Nancy-Playground: A Console Calculator for Deterministic Network Calculus (Artifact)

Raffaele Zippo
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy

Giovanni Stea
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy

Abstract

`nancy-playground` is an open-source, cross-platform command-line tool for Deterministic Network Calculus (DNC) computations. It implements the MPPG scripting syntax used by RTaW's min-plus playground [2], executing computations via the `Nancy` [4] and `Nancy.Expressions` [3] libraries and their state-of-the-art algorithmic optimizations. It also supports converting MPPG scripts to C# code and an interactive console mode. This artifact provides the source code of `nancy-playground` (version 1.0.8), example scripts corresponding to the listings of the companion paper, and scripts to reproduce the test and coverage results (Table 7) and the performance measurements (Table 6) of [5].

2012 ACM Subject Classification Computer systems organization → Real-time systems; Networks → Network performance analysis; Mathematics of computing → Mathematical software performance

Keywords and phrases Deterministic Network Calculus, min-plus algebra, tooling

Digital Object Identifier 10.4230/DARTS.12.2.6

Funding This work was supported in part by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence).

Related Article Raffaele Zippo and Giovanni Stea, "Nancy-Playground: A Console Calculator for Deterministic Network Calculus", in 38th European Conference on Real-Time Systems (ECRTS 2026), LIPIcs, Vol. 375, pp. 5:1–5:16, 2026. `https://doi.org/10.4230/LIPIcs.ECRTS.2026.5`

Related Conference 38th European Conference on Real-Time Systems (ECRTS 2026), July 7–10, 2026, Lund, Sweden

1 Artifact

This document introduces the artifact related to [5], i.e., the source code of `nancy-playground`, the scripts used to demonstrate its features, and the scripts used to collect the data presented in [5]. This artifact is provided, with an MIT license, as a GitHub repository at `https://github.com/rzippo/nancy-playground-artifact` [7], and is also archived on Software Heritage [6]. It is composed of:

- the source code of `nancy-playground` version 1.0.8, a cross-platform CLI tool written in C# using .NET 10, in the `Nancy-Playground/` directory;
- the `paper-examples/` directory, containing the MPPG scripts corresponding to the listings of [5], labelled by listing number;
- `run-tests-and-coverage-report.ps1`, a PowerShell script that runs the test suite and generates a coverage report, reproducing Table 7 of [5];
- `benchmark-np-run-vs-convert.ps1`, a PowerShell script that measures execution time and peak memory usage of `nancy-playground` and of an equivalent compiled C# program, reproducing Table 6 of [5];
- a `Dockerfile` providing a self-contained environment with all required tools pre-installed.

`nancy-playground` is also distributed as a binary package on NuGet [8], installable with a single command, which is the recommended approach for the qualitative tests.

2 Claims

The companion paper [5] presents `nancy-playground` as a tool with the following claims.

Qualitative claims

1. The `interactive` mode can be used to write MPPG scripts line by line, with immediate feedback and access to the integrated `!help` documentation.
2. The `run` command parses and executes entire MPPG scripts from file.
3. The `convert` command converts an MPPG script to an equivalent C# program using the `Nancy` library, for further development.

Quantitative claims

1. The MPPG parsing and computation is verified via a large test suite. In particular, `RunCommandGoldenTests` verify that `nancy-playground` produces the same results as RTaW's min-plus playground [2], and `ConvertCommand*Tests` verify that the converted C# program produces the same results as the `run` command on the same scripts. The current code passes 100% of these tests.
2. The test suite achieves 67% line coverage of the codebase (Table 7 of [5]).
3. Interpreting scripts via `run` incurs a measurable performance overhead compared to an equivalent compiled C# program (Table 6 of [5]). This overhead is small in the intended context of prototyping and experimentation.

3 To Reproduce the Results

3.1 Requirements

We provide two main ways of reproducing these results: using the published version of `nancy-playground`, directly on one's machine, or using the provided Dockerfile with all necessary tools.

Option A – Install from NuGet

`nancy-playground` is published on NuGet as a .NET Tool, which can be installed on any system with .NET 10 SDK `https://dotnet.microsoft.com/en-us/download/dotnet/10.0` installed. This method is recommended to verify the qualitative claims, as it makes easier to use `plot()`, which will open the generated plots using your default image viewer.

After installing the .NET 10 SDK, run the following command in your terminal:

```
dotnet tool install --global unipi.nancy.playground.cli
```

This installs `nancy-playground` as a global .NET tool, available as `nancy-playground` in the shell. Verify with:

```
nancy-playground --version
```

Table 1 MPPG scripts in `paper-examples/` and their corresponding listings in [5].

File	Listing in [5]
`listing-1a-mppg-syntax.mppg`	Listing 1a (syntax comparison, MPPG script)
`listing-3-wcd-single-node.mppg`	Listing 3 (WCD bounds, single node)
`listing-4-wcd-pboo.mppg`	Listing 4 (WCD bounds, two nodes, PBOO)
`listing-5-rsc-strict.mppg`	Listing 5 (Residual service curve, strict)
`listing-6-subadditive-convolutions.mppg`	Listing 6 (Subadditive convolution)
`listing-7-super-isospeed.mppg`	Listing 7 (Super-isospeed convolution)

Option B – Use the provided Dockerfile

The provided Dockerfile sets up a self-contained environment with all necessary tools pre-installed, including .NET 8, 9 and 10, PowerShell 7, and all dependencies for testing and coverage. This method is recommended to verify the quantitative claims, as it makes easier to run the provided scripts.

