

# EDF Scheduling with Constant-Bandwidth Servers in Apache NuttX: Design, Implementation, and Virtual-Time Evaluation

Felipe Thomaz Pedroni

*Founder, ExtenSilica*

`felipe.pedroni@extensilica.com`

October 2026

## Abstract

We present a uniprocessor earliest-deadline-first (EDF) scheduling policy with per-task constant-bandwidth servers (CBS) for Apache NuttX. The integration combines absolute-deadline ordering, execution-time accounting, budget throttling, watchdog-based replenishment, repayment of execution overrun debt, and admission control capped at 95 percent of one processor for implicit-deadline reservations ( $D = P$ ). An injected-time transition check and deterministic QEMU instruction-count virtual time exercise the implementation at the scheduler and kernel levels. In a 1.8-second budget-enforcement experiment, two protected periodic tasks completed all 75 jobs on time while an over-demanding server generated 20 throttles and 19 replenishments within the observation window. In a separate higher-utilization, 190-job comparison on the same traced firmware, NuttX sporadic scheduling missed four deadlines; EDF/CBS met all 190. The released source, transition evidence, firmware identity, and traces establish a reproducible experimental baseline for deadline-oriented, budget-enforced scheduling in Apache NuttX.

**Keywords:** earliest-deadline-first, constant-bandwidth server, Apache NuttX, real-time scheduling, virtual-time evaluation, budget enforcement.

## 1 Introduction and contributions

Resource-constrained real-time systems must schedule periodic control work alongside tasks with variable or excessive execution demand. Deadline ordering identifies which job should execute next; execution reservations determine how much processor service each job may consume. Both mechanisms must operate together at dispatch, tick accounting, preemption, replenishment, and scheduler-lock boundaries. We implement that integration in Apache NuttX and expose its behavior through reproducible kernel-level experiments.

NuttX’s tasking manual documents strict priority, optional round-robin, and optional sporadic scheduling [5]. Rate-monotonic assignment can be used to choose those priorities. It is not a scheduling class [6]. Linux already ships an EDF policy whose deadlines are maintained by a constant-bandwidth server, with admission control [3], [4]. RTEMS documents a bandwidth server that drops an overrunning task to a background priority [7]. Zephyr’s ordinary deadline option is only a tie-break among equal static priorities [8]. A 2026 Zephyr paper then adds a separate, task-based server for aperiodic work [9]. Our contribution is an integrated, source-traceable EDF/CBS execution path for NuttX rather than a new EDF or CBS algorithm. Its distinctive engineering concerns are ready-queue eligibility, charging the running task, deferring an exhausted-budget throttle across `sched_lock()`, and repaying accumulated overrun without creating additional budget.

The contributions are:

1. A uniprocessor `SCHED_DEADLINE` integration: eligible-deadline order, accounting of the task that is running, throttling when the budget is exhausted, watchdog replenishment, and a reservation ledger capped at 950000 parts per million of one CPU. This revision supports implicit deadlines only ( $D = P$ ); rejected attribute updates leave the previous reservation and ledger charge unchanged.
2. An integrated virtual-time budget-enforcement experiment with paired protected-task releases, observable throttling and replenishment of an over-demanding server, and next-grant repayment of recorded overrun. This experiment directly observes budget enforcement on the measured workload.

3. One matched comparison, on that same firmware, between NuttX `SCHED_SPORADIC` and `SCHED_DEADLINE` on a periodic set whose reservation sits above the classical rate-monotonic utilization bound and below one.

The design targets one processor and implicit-deadline reservations ( $D = P$ ). Section 8 defines the deployment and experimental scope.

## 2 Background and related work

### 2.1 EDF, servers, and fixed priority

For periodic tasks whose deadlines equal their periods, Liu and Layland showed that fixed priorities ordered by rate are optimal among fixed-priority rules, and that a deadline-driven rule can use the whole processor when total utilization is at most one [1]. Their rate-monotonic test  $n(2^{1/n} - 1)$  is sufficient under that paper’s assumptions. For three tasks it is about 0.780. Crossing it does not by itself predict a miss, and staying under it does not certify a kernel that adds ticks, servers, and priority demotion.

