb.readthedocs.io`
[3] `https://docs.python.org/dev/library/concurrent.futures.html`
[4] `https://docs.python.org/3/library/subprocess.html`

the current status of the simulation campaign, so the user can choose to interrupt the campaign and change the parameter choice.

In case the simulation process ends with an ns-3 error or if the program outputs warnings (e.g., in the form of a $\text{NS}_{\text{LOGWARN}}$ logging call), no entry is added to the database and the user is notified of the event. Similarly, if the simulation output contains Nan or Inf values the user will be warned.

Each run is performed in a dedicated temporary directory, and the results are copied over to the database once the run finishes. The choice to keep all results in a database instead of a directory tree is mainly driven by the easier access that the python framework can have to an object oriented database, in which e.g. all runs featuring a certain parameter choice can be easily queried, opposed to the more cumbersome directory structure navigation that would be necessary in case a directory tree was chosen as storage method.

This component will be tested by running simulation campaigns on ns-3 scripts which yield known results, with a particular focus on ensuring that sequential simulations give the same results as parallel simulations.

3. **Result management**. The SimulationCampaign object can export simulation results in three main ways:

   - *Raw export*: results are placed in a directory tree, corresponding to the available parameter combinations;
   - *numpy export*: a numpy multidimensional array is created, containing a results vector for each combination of parameters and run index, and optionally saved to a .npy file;
   - *matlab export*: analogous to the numpy export, except that the export format is a `.mat` file.

   This component will be tested by comparing data structures exported from simulation campaigns which yield known results against manually created (and thus surely correct) versions of the same data structures.

2.3 Deliverables

This section first provides a feature list for each component, and then describes how features can be grouped together in releases to create a progressively more complex and sophisticated framework.

2.3.1 Feature summary

1. **Configuration**. The configuration and database management subsystem is expected to reach the following main features list:

   - CONF1: Create, save and load campaign configurations;
   - CONF2: Create a campaign's database, add items to it;
   - CONF3: Perform complex queries on a campaign's database.

   The following features will also be pursued, but are to be given a lower priority in case unexpected issues with the main features arise:

   - CONF4: Save current state of the code to ensure future reproducibility.

2. **Running simulations**. The simulation running subsystem is expected to reach the following main features list:

   - RUN1: Handle ns-3 compilation;
   - RUN2: Perform a single-shot simulation;
   - RUN3: ns-3 exception handling;
   - RUN4: Perform a sequence of single-shot simulations;
   - RUN5: Perform multiple simulations in parallel leveraging multicore systems.

   The following features will also be pursued, but are to be given a lower priority in case unexpected issues with the main features arise:

   - RUN6: Visualize current state of simulations;
   - RUN7: Allow users to remove simulations from execution schedule.

3. **Result management**. The result management subsystem is expected to reach the following main features list:

   - RES1: Export to numpy format;
   - RES2: Export to directory tree;
   - RES3: Export to MATLAB `.mat` file.

4. **General**. The tasks contained in this section do not refer to a particular component of the framework:

- GEN1: Repository and project setup;
- GEN2: System-wide API definition;
- GEN3: Plotting examples;
- GEN4: Manual entry;
- GEN5: API documentation publication.

2.3.2 Milestones

Three incremental versions of the framework will be released, each providing a precise set of functionalities that are built on top of the previous versions:

