Achieving Isolation in Mixed-Criticality Industrial Edge Systems with Real-Time Containers (Artifact)

Marco Barletta
Università degli Studi di Napoli Federico II, Italy

Marcello Cinque
Università degli Studi di Napoli Federico II, Italy

Luigi De Simone
Università degli Studi di Napoli Federico II, Italy

Raffaele Della Corte
Università degli Studi di Napoli Federico II, Italy

Abstract

Real-time containers are a promising solution to reduce latencies in time-sensitive cloud systems. Recent efforts are emerging to extend their usage in industrial edge systems with mixed-criticality constraints. In these contexts, isolation becomes a major concern: a disturbance (such as timing faults or unexpected overloads) affecting a container must not impact the behavior of other containers deployed on the same hardware. In this artifact, we propose a novel architectural solution to achieve isolation in real-time containers, based on real-time co-kernels, hierarchical scheduling, and time-division networking. The architecture has been implemented on Linux patched with the Xenomai co-kernel, extended with a new hierarchical scheduling policy, named `SCHED_DS`, and integrating the RTNet stack. Experimental results, presented in the related scholarly paper, are promising in terms of overhead and latency compared to other Linux-based solutions. More importantly, the isolation of containers is guaranteed even in presence of severe co-located disturbances, such as faulty tasks (elapsing more time than declared) or high CPU, network, or I/O stress on the same machine.

2012 ACM Subject Classification Software and its engineering → Real-time systems software

Keywords and phrases Real-time, Mixed-criticality, Containers, Edge computing

Digital Object Identifier 10.4230/DARTS.8.1.1

Acknowledgements This work has been partially supported by the project COSMIC of UNINA DIETI and by the R&D project "REINForce: REsearch to INspire the Future" (CDS000609), funded by the Italian Ministry for Economic Development (MISE).

Related Article Marco Barletta, Marcello Cinque, Luigi De Simone, and Raffaele Della Corte, "Achieving Isolation in Mixed-Criticality Industrial Edge Systems with Real-Time Containers", in 34th Euromicro Conference on Real-Time Systems (ECRTS 2022), LIPIcs, Vol. 231, pp. 15:1–15:23, 2022.
`https://doi.org/10.4230/LIPIcs.ECRTS.2022.15`

Related Conference 34th Euromicro Conference on Real-Time Systems (ECRTS 2022), July 5–8, 2022, Modena, Italy

1 Scope

This artifact has the aim to back the implementation part of our scholarly paper. In particular, we provide all the code we used to set up the system and to obtain the results shown in the experimental section, along with step-by-step instructions on how to use it.

2 Content

The artifact package is composed by the following folders:

- `Documentation`, containing this guide.
- `testTimes`, containing the scripts and applications to run the experiment described in the Section 5.1 of the paper.
- `testOverhead`, containing the scripts and applications to run the experiment described in the Section 5.2 of the paper.
- `testIsolation`, containing the scripts and applications to run the experiment described in the Section 5.3 of the paper.
- `testRtnet`, containing scripts, applications and source code to run the experiment described in Section 5.4 of the paper.
- `testLatency`, containing the scripts and applications to run the experiment described in the Section 5.5 of the paper.
- `Src`, containing the source code of the applications used, the patches and the rt-lib API.

The same files can be found within the virtual machine, in the path `/home/ecrts/Documents`. Since absolute paths are used in the scripts (docker volumes require them), if the folders and the files are moved, take care of following carefully the instructions in the "Installation of the artifact" section to change the paths in the scripts. In the following, we detail the content of each folder.