The constant-bandwidth server gives a served task a budget  $Q$  and a period  $P$  and postpones the scheduling deadline when the budget is exhausted, so one task’s demand is held to that share [2]. A later analysis treats self-suspending tasks, which the original server did not cover, and discusses how to choose  $Q$  and  $P$  for them [10]. That paper is not a specification of every constrained-deadline path in Linux. Linux `SCHED_DEADLINE` selects by earliest deadline among deadlines maintained by a server, with a runtime, a period, a deadline, and admission control [3], [4]. Its documentation describes at most one runtime per period, a wake-up density check, and a single period-and-runtime update at replenishment [4]. Bandwidth reclaiming (GRUB) is a different algorithm and is not implemented here. The documented Linux default allows deadline tasks at most 95 percent of the processors in each root domain [4]. The NuttX ledger uses 950000 ppm of one CPU. The ratio matches that default on one processor. The code is not a port of the Linux class.

### 2.2 Servers in other open-source kernels

RTEMS documents a Constant Bandwidth Server in which each server has a budget and a period, at most one task is attached, and a budget overrun drops that task to background priority until the next period [7]. That is a priority drop, not a removal from an EDF ready queue. Zephyr’s `CONFIG_SCHED_DEADLINE` does not implement that mechanism and is not Linux’s scheduling class: the highest static priority still wins, and the deadline only orders equal priorities, through `k_thread_deadline_set` [8]. Paschoaletto, Sousa, Pinho, and Carvalho describe a distinct Zephyr component, one CBS instance per task, together with richer EDF support, aimed at aperiodic soft real-time work and checked with test cases [9]. Upstream NuttX, in the manual current when it was read on 2026-10-11, documents strict priority, optional round-robin, and optional sporadic scheduling, without describing a deadline-based scheduling policy [5]. A project issue discusses EDF as future work and refers to Linux as an example [6].

Table 1: Documented EDF and reservation capabilities in open-source kernels. Distinctions refer to the cited upstream manuals and the research prototype, not every possible add-on [4,5,7,8].

| System          | Deadline dispatch                      | Server / budget behavior                 | Admission control                      |
|-----------------|----------------------------------------|------------------------------------------|----------------------------------------|
| Linux           | EDF class                              | Per-task CBS; throttle and replenish     | Bandwidth cap (95% default)            |
| RTEMS           | EDF scheduler                          | CBS API; overrun demotes task            | Global test not specified in cited API |
| Zephyr          | EDF tie-break at equal static priority | No CBS in documented scheduler option    | No EDF reservation test in manual      |
| Upstream NuttX  | Strict priority; sporadic option       | Sporadic budget causes priority demotion | No EDF reservation API in manual       |
| NuttX prototype | Absolute-deadline EDF                  | CBS throttle, replenish and overrun debt | 95% ledger, $D = P$                    |

The integration in this paper uses NuttX’s uniprocessor ready list. Eligible deadline tasks are ordered by absolute deadline and a sequence number. A per-task server accounts the task that is actually running and throttles it when the budget is exhausted. The comparison policy in the second experiment is NuttX sporadic scheduling, whose budget and replenishment mechanism changes a thread’s priority [5], building on the sporadic-server literature [11]. It is not rate-monotonic FIFO and not a constant-bandwidth server.

### 3 Integration architecture

The policy is compiled for one processor (`CONFIG_SMP` is not set on the measured configuration). Eligible deadline tasks precede FIFO, round-robin, and sporadic tasks. Two eligible deadline tasks are ordered by absolute deadline and then by a monotonic sequence number. Equal keys leave the already queued task in place. An ineligible deadline task is not inserted. The comparison is `nxsched_runnable_precedes`.

Accounting charges a task only when it is marked executing and eligible. A zero-length interval does not debit. Time beyond the remaining budget becomes overrun and the remaining budget becomes zero. If the scheduler lock is held, exhaustion sets a pending throttle and the task stays eligible until the lock drops. The pending throttle is applied on the unlock transition. Otherwise the task is detached and marked throttled. This deferred transition preserves the scheduler-lock contract and charges any excess execution as debt against future budget grants.

