Controlling Adaptive HARQ Erasure Coding for Real-Time Transport under Channel Model Mismatch (Artifact)

Moritz Miodek
Saarland Informatics Campus, Saarland University, Saarbrücken, Germany

Marlene Böhmer
Saarland Informatics Campus, Saarland University, Saarbrücken, Germany

Thorsten Herfet
Saarland Informatics Campus, Saarland University, Saarbrücken, Germany

Abstract

This artifact contains the source code of the PRRT protocol implementation, and evaluation scripts in the form of Jupyter notebooks to reproduce Figure 2, Figure 3, Figure 4, and Table 2 presented in the paper. The evaluation is split into two parts: evaluating the PRRT network protocol operating over a lossy loopback device (using Linux traffic control), and evaluating the performance aspects of the incremental search in isolation. The artifact contains two repositories, the PRRT protocol implementation, and the evaluation scripts.

Running the complete suite of experiments to reproduce all paper results takes approximately 1 hour on the specified hardware.

The evaluation is sensitive to the real-time capabilities of the underlying operating system. To faithfully reproduce the results of the paper, we recommend using the existing real-time capabilities of modern Linux kernels, and provide a Docker image to ease the installation process compared to the native setup. We also provide the OVA for cross-platform evaluation purposes, it allows to functionally reproduce the results, but may produce degraded results depending on the host and hypervisor real-time capabilities.

2012 ACM Subject Classification Networks → Network reliability; Networks → Transport protocols; Computer systems organization → Real-time systems

Keywords and phrases Transport protocol, Real-time systems, Real-time communication, Linux, Rust, Network reliability, adaptive erasure coding, HARQ, closed-loop control, anytime search, model mismatch

Digital Object Identifier 10.4230/DARTS.12.2.5

Related Article Moritz Miodek, Marlene Böhmer, and Thorsten Herfet, "Controlling Adaptive HARQ Erasure Coding for Real-Time Transport Under Channel Model Mismatch", in 38th European Conference on Real-Time Systems (ECRTS 2026), LIPIcs, Vol. 375, pp. 7:1–7:24, 2026.
`https://doi.org/10.4230/LIPIcs.ECRTS.2026.7`

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

1 Scope

This artifact contains the source code of the PRRT reference implementation, and accompanying evaluation data and scripts in the form of Python Jupyter notebooks. The evaluation scripts allow reproducing the paper's figures, namely Figure 2, Figure 3, Figure 4 and Table 2. We also provide the data from our execution that was used to produce the paper figures and table. Figure 2 and Figure 3 focus on the evaluation of the proposed adaptive control scheme within the PRRT

© Moritz Miodek, Marlene Böhmer, and Thorsten Herfet; licensed under Creative Commons License CC-BY 4.0

protocol, while Figure 4 and Table 2 evaluate the incremental search algorithm in isolation. We also provide source code for a minimal sender and receiver application to provide an example on how to use the PRRT protocol outside the evaluation script framework.

2 Content

The artifact contains the code and data to reproduce the results of the paper in `prrt-eval/`. For convenience, we further provide the artifact as an OVA image and as a pre-build Docker image. The two images were built using the native setup instructions of the artifact.

- `prrt-eval/` The artifact, containing the relevant source code and evaluation data.
- `ecrtsVB.ova` The pre-build OVA image containing the artifact.
- `prrt_ecrts_artifact_docker_240sTraces.tar` The pre-build Docker image containing the artifact.

2.1 PRRT Protocol Implementation (`./prrt-eval/prrt/`)

The PRRT Protocol Implementation contains the PRRT-rs reference implementation source code, including standalone usage examples.

- `./prrt-eval/prrt/` contains the workspace (groups of library crates) of the PRRT reference implementation, which the evaluation harness evaluates.
- `./prrt-eval/prrt/prrt/` contains the PRRT reference implementation main crate.
- `./prrt-eval/prrt/prrt-bin/src/{sender.rs,receiver.rs}` implement a minimal, sender and receiver application as a standalone demo of the protocol. Consult `prrt/README.md` for how to run them.

2.2 Evaluation Harness (`./prrt-eval/`)

This Evaluation Harness contains the scripts, data, and testing binaries used to run the experiments and generate the paper's results. It is located in `./prrt-eval/`.