- `Documentation`:
  - `Xeno-containers_artifact_guide.pdf`: file containing this guide.
- `testTimes`:
  - `testempi_23/46/69`: programs that create the thread groups and generate the workload needed for the test, respectively 2 groups * 3 threads, 4 groups * 6 threads, 6 groups * 9 threads.
  - `testempi_23/46/69.c`: sources of the programs.
  - `analysis.py`: Python script that analyzes results and plots the figure.
  - `makefile` to compile programs.
  - `test.sh`: Bash script that runs the test and generates results in files.
  - `mixed`: folder containing the scripts and the executable to run the test with mixed workload, generated through rt-app. In particular, it contains test.sh with the same aim of the script in the upper folder (it also copies results in the upper folder), and some utility scripts.
- `testOverhead`:
  - `period.py`: Python script that analyzes results by reading the files in the generated subfolders and plots results.
  - `testOverhead.py`: Python script that runs the overhead experiment, generating results in three subfolders, depending on the yielding period described in the paper.
  - `testQuotaMod1/01/10`: programs that execute one test. Each program corresponds to a different yielding period.
  - `testOverhead_stress.py`: Python script that runs the overhead test under stress, generating results in the load subfolder with the same pattern as the upper folder.
  - `load`: folder that contains results of the overhead test under stress generated by stress-ng and period.py, that has the same aim as the script in the upper folder.
- `testIsolation`:
  - `analysis.py`: Python script that analyses results by reading the files in the generated subtree and plots results.
  - `startup / shutdown` described later, in Src section.
  - `taskgen.py`: Python script, implementation of the RandFixedSum algorithm cited in the paper, and used by test.py to generate tasksets.

  - `test.py`: Python script that runs the isolation test in the “low-high” scenario described in the paper. Results are generated in a subtree folder named results. It enforces the schedulability test described in the paper.
  - `stress`: folder containing both the scripts code and executables needed to run the Isolation test under the workloads generated by stress-ng. Results are generated in a subtree folder named results. The analysis.py script and taskgen.py script have the same aim of the upper folder.
  - `porting`: folder with the experiment of the upper folder, but using the rt-app_porting, i.e., the rt-app that runs in docker containers and exploits our rt-lib for a porting with minimal modifications. The script has the aim of functionally testing the rt-lib, results of the paper are not generated with this script. Since the rt-lib only operates out of critical paths, there are no differences in performance.
- `testRtnet`
  - `startup/shutdown` described later, in Src section.
  - `testnetserver / testnetserver.c`: application to run on server side.
  - `testnet / testnet.c`: application to be run on client side. It is called from the Python script.
  - `makefile` to compile executables.
  - `APIc.h / list.wrappers`: porting layer for containers, the rt-lib described in the paper.
  - `netanalysis.py`: Python script that analyzes a pcap file containing traffic traces and outputs results.
  - `testTDMA.py`: Python script that runs the test, opening containers and sending traffic.
  - `rtnet.conf / rtnetserver.conf`: configuration files for RTnet on both client and server.
  - `tdma.cfg`: configuration file for the TDMA slotting.
  - `rtnet`: script to setup the RTnet.
- `testLatency`:
  - `hcbs`: folder containing the script and the application needed for the latency test in the HCBS kernel. Results for this test are generated in a subfolder tree called results.
  - `xenomai`: folder containing the script and the application needed for the latency test in the xenomai kernel. Results for this test are generated in a subfolder tree called results.
  - `vanilla`: folder containing the script and the application needed for the latency test in the low latency kernel. Results for this test are generated in a subfolder tree called results.
  - `analysis.py`: Python script that reads the result subtrees in the three folders and plots the data.
- `Src`:
  - `Kernels`: folder containing archives of the Linux and Xenomai source code, along with the .config files of our setup.
  - `Patches`: folder containing
    - `quota.patch`, our `SCHED_QUOTA` patch;
    - `quota_times.patch`, the patch for runtime sampling (Section 5.1 of the paper);
    - `ipipe-core-5.4.77-x86-2.patch`, the ipipe patch for the Linux kernel we used;
    - `LinuxPatches-HCBS.zip` containing the patch set for HCBS for Linux 5.2.8;
    - `rtapp.patch`, to patch rt-app makefile for Xenomai recompilation after a `./configure`;
    - `rtapp_porting.patch`, to patch rt-app_porting;
    - `patchHCBS.sh`, a script to apply HCBS patch set.
  - `apps`, folder containing:
    - `rt-app` is our version of rt-app extended with `SCHED_QUOTA` (`SCHED_DS`) policy, the vanilla version of rt-app available at [4];