The user entry point is `sched_setattr`. `sched_setscheduler` is not the deadline entry and returns an error for this policy. Fork of a deadline parent is refused. The measured image is a flat build with a 1 ms periodic tick (`CONFIG_USEC_PER_TICK=1000`). This configuration is the basis for the reported measurements.

Admission rejects a new reservation whose ceiling charge would pass 950000 ppm. The charge of one task is  $\lceil 1000000 \times \text{runtime}/\text{period} \rceil$ . The ledger stores the sum of those ceilings and rejects reservations that exceed the cap. Runtime charging and replenishment then enforce the admitted execution budgets, as characterized in Sections 4 and 6.

The benchmark releases periodic jobs from a watchdog. Response time is completion minus the nominal release on `CLOCK_MONOTONIC`. A miss is a completion after nominal release plus the period. Each job records a checksum of a fixed-iteration mix. Deadline tracing, when enabled, writes a fixed ring of 6144 events. The measured image enables that ring. The ring copies the overrun counter into the event. That copy is not an input to a charge, a replenishment, or an admission decision. Section 7 and the appendix identify the firmware that contains it.

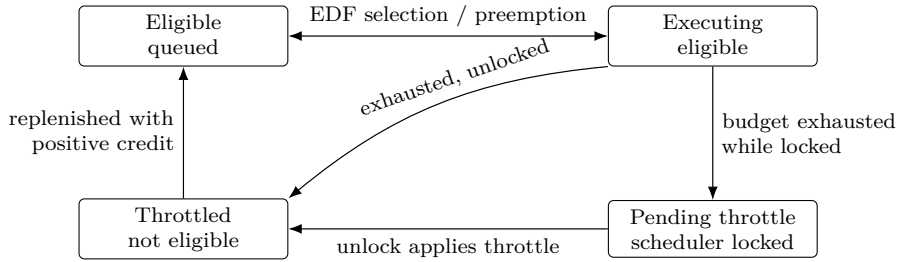


Figure 1: Conceptual EDF/CBS eligibility transitions. A replenishment that only reduces accumulated overrun debt leaves the server throttled; “pending throttle” defers removal until the scheduler lock is released.

### 4 CBS rules and API scope

The supported contract is implicit deadlines on one processor. The measured source implements the following explicit reservation, replenishment, and debt-accounting rules. These rules provide testable contracts for the transition checks and the execution traces.

**Admission.** A reservation is accepted only when the runtime is positive, the runtime does not exceed the deadline, and the deadline equals the period, and only when the ceiling charge fits under the cap. Both the attribute check used by `sched_setattr` and the reserve function apply that test before any ledger write. A rejected update does not replace the previous object. In the injected-time oracle, a request of 10 ms runtime, 20 ms deadline, and 40 ms period returns `-EINVAL` on an empty task and leaves the ledger at 0. After a legal 10 ms / 40 ms / 40 ms reservation has been admitted at 250000 ppm, the same illegal update returns `-EINVAL` and leaves both the object and the 250000 ppm charge in place. Release returns the ledger to 0. The same oracle rejects a reservation that would pass the cap (`-EBUSY`) when deadline equals period.

**Replenishment.** When a callback is due, the period is positive, and `now + period` fits in a 64-bit integer, the absolute deadline becomes `now + period` and one budget is posted. Missed periods are not banked. A second call at the same timestamp sees a deadline in the future and posts nothing. If `now + period` overflows, the call changes nothing. For a debt-free server with  $Q = 10$  ms and  $P = 40$  ms, a callback at 125 ms leaves 10 ms of credit and an absolute deadline of 165 ms, so the instantaneous ratio is 10/40.