Build the Docker image from the root of the repository:

```
docker build . -t nancy-playground
```

Then start a container with the repository mounted:

```
docker run -it -v $(pwd .):/home/dotnet/nancy-playground-artifact \
    -w /home/dotnet/nancy-playground-artifact nancy-playground bash
```

`nancy-playground` can also be installed from NuGet inside the container as described for Option A, but note that `plot()` commands may not produce visible windows in this environment. To run `nancy-playground` from source (inside the container, or with the .NET 10 SDK installed):

```
dotnet run --project Nancy-Playground/Nancy-Playground/Nancy-Playground.
    csproj \
    --framework net10.0 -- <command>
```

3.2 Qualitative Claims

The behavior of `nancy-playground` can be verified by running the example scripts corresponding to the listings of [5] and by using the interactive console. The `paper-examples/` directory contains the MPPG scripts corresponding to the listings of [5]. Table 1 shows each file and its corresponding listing. The results of these scripts should then be matching the ones produces by the RTaW min-plus playground [2] on the same script, which is the basis for the test suite of `nancy-playground`.

Any script can be directly run with

```
nancy-playground run paper-examples/<filename>.mppg
```

To start an interactive session, instead, use

```
nancy-playground interactive
```

Then type MPPG statements one at a time. Use `!help` to access the integrated manual; `!exit` or `Ctrl+C` to quit.

To convert an MPPG script to C#, use the `convert` command, which produces a .cs file with the same name as the input script.

```
nancy-playground convert ./paper-examples/listing-1a-mppg-syntax.mppg
```

This produces `listing-1a-mppg-syntax.mppg.cs`, which can be executed directly with:

```
dotnet run ./paper-examples/listing-1a-mppg-syntax.mppg.cs
```

The output should match that of the `run` command on the same script.

3.3 Quantitative Claims: Tests and Coverage

Run the following script to execute the test suite and generate the coverage report (takes approximately ten minutes):

```
./run-tests-and-coverage-report.ps1
```

The script reports whether all tests pass and prints a coverage summary. A detailed interactive breakdown can be viewed by opening `coveragereport/index.html` in a browser. These results correspond to Table 7 of [5]: all tests should pass, and the coverage figures should match those reported in the paper.

3.4 Quantitative Claim: Performance Benchmark

The following script benchmarks execution time and peak memory usage for three scenarios: `nancy-playground` interpreting the script via `run`, `nancy-playground` converting the script via `convert`, and the resulting compiled C# program.

```
./benchmark-np-run-vs-convert.ps1 ./paper-examples/guidolin--pina-phd-
    listing-b.1.mppg
```

The benchmark can be run on any MPPG script; the file above is the one used for Table 6 of [5], i.e. Listing B.1 from [1].

4 Expected Results

All MPPG scripts, including those in `paper-examples/`, should run without errors, and produce the same results as the RTaW min-plus playground [2]. The C# program produced by `convert` should also yield the same output as the corresponding `run` invocation.

Regarding the test suite, which includes examples of the above comparison between outputs, all tests should pass. Coverage results should match Table 7 of [5], with approximately 67% line coverage overall.

Regarding the performance benchmark, absolute values depend on the hardware, but the overall trend should match Table 6 of [5]: the `run` command is measurably less efficient than converting and running a C# program, however this overhead should appear negligible in the context of a tool used directly by researchers for prototyping and experimentation.

5 Documentation

The `syntax.md` file in this repository documents all supported MPPG constructs. The online documentation for `Nancy` and related tools, including `nancy-playground`, is available at `https://nancy.unipi.it`. The integrated help is accessible at any time by running `nancy-playground interactive` and typing `!help`.

6 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

7 License

The artifact is available under the MIT license.

8 MD5 sum of the artifact

4eaca52b4e28d662dbfbc390a559a4ab

9 Size of the artifact

289 KB

References

1 Damien Guidolin-Pina. *Evaluation of worst-case performances of systems in Network Calculus*. Phd thesis, ISAE – Institut Supérieur de l'Aéronautique et de l'Espace, November 2024. URL: https://hal.science/tel-05023338.

2 RealTime-at-Work. RealTime-at-Work online Min-Plus interpreter for Network Calculus. URL: https://www.realtimeatwork.com/minplus-playground.

3 Andrea Trasacco, Raffaele Zippo, and Giovanni Stea. Nancy.Expressions: Towards a Computer Algebra System for Deterministic Network Calculus. In *EAI VALUETOOLS 2024*, 2024. doi:10.1007/978-3-032-06818-7_23.

4 Raffaele Zippo and Giovanni Stea. Nancy: An efficient parallel Network Calculus library. *SoftwareX*, 19:101178, 2022. doi:10.1016/j.softx.2022.101178.

5 Raffaele Zippo and Giovanni Stea. Nancy-Playground: A Console Calculator for Deterministic Network Calculus. In *38th European Conference on Real-Time Systems (ECRTS 2026)*, Leibniz International Proceedings in Informatics (LIPIcs), 2026. doi:10.4230/LIPIcs.ECRTS.2026.5.

6 Raffaele Zippo and Giovanni Stea. Nancy-Playground artifact repository archived on Software Heritage, 2026. URL: https://archive.softwareheritage.org/swh:1:dir:c4558a296698005f518856f369ed6e0f3bf4dac8.

7 Raffaele Zippo and Giovanni Stea. Nancy-Playground artifact repository on GitHub. https://github.com/rzippo/nancy-playground-artifact, 2026. URL: https://github.com/rzippo/nancy-playground-artifact.

8 Raffaele Zippo and Giovanni Stea. Nancy-Playground binary packages on NuGet. https://www.nuget.org/packages/Unipi.Nancy.Playground.Cli, 2026. URL: https://www.nuget.org/packages/Unipi.Nancy.Playground.Cli.