  - `rt-app_porting` is the version of rt-app that exploits our rt-lib for a clean porting to run in containers, with minimal modifications. Since the rt-lib only operates out of critical paths, there are no differences in performance;
  - `testQuotaMod01/1/10.c` that are the source codes used for the overhead test;
  - `Makefile` to compile Xenomai applications;
  - `latencyMod.c` the source code for latency test, modified for `SCHED_QUOTA`;
  - `startup.c / shutdown.c`, source codes for the executables that bring up/tear down quota groups;
  - `testempi_23.c/46.c/69.c`, source codes for the executables used in testTime experiment. testempi_23 generates workload with 2 groups * 3 thread, _46 a workload with 4 groups*6 threads and _69 a workload with 6 groups * 9 threads.
  - `Linux`, a folder containing latency.c, i.e., the source code for latency test, (Low latency and vanilla) and its makefile.
- `rt-lib`: folder containing `APIc.h`, i.e., the file that implements the rt-lib in our proposed architecture, list.wrappers, that contains a file necessary to Makefile to wrap additional functions and Makefile, that contains a simple makefile that shows how to compile a Xenomai application with additional lines for rt-lib wrapping.

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 latest version of the artifact is also available at: `https://dessert.unina.it:8088/marcobarlo/xeno-containers`. We also provide a Virtual Machine (VM) with all the software we used inside for ease of use and functional testing. The VM can be downloaded at the link in [2]: The password to get the .ova file is `ecrts`; The user for the VM is `ecrts` and the password is `ecrts`. The virtual machine we set up has the following hardware configuration:
- A Multi-core processor (4);
- 8 GB RAM;
- $>$ 17 GB HDD (needed to compile the kernels).

The multi-core processor is not a strict constraint, however our configuration commands assume having a multi-core processor. It also guarantees that Linux does not starve even when a core is dedicated to real-time.

4 Tested platforms

In order to run the experiments detailed in the Section 5 of our paper, it is mandatory to setup the execution environment needed. To fully reproduce results similar to ours, an execution environment on a physical machine would be needed, as we done for the experiments of the paper, avoiding the usage of VM. Indeed, the usage of Xenomai (i.e., the real-time co-kernel extension for Linux we used) within a VM does not allow obtaining satisfying performance in terms of both latency and overruns. Please note that it is extremely hard to obtain exactly the same results indicated in the table and figures of the paper. In fact, results strictly depend on the available execution environment, and especially on the configuration of the physical machine running the environment.

As specified in the paper, our results have been obtained with a machine equipped with:
- Intel Core i5-6500 processor (4 physical cores, no hyperthreading, up to 3.2 GHz);
- 16 GB DDR4 RAM;
- 512 GB Samsung 980EVO SSD.

The execution environment we used encompasses the following software:

- Linux Mint 20.3 as Linux distro;
- Linux kernels:
  - Linux kernel 5.4.77 patched with Xenomai (ipipe [11]) for this specific kernel version;
  - Linux kernel 5.4.77 in low-latency preemption mode;
  - Linux kernel 5.2.8 in low-latency preemption mode and patched with HCBS [1] patch;
- Xenomai 3.1 [12];
- Python 3.7.3 with libraries such as Numpy, Scipy and Matplotlib;
- Dockstation, to check container status at a glance; [10]
- The applications and scripts that we use in the tests, available in our artifact.

It is important to note that the Linux kernel 5.4.77 must be patched with the specific ipipe patch, i.e., the patch for the 5.4.77 kernel version, and re-compiled. The patched kernel must be selected during the boot of the system. Since Xenomai versioning is independent of Linux ipipe patch, the kernel version 5.4.77 is not a strict constraint, however that is the one we used. **The Xenomai version, i.e., 3.1 or 3.1.1, is instead a strict constraint**, since our patches are not tested on other versions. In the VM we provide, the kernel patched with Xenomai comes with the relative build tree, since few recompilations are needed to reproduce our tests.

The VM is shipped with two Xenomai kernels:

- the one tagged “Xenomai” is the kernel used for application level tests;
- the one tagged “Xenomai-TIMES” is the kernel instrumented with time sampling for the evaluation of execution times of the scheduler.