**Debt.** One posted budget pays accumulated overrun at most once, and pays at most one  $Q$ . If the overrun  $d$  is below  $Q$ , the new remaining budget is  $Q - d$  and the overrun becomes 0. If  $d$  is at least  $Q$ , the remaining

budget stays 0, the overrun decreases by one  $Q$ , and the server stays throttled. The injected-time witness uses  $Q = 10$  ms and  $P = D = 40$  ms. Thirteen milliseconds executed while the scheduler is locked leave 3 ms of debt; the next grant is at most 7 ms. Twenty-six milliseconds leave 16 ms of debt; the next post does not make the server eligible and leaves 6 ms of debt; the following post leaves 4 ms of credit at absolute deadline 120 ms, which is `now + period` at that post. These rows are checks of the payment function. They are not the isolation experiment.

**Design scope and API contract.** Execution need not be at most  $Q$  in every arbitrary window of length  $P$ . A callback that runs late posts one  $Q$  at `now + P`, so two grants can sit slightly farther apart than  $P$ , or, when a tick delays the callback, a little more than  $P$ . A non-preemptible interval can consume more than  $Q$  before unlock; that excess is debt, not a discarded sample. Constrained deadlines  $D < P$  are not accepted. A sum of  $Q/P$  under the cap is a utilization cap for the implicit-deadline tasks the API allows. It is not a feasibility test for task sets the API rejects, and it is not a proof that every admitted set meets its deadlines once tick overhead and locks are present.

**Deterministic conformance check.** The independent oracle is a set of expected state transitions derived from the reservation and CBS contracts. An injected-time test harness calls production accounting, replenishment, admission, and scheduler-lock release functions against a private runnable queue. The comparator checks the resulting state and scheduling transitions against 49 expected records; all 49 matched. The comparator detected a deliberately corrupted captured queue head. A separate host program implementing the older “replace remaining by  $Q$  and leave the debt” behavior fails the same numerical checks and exits with status 1. The check covers those helper transitions, not all interactions with NuttX’s global ready list.

On a simulator built from this source with a 1 ms tick, the deadline test passed, including the debt rows, the rejected constrained update, the 125 ms witness, and one interrupt epilogue: running pid 63, ready-list head 62, the running task’s budget decreased, and the ready task’s budget stayed 20000000 ns. That is one epilogue. A 10 ms tick boundary case that had failed on an earlier image was not re-run. It remains unverified.

## 5 Experimental method

The guest is QEMU 10.2.1, one Cortex-A7 virtual CPU, single-threaded TCG (`-smp 1`), and instruction-count time (`-icount` with `shift=6`, `sleep=off`, `no align`). The image is a raw BIOS. It is not loaded with `-kernel`. In this mode the virtual clock advances by 64 ns of guest time per executed guest instruction, and idle time warps to the next virtual timer. Host scheduling of the QEMU process does not add to that clock. This controlled clock makes scheduler-state transitions and same-image job measurements reproducible without host scheduling jitter. Its interpretation and hardware-fidelity boundaries are detailed in Section 8.

Each QEMU process was started paused inside a memory cgroup of 536870912 bytes with swap disabled, and was continued only after those limits were read back. One guest ran at a time. Host wall time is a containment figure. The guest clock is the virtual clock.

Jobs of the protected tasks, and every job of the comparison in Section 7, are released by the same watchdog path. The mix function is deterministic for a given iteration count, so equal checksums mean the same mix output. Iteration counts for the isolation fixture were chosen on this image by a FIFO probe at priority 240, then frozen. The stress90 iteration counts were frozen from an earlier calibration and are reported against this image’s FIFO probe in Section 7.

The traced image is the only firmware in Sections 6 and 7. Its configuration enables `CONFIG_SCHED_DEADLINE_TRACE`. The appendix identifies the measured firmware, exact source revision, and archived trace artifacts.

## 6 CBS budget enforcement under overload

### 6.1 Workload

The protected tasks are periodic and implicit-deadline. The overrunning task is a deadline task that repeatedly executes a kernel requiring approximately 60 ms of virtual work until the end of the window. Its offered workload is continuous rather than a sequence of 20 independently completed periodic jobs: the worker waits for the next semaphore only after the kernel returns, and a 60 ms kernel executing against a 10 ms grant cannot complete 20 jobs within the window. The loop condition is checked between kernels, so the last kernel can extend past the window by at most one offered chunk. The controller waits three virtual seconds past the window and would delete the task if it had not exited. It exited on its own.