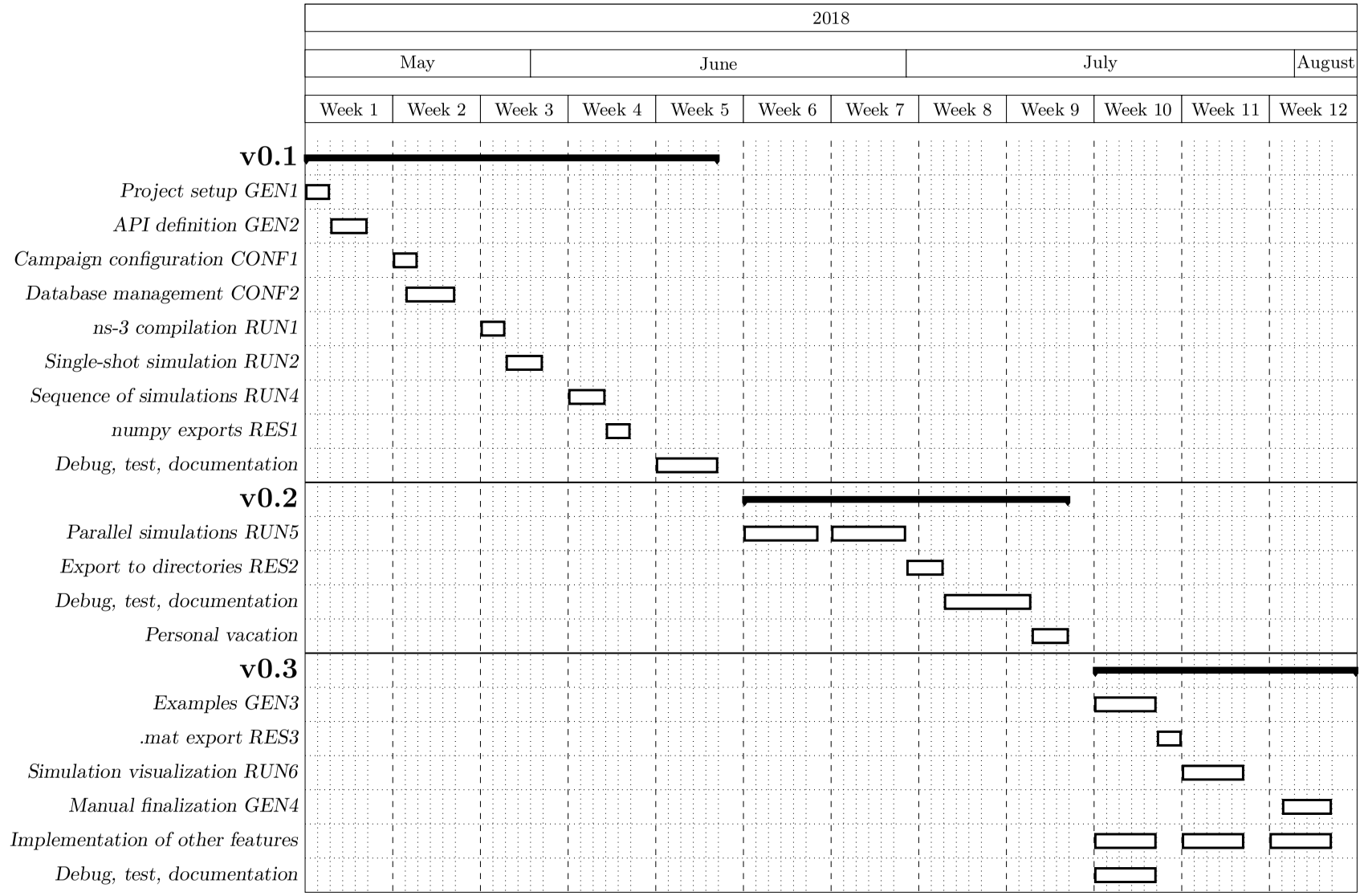
- Version 0.1 will provide a complete, albeit naive, implementation of the whole workflow. The user will be able to specify a campaign both from a dictionary and from a previously saved file, perform multiple simulations according to a certain parameter combination and have the framework add the results to the database. numpy array exporting will also be supported. Version 0.1 will be completed by June 15.

- Version 0.2 will allow users to export to directory tree structure, and add support for parallel execution of simulations. Version 0.2 will be completed by July 13.

- Version 0.3 will provide a more polished and thoroughly tested experience, and add `.mat` exports. The specification for this version is purposefully vague to leave room for backlogged tasks, bug catching, future feature exploration and implementation of secondary features. This version will be completed by August 5.

2.4 Plan

A complete work plan based on the features described in the previous section can be seen in the following page.


2018
May
June
July
August
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
v0.1
Project setup GEN1
API definition GEN2
Campaign configuration CONF1
Database management CONF2
ns-3 compilation RUN1
Single-shot simulation RUN2
Sequence of simulations RUN4
numpy exports RES1
Debug, test, documentation
v0.2
Parallel simulations RUN5
Export to directories RES2
Debug, test, documentation
Personal vacation
v0.3
Examples GEN3
.mat export RES3
Simulation visualization RUN6
Manual finalization GEN4
Implementation of other features
Debug, test, documentation

2.5 Timezone

During the GSoC coding period I will be located in Padova, Italy. I expect I will be working in the 7:00 - 11:00 and 13:00 - 17:00 UTC time slots.

2.6 Commitments

I will need to attend an afternoon class from June 24 to June 31. I also have a personal vacation planned from July 11 to July 13, however I plan to make up the lost time by working on the previous weekends.