The VM could appear with a black screen after startup. In that case, go to “View” options of the VirtualBox window and change the screen size.

5 License

The artifact is available under license General Public License v2.

6 MD5 sum of the artifact

2021a1c085510b4efe773fe73627973b

7 Size of the artifact

0.32 GiB

A Kernel Building and Configuration details

You can skip this section if you use the provided VM, since kernels are already built on it.

Note on configuration. The main changes to the all kernel regard:

- The Low latency preemption mode (-> General setup -> preemption model)
- Disable SCHED_MC_PRIO (-> Processor type and features -> Multi-core scheduler support -> CPU core priorities scheduler support)
- Disable CPU_FREQ (-> Power management and ACPI options -> CPU Frequency scaling)
- Disable ACPI_PROCESSOR (-> Power management and ACPI options -> ACPI -> Processor)
- Disable CPU_IDLE (-> Power management and ACPI options -> CPU Idle)
- Disable CONFIG_APM (-> Power management and ACPI options -> APM)
- Disable debugging modes.

To retrieve useful packages for kernel building and experiment execution, you can use the command:

```
sudo apt-get install git fakeroot build-essential ncurses-dev \
                xz-utils libssl-dev bc flex libelf-dev bison \
                autogen libtool autoconf qt5-default docker \
                docker.io  libjson-c-dev stress-ng gcc-8 socat
```

A.1 Xenomai

To build the Xenomai environment from scratch, we suggest following the whole official guide [9], paying great attention to software/hardware "latency killers", and to kernel fine-tuning on your hardware. Hardware latency killers can be modified from UEFI page and are strictly dependent on your hardware. We disabled Turboboost, SpeedStep, intel c-states and other power optimization settings. We also pinpoint that laptops may have custom hardware with power optimization settings difficult to disable.

NOTE: During Xenomai configuration, keep in mind to **enable SCHED_QUOTA** (-> Xenomai/cobalt (XENOMAI) -> Core features -> Extra scheduling classes -> **Thread groups with runtime quota**, in the kernel config menu) and tune **the quota period** (-> Thread groups with runtime quota -> **Quota Interval**, in the kernel menu config) for the various experiments. We give more details on this in the section "Run the experiments". Keep in mind that we provided our .config files in the artifact, if needed.

Before building, remember to patch SCHED_QUOTA by running the following command in the Xenomai top folder (that is, in our VM, `/home/ecrts/Documents/xenomai-stable-v3.1.x`), after moving the patch in the upper folder (`/home/ecrts/Documents/`):

```
patch -p2 < ../quota.patch
```

To enable the instrumentation for SCHED_DS times (needed for testTimes experiment), run the following command in the Xenomai top folder (`/home/ecrts/Documents/xenomai-stable-v3.1.x` in our VM), after moving the patch in the upper folder (`/home/ecrts/Documents/`):

```
patch -p1 < ../quota_times.patch
```

After the kernel configuration, guided by the official site, the kernel can be installed through:

```
sudo make bzImage modules
```

and then

```
sudo make modules_install install
```

launched in the linux-5.4.77 folder. Keep attention to modules compiled: if there are too many unneeded modules, the system may not be able to boot! To avoid this problem you can use `make localmodconfig`, but make sure you have all modules needed loaded (e.g., run a docker container to load veth module).

When installing libraries (a step of the official guide), use this configuration command (in the Xenomai top folder), since Python scripts rely on it (in particular the –prefix=/usr/xenomaiB option is important):

```
./configure --with-core=cobalt --disable-debug --enable-pshared --enable-smp \
--prefix=/usr/xenomaiB
```

Finally, after the installation, remember to set your user [7] as real-time user and to set udev rules [8] to access /dev/rtdm, in order to properly run tasks in containers. A default file that assumes that the user belongs to a group called `xenomai` is in `/usr/xenomaiB/etc/udev/rules.d/rtdm.rules`, and must be moved to `/etc/udev/rules.d/`.
Change the group name at need and then reboot the system to make modifications effective.