Table 2: Task parameters and offered workload in the CBS overload experiment.

| Task | Role                              | $P = D$ | $Q$   | Offered work                                 | Jobs in five hyperperiods |
|------|-----------------------------------|---------|-------|----------------------------------------------|---------------------------|
| A    | protected                         | 40 ms   | 14 ms | 15178 iterations, about 9.007 ms             | 45                        |
| B    | protected                         | 60 ms   | 20 ms | 23611 iterations, about 14.007 ms            | 30                        |
| C    | overrunning task, second run only | 90 ms   | 10 ms | 101191 iterations, about 60.006 ms, repeated | not a job table           |

The hyperperiod is  $\text{lcm}(40, 60, 90) = 360$  ms. Five hyperperiods are 1.8 s of virtual time. Reserved utilization with C is  $14/40 + 20/60 + 10/90 = 0.794444\dots$ . The ceiling charges are 350000, 333334, and 111112 ppm, sum 794446, under the cap. The control run admits only A and B, 683334 ppm. A and B use the same iteration counts in both runs. The offered times are the FIFO probe’s median samples on this image, not a deadline-server debit and not a WCET.

## 6.2 Protected jobs

Table 3 counts completed jobs, deadline misses, the single checksum of every job of that task, and the minimum and maximum response. Response is nominal release to completion and includes waiting. We also compare same-index releases to separate changes in watchdog posting time from the protected tasks’ execution demand.

Table 3: Protected jobs on the traced image; virtual time, one boot per condition.

| Run      | Task | Jobs | Misses | Checksum   | Response min (ns) | Response max (ns) |
|----------|------|------|--------|------------|-------------------|-------------------|
| Control  | A    | 45   | 0      | 109396200  | 9552576           | 9641408           |
| Control  | B    | 30   | 0      | 1140539847 | 14655424          | 24025152          |
| Overload | A    | 45   | 0      | 109396200  | 9355520           | 9503360           |
| Overload | B    | 30   | 0      | 1140539847 | 14455168          | 23882304          |

Every per-job checksum matches across the two runs. The maxima are inside the 40 ms and 60 ms deadlines. The worst responses are 9.641408 ms and 24.025152 ms in the control run, and 9.503360 ms and 23.882304 ms in the overload run. After both runs the admission print is 0 ppm.

**Paired release-phase analysis.** The watchdog’s mean post lateness was 0.293837 ms (A) and 0.277197 ms (B) in the control boot, versus 0.097213 ms (A) and 0.077675 ms (B) with the overrunning task. For all 45 A jobs and all 30 B jobs, response time was lower in the latter boot (mean same-index difference  $-0.187803$  and  $-0.182195$  ms, respectively). By contrast, the mean start-to-completion interval was slightly *longer* with the overrunning task (A: 9.198498 vs 9.194155 ms; B: 14.303514 vs 14.291424 ms). The phase analysis therefore attributes most of the lower nominal-release response times to earlier watchdog posts rather than faster protected computation. This avoids conflating release timing with the CBS enforcement effect.

The two measurements address distinct properties: the job table establishes that protected deadlines were met, while the trace in Section 6.3 directly demonstrates that the competing server’s consumed budget was accounted, throttled, and replenished.

## 6.3 The overrunning task’s server

C is pid 8. The ring finished with `used = 2714`, `overwritten = 0`, 25 throttle events, and 25 replenishment events. The observation window is  $t_0$  to  $t_0 + 1.8$  s, with  $t_0 = 279947456$  ns and window end 2079947456 ns. Inside that window the listing contains 20 throttles and 19 later replenishments, all for pid 8. The other 5 throttles and 6 replenishments have timestamps after the window, while the last offered kernel was still being



Figure 2: Same-index protected-job response differences between the overload and control boots. Negative numbers are mainly associated with earlier watchdog posts, not a benefit from overload.

retired. They were not overwritten. The in-window lines are the ones used below. Charge, wake, and switch lines are a sample of at most 64; throttle and replenishment lines in the window are listed in full. The window contained 2198 events.