- `README.md` contains the Native Quick-start guide, a project overview and additional information.
- `src/main.rs` is the parameterized evaluation binary that the Jupyter notebooks call to evaluate the PRRT protocol.
- `inc_search_eval/` is a Rust crate that evaluates the PRRT incremental search and its non-greedy variant.
- `python/` contains the Python scripts in the form of Jupyter notebooks to reproduce Figures 2, 3, and 4 and Table 2 of the paper:
  - `eval_missmatch.ipynb` generates Figure 3
  - `eval_convergence.ipynb` generates Figure 2
  - `execution_time_CDF.ipynb` generates Table 2
  - `incremental_search_eval.ipynb` generates Figure 4
- `python/traces/` contains the data that was used to generate the paper Figure 2 and Figure 3.
- `python/inc_search_eval/paper_data/` contains the data that was used to generate the paper Figure 4 and Table 2.
- `Dockerfile` can be used to build a docker image to simplify the setup phase (alternatively, use the provided pre-build image).

The `eval_missmatch.ipynb` and `eval_convergence.ipynb` notebooks define respective experiments which are defined by the network configuration (loss rate and delay) and the PRRT application parameters (target loss, target delay, source packet interval). The parameterized `src/main.rs` parses these inputs, instantiates the experiment and simulates it. This generates traces, which the notebooks visualize to produce Figures 2 and 3 in the paper. The experiments run on the loopback device, and losses/delays are introduced with Linux traffic control (tc/netem).

The `execution_time_CDF` notebook evaluates the execution time of the incremental search algorithm using the `inc_search_eval/` crate. The crate compares the quality of the incremental search to its non-greedy variant across different ARQ cycle limits. See `inc_search_eval/` for the data generation source code. The `inc_search_eval` notebooks visualize this data and produce Table 2 and Figure 4 of the paper.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available on GitHub: The evaluation code lives inside `https://github.com/miodic/prrt-eval-artifacts/tree/ECRTS26AE`, and the PRRT Rust implementation at `https://github.com/miodic/prrt/tree/ECRTS26AE`.

4 Tested platforms

This artifact has been tested on Gentoo and Debian 13.4.0 running on a 16-core AMD Ryzen 5950x with 32GB DDR4 RAM.

4.1 System Requirements

To have enough space for the evaluation resources and results, the hardware must have at least 30 GB of available disk space, 32 GB (DDR4 or DDR5) RAM, and at least 8 CPU cores, preferably on a desktop, workstation, or server platform. We recommend avoiding big/little CPU core architectures, and thus recommend AMD Zen 3,4,5 processors, or Intel Gen 12+ CPUs with efficiency/little cores disabled. The evaluation code targets the Linux operating system. The artifact can be evaluated on any modern Linux OS running kernel 6.12+ with the following Real-time Kernel Requirements. We recommend using Debian 13 (Trixie) and installing the linux-image-rt-amd64 kernel package.

4.2 Real-time Kernel Requirements

The PRRT protocol implementation of the artifact is sensitive to the real-time scheduling capabilities of the underlying host system. For Linux, the artifact does not require hard real-time capabilities, it is sufficient to enable full dynamic preemption (preempt=full in `/etc/default/grub`) for PREEMPT_DYNAMIC kernels. This provides a softer low-latency preemption mechanism without needing to replace the kernel. That said, Debian 13 does provide the full RT-kernel via the `linux-image-rt-amd64` package. For instructions on how to configure your system, consult the `README.md` inside `prrt-eval`.

4.3 Compatibility

1. If you have less than 24 GB of RAM, edit `prrt-eval/prrt/prrt/src/constants.rs` and lower "TRACE_DURATION" from "240" seconds to "180" seconds, or at the lowest "130" seconds for the evaluation to run. This should reduce the memory usage to roughly 10GB for '130' seconds, and 13GB for '180' seconds. In this case, we do not recommend using the Docker image.
2. If you run on old/slow hardware or use virtualization, you might run into the issue that your kernel networking stack starts dropping packets, which might lead to congestion collapse in the evaluation. This is likely due to the additional overhead or default configuration of tc/netem, which is used under the hood to simulate network losses. Make sure to increase "SLOWDOWN_FACTOR" in the convergence or mismatch notebooks, which slows down the packets per second, to whatever your hardware can sustain.

5 License

The artifact is available under the MIT license

6 MD5 sum of the artifact

175c4f701255fa49570992cd6c57d5c6

7 Size of the artifact

3.44 GiB