A.2 Low Latency

For the installation of the low latency kernel is enough to make the general changes described above. Keep in mind that we provided our .config files in the artifact, if needed.

A.3 HCBS

After extracting the linux-5.2.8 zip, in the path `YOUR_PATH`, such that linux root folder is `YOUR_PATH/linux-5.2.8`:

1. extract the HCBS patchset in the same folder (`YOUR_PATH/LinuxPatches-HCBS`)
2. then run the `patchHCBS.sh` script in `YOUR_PATH`.

The configuration is the same of the low latency version. Keep in mind that we provided our `.config` files in the artifact, if needed. Before building the kernel, keep in mind that the compiler may find errors. To avoid them, the gcc version must be changed to gcc-8 with the command:

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-8 10
```

B Installation of the artifact

In the following, you can find instructions for installing the artifact. *You can skip this section if you use the VM since it is already installed.*

Once the zip file containing the artifact has been downloaded, the following steps allow installing the artifact:

1. Unzip the zip file in a selected folder.
   In the following, we refer to this folder as `ART_INSTALL_FOLDER`.
2. Verify that the 4 folders included in the zip have been extracted into `ART_INSTALL_FOLDER`.
3. (Optional) If executables provided for some reason do not work, rebuild them through the make command in `ART_INSTALL_FOLDER/Src/Apps` and `ART_INSTALL_FOLDER/Src/Apps/Linux`.
   Replace the programs with the ones in the experiment folder, i.e.:
   - testQuotaMod01/1/10 into `ART_INSTALL_FOLDER/testOverhead`;
   - latencyMod into `ART_INSTALL_FOLDER/testLatency/xenomai`;
   - latency into `ART_INSTALL_FOLDER/testLatency/vanilla` and `ART_INSTALL_FOLDER/testLatency/hcbs`;
   - testempi_23/46/69 into `ART_INSTALL_FOLDER/testTimes`;
   - startup and shutdown into `ART_INSTALL_FOLDER/testIsolation` , `ART_INSTALL_FOLDER/testIsolation/stress`, `ART_INSTALL_FOLDER/testLatency/xenomai` and `ART_INSTALL_FOLDER/testTimes/mixed`.
   - To install Xenomai rt-app (already installed in the VM):
     - move into rt-app folder `cd ART_INSTALL_FOLDER/Src/Apps/rt-app`
     - open a terminal and run `./autogen.sh`
     - run `./configure`

   - move the rtapp.patch to `ART_INSTALL_FOLDER/Src/Apps`
     `cp ART_INSTALL_FOLDER/Src/Patches/rtapp.patch  ART_INSTALL_FOLDER/Src/Apps/`
   - run `patch -p1 < ../rtapp.patch`
   - `sudo make && sudo make install`
   - the same procedure is to be done for rt-app_porting, **BUT DO NOT INSTALL IT**, scripts assume having the other rt-app version installed. The patch to use with rt-app_porting is rtapp_porting.patch.
4. Modify the paths in the test.py script in the testIsolation/porting folder to test the rt-lib API:
   - Open a terminal
   - Move in the `ART_INSTALL_FOLDER/testIsolation/porting` folder:
     `cd ART_INSTALL_FOLDER/testIsolation/porting`
   - Edit the this_path and rt_app_path variables at first lines after imports of the `test.py` script
5. Modify the paths in the testLatencyContainer.py scripts in the testLatency folder for each subfolder:
   - Open a terminal
   - Move in the `ART_INSTALL_FOLDER/testLatency` folder:
     `cd ART_INSTALL_FOLDER/testLatency`
   - Edit the this_path variable at the first line after the imports of the script `vanilla/testLatencyContainer.py` and set its value at `ART_INSTALL_FOLDER/testLatency`
   - Do the same for xenomai and hcbs folders.

Before going deep into experiments reproduction, there is one last step of configuration. The file `/etc/security/limits.conf` must be modified. The following lines must be added:

```
root hard memlock 1048576
root soft memlock 1048576
root hard rtprio 99
root soft rtprio 99
ecrts hard memlock 1048576
ecrts soft memlock 1048576
ecrts hard rtprio 99
ecrts soft rtprio 99
```

After a log out, these parameters allow both the “ecrts” user (replace the ecrts username with yours) and the root user to run tasks with maximum real-time priority equal to 99 and to lock a great quantity of memory.

Also, remember to make your non-root user able to use docker. A guide is available at [3].

C Run the experiments

In the following, we describe the steps to run the experiments presented in the Section 5 of the paper.

NOTE: As already stated at the beginning of this guide, it is important to note that it is difficult to obtain the exactly same results presented in the paper. In fact, results strictly depend on the available execution environment, and especially on the configuration of the physical machine running the environment. Moreover, the rt-tasks emulating the task composing the taskset should be calibrated for the specific machine. analysis of our artifact.

C.1 Experiments in Section 5.1 – Scheduler runtimes

NOTE: This experiment should be executed on top of the Xenomai kernel tagged with Xenomai-TIMES. Reboot your system and select the kernel under "Advanced Options for Ubuntu", in the grub menu. (In VirtualBox case, after focusing with pointer on the VM, hold shift at VM startup to enter grub). **Remember to switch back to the kernel without time instrumentation after the test.** The kernel logs the scheduler times, thus disk saturates very quickly if the log isn't kept under control.

In order to reproduce the experiments in Section 5.1, the following steps should be followed:

1. Open a terminal
2. Move into the `ART_INSTALL_FOLDER/testTimes` folder
   `cd  ART_INSTALL_FOLDER/testTimes`
3. Run the script test.sh as superuser
   `sudo ./test.sh`
   NOTE: This script flushes the kernel log before the test and then runs the program to generate the workload. Then it copies the kernel log and stores it into a file. The files contain a sequence of lines that are "P [time in ns]" or "R [time in ns]", where P stands for pick and R for replenish.
4. Move into mixed subfolder
   `cd mixed`
5. Run the script test.sh as superuser
   `sudo ./test.sh`
   The script makes the same thing as before, but the workload is generated through a random taskset with rt-app.
6. Move back
   `cd ..`
7. Plot results through analysis.py script:
   `python3 analysis.py`

To modify the group/thread number, modify the run_quota function in `Src/Apps/testempi_xx.c` and create a new program. The diff (running `diff testempi_23.c testempi_46.c` in `Src/Apps` for example) of the three files can be used as guide to modify the function.

C.2 Experiments in Section 5.2 – Overhead test

In order to reproduce the experiments discussed at the beginning of the Section 5.2, the following steps should be followed:

1. Compile the kernel with the quota period you want to analyze. Details below*
2. Open a terminal
3. Move into the `ART_INSTALL_FOLDER/testOverhead` folder
4. Run the script testOverhead.py
   `python3 testOverhead.py`
5. Move into the 0.1ms 1ms and 10ms folder that have been generated, and rename the file called results in results_[quota period of the running kernel in ms]. For example, on the kernel shipped with the VM the quota period is 2.5 ms. Thus the results file in the three folders must be renamed "results_2.5". With a period of 1 ms adopt the convention "results_1", and so on.
6. Run the period.py script to generate the figure
   `python3 period.py`
7. Go back to point 1, selecting another period.

For the test under stress:

1. Run the script testOverhead_stress.py. It will generate results in the same way of the first script, but under load folder. This time, files must not be renamed, since they are called as results[load number].
2. Move into load folder:
   `cd  ART_INSTALL_FOLDER/testOverhead/load`
3. Run the period.py script to generate the figure.

*To compile the kernel with an appropriate quota_period, move into the linux-5.4.77 tree. In our VM: `cd /home/ecrts/Documents/Src/Kernels/linux-5.4.77` Then, configure the kernel `make xconfig` or `make menuconfig` Change the quota period under "-> Xenomai/cobalt (XENOMAI) -> Core features -> Extra scheduling classes -> Thread groups with runtime quota -> Quota Interval" Set the period in microseconds, so for a period of 2.5 ms -> 2500. Save the configuration of the kernel and run: `sudo make -j$(( $(nproc) + 1 )) bzImage` Where nproc is the number of processors of your machine. Then run: `sudo make install` . At this point, reboot the machine and select Xenomai in the "Advanced options" of the grub menu. To change the yielding period of the application, change the source code (`Src/Apps/testQuotaMod[xx].c`) at `line 68` (`"loops=crunch_per_sec / xx"`), that is the only line that differs between testQuotaMod01, testQuotaMod1 and testQuotaMod10. Consequently, modify both the `period.py` and `testOverhead.py` scripts.

Results file only contain percentages that are returned as result from the execution of testQuotaModxx executables.

C.3 Experiments in Section 5.3 – Isolation test

In order to reproduce the experiments in Section 5.3 on the Isolation test, the following steps should be followed:

1. Move into the `ART_INSTALL_FOLDER/testIsolation` folder
2. Run the script test.py
   NOTE: the script can take a very long time (3/4 hours). Modify the number of repetitions at the beginning of the script if there is the need for a smaller number of repetitions.
3. Run the script `analysis.py`. It generates the figure and prints tests with non-zero deadline misses. At the end, it prints the array of deadline miss by test for both the low-prio container and high-prio one.
4. Move into the stress folder.
5. Run the `stresstest.py` script
   NOTE: the script can take a very long time (14/15 hours). Modify the number of repetitions at the beginning of the script if there is the need for a smaller number of repetitions.
6. Run the `analysis.py` script. It prints the files that present non-zero deadline misses and the arrray of the number of deadline misses by test.

NOTE: The scripts test.py and stresstest.py impose the runtime calibration to the rt-app. We had to manually impose the calibration since, due to hardware unpredictability, the calibration returned 19 ns and 20 ns alternatively (the time needed to execute the minium batch of workload). Through kernel instrumentation, we noticed that with 19 ns tasks actually executed for a longer time than planned. **To remove manual calibration** and use **autocalibration**, remove lines 364 and 316 of test.py of and lines 335 and 383 of stresstest.py (calibration line in json files). This enlarges significantly the execution time of the scripts, since autocalibration may take also dozens of secons. We advise running a few autocalibrations, getting the results (can be read on the console output of rt-app), and then editing the value for manual calibration in the scripts.

The folder of the results is composed in this way: the first level of folders represent the actual load in terms of CPU utilization. The second level of folders are named by the repetition index. The third level of subfolder is named with the indexes of the loads. This level has a sense only for the stress test in the load folder. Finally in the folders there are the logs generated by rt-app. Each line of log contains some information. We are interestend in the slack parameter. In the first lines of the stresstest.py and test.py scripts are defined the parameters we used in the paper, you can modify them to change a little bit the experiment.

C.4 Experiments in Section 5.4 – RTnet test

The RTnet test is hard to reproduce, because of the necessary setup. Two machines are needed with appropriate Ethernet controllers and driver that must be supported by RTnet, and a switch downgraded to 100 MBits/s dedicated to real-time network. Any protocol different from RTnet cannot be used on the network. In a couple of VMs it doesn't work because of timing issues, that prevent from sending/receiving packets. We advise following official guides [5, 6] if you want to try to setup a RTnet testbed. However, we ship source code needed with instructions:

1. Move into the `ART_INSTALL_FOLDER/testRtnet`
2. Compile source codes if needed, use the provided makefile
3. Copy the `tdma.cfg` file into `/usr/xenomai/etc` on both client and server
4. Copy the `rtnet.conf` file into `/usr/xenomai/etc` on the client
5. Copy the `rtnetserver.conf` file into `/usr/xenomai/etc` on the server, renaming it into `rtnet.conf`
6. Run the script `rtnet` as superuser, following the guides [5, 6] to correctly set up the network and troubleshoot it
7. On the server, run `testnetserver`
8. On the client, run the python script `testTDMA.py`, changing the this_path variable if needed
9. At the same time, use a packet tracer such as Wireshark or tcpdump to capture traffic.
10. Analyze the created file with `netanalysis.py`, passing the file: `netanalysis.py -pcap [filename]`
    To run the script, scapy library is needed. Get it from here [1], otherwise you can incur in a known bug of previous versions.

C.5 Experiments in Section 5.5 – Latency the test

In order to reproduce the experiments in Section 5.5 on the latency comaprison, the following steps should be followed:

1. Boot the Xenomai kernel
2. Open a terminal
3. Move into the `ART_INSTALL_FOLDER/testLatency` folder
4. Move into `xenomai` subfolder
5. Run the `testLatencyContainer.py` script.
   NOTE: the script can take a very long time (6/7 hours). Modify the number of repetitions at the beginning of the script if there is the need for a smaller number of repetitions.
6. Reboot into the Low latency or HCBS kernels.
7. Open a terminal
8. Modify the docker daemon parameters. More on this at the end of this subsection*

[1] `https://scapy.readthedocs.io/en/latest/installation.html#current-development-version`

9. Move into the `ART_INSTALL_FOLDER/testLatency` folder
10. Basing on the kernel on top of which you are running, move into the appropriate folder. e.g. if you are on HCBS, move into hcbs subfolder.
11. Run the testLatencyContainer.py script.
12. Do the same for the remaining kernel
13. Move into the `ART_INSTALL_FOLDER/testLatency` folder
14. Run the script analysis.py. It generates the figure of comparison that you can find in the paper. It reads in all subtrees of results the worst latency written in the latency report, and then creates the boxplots.

The folder of the results is composed in this way: the first level of folders are named with the index of the co-located workload generated by stress-ng. The second level of subfolder are indexed by repetition number of the test. Within the folder, there is a single file that contains a report of the latency application.

*For the kernels exploiting rt-cgroups, the docker daemons parameters must be modified. In order to do this, we must modify the "/lib/systemd/system/docker.service" file. Thus, run the command sudo nano /lib/systemd/system/docker.service At this point find the line that starts by "ExecStart". Replace it with

```
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--cpu-rt-runtime=950000
```

Now the docker daemon must be reloaded. Reboot the system or simply run:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Do not forget to switch to the previous configuration when you return on the Xenomai kernel.

References

1 Luca Abeni et al. HCBS patch for Linux containers. `https://github.com/lucabe72/LinuxPatches/tree/Hierarchical_CBS-patches`. Accessed 17th June 2022.

2 Barletta, Marco and Cinque, Marcello and De Simone, Luigi and Della Corte, Raffaele. Artifact VM. `http://gofile.me/2wkf8/LXche0Ofa`. Accessed 17th June 2022.

3 Multiple contributors. Run docker as non root user. `https://www.thegeekdiary.com/run-docker-as-a-non-root-user/`. Accessed 17th June 2022.

4 Multiple contributors. Scheduler tools /rt-app. `https://github.com/scheduler-tools/rt-app`. Accessed 17th June 2022.

5 Philippe Gerum. Rtnet configuration. `https://source.denx.de/Xenomai/xenomai/-/wikis/RTnet_Conf`. Accessed 17th June 2022.

6 Philippe Gerum. Rtnet setup. `https://source.denx.de/Xenomai/xenomai/-/wikis/RTnet_Setup`. Accessed 17th June 2022.

7 Philippe Gerum. Running xenomai as regolar user. `https://source.denx.de/Xenomai/xenomai/-/wikis/Running_As_Regular_User`. Accessed 17th June 2022.

8 Philippe Gerum. Seting udev rules. `https://www.mail-archive.com/xenomai@xenomai.org/msg11928.html`. Accessed 17th June 2022.

9 Jan Kiszka. Installing xenomai 3. `https://source.denx.de/Xenomai/xenomai/-/wikis/Installing_Xenomai_3`. Accessed 17th June 2022.

10 Igor Kozlovsky and Pavel Lozko. Dockstation page. `https://dockstation.io/`. Accessed 17th June 2022.

11 Xenomai maintainers. Donwload page for ipipe patch. `https://xenomai.org/downloads/ipipe/`. Accessed 17th June 2022.

12 Xenomai maintainers. Xenomai download page. `https://xenomai.org/downloads/xenomai/stable/latest/`. Accessed 17th June 2022.