Every listed throttle has remaining budget 0, state throttled, and the flag that the task was live and was removed from the ready queue. The first pair is the debt rule on a tick quantum, not a `sched_lock()` interval:

- At 314023616 ns the previous remaining budget is 95040 ns. The charge debits 1000000 ns, remaining becomes 0, and overrun becomes 904960 ns.  $95040 + 904960 = 1000000$ .
- At 314035264 ns, 11648 ns later, the throttle is recorded: remaining 0, overrun 904960 ns, state throttled.
- At 371030592 ns the replenishment leaves remaining 9095040 ns and overrun 0.  $10000000 - 904960 = 9095040$ .

The charge line still shows the eligible state because the debit is stored before the throttle. The task does not continue as an eligible deadline task on that exhausted grant: the next recorded event is the throttle, then a switch away from pid 8. The largest listed overrun on a throttle is 904960 ns, under one 1 ms tick. Across the 19 in-window pairs, remaining after replenishment equals  $Q$  minus that overrun, the new overrun is 0, and no replenishment posts more than one  $Q$ . Remaining after those replenishments lies between 9095040 ns and 9998912 ns.

Successive replenishment deadlines move by 90980416 ns to 91019584 ns, about 91 ms rather than a banked multiple of 90 ms. The callback posts one  $Q$  at `now + P`. A late tick moves the next absolute deadline with `now`. For this  $P = 90$  ms server, the observed spacing of about 90.98–91.02 ms corresponds to roughly 1.1% less average grant frequency than an ideal 90 ms period, assuming one grant per callback. This is a quantified limitation of the observed trace, not a measurement of short-period servers. The replenishment event is written before the server is marked eligible again, so a replenishment line can still show the throttled state together with a positive remaining budget. Later samples between C’s kernels show the eligible state and a positive remaining budget, and the next live throttle shows that the server ran and was removed again.

The experiment directly connects three CBS behaviors: an exhausted budget removes C from EDF eligibility, overshoot is retained as debt, and the next replenishment subtracts that debt from the posted grant. Throughout this interference, A and B met all 75 deadlines. The combined reserved utilization of A, B, and C was approximately 0.794, with C limited to a nominal share of 0.111. The CBS trace documents budget enforcement, and the protected-job table establishes that all deadlines were met in this experiment. Section 8 discusses the limits of causal attribution and generalization.

The 904960 ns overshoot is the periodic tick charging a full millisecond after 95040 ns remained. It is separate from the locked-overrun rows in Section 4, where `sched_lock()` prevents the throttle until unlock and the debt is paid on later grants. Both end as `overrun_ns`. They are not the same interval.

## 7 A higher-utilization comparison

### 7.1 Task set and sporadic parameters

The second workload uses the same traced image and the iteration counts 79266, 160219, and 263097. Those counts were not recalibrated here. Period equals deadline. Ten hyperperiods of 1800 ms are 18 s of virtual

time and  $90 + 60 + 40 = 190$  jobs per policy. One sporadic boot and one deadline boot were run. This is one pair.

Table 4 lists the `SCHED_SPORADIC` (policy 3) parameters accepted by the fixture. The apply path aborts unless the readback matches these values, including three replenishments. The console prints policy, high priority, low priority, budget, and period. The replenishment count is the value that path requires.

Table 4: Sporadic scheduling parameters accepted by the comparison fixture.

| Task | High priority | Low priority | Budget | Replenishment<br>period | Max repl. |
|------|---------------|--------------|--------|-------------------------|-----------|
| T1   | 220           | 50           | 50 ms  | 200 ms                  | 3         |
| T2   | 210           | 50           | 100 ms | 300 ms                  | 3         |
| T3   | 200           | 50           | 164 ms | 450 ms                  | 3         |

Deadline tasks are created at priority 50 and ordered by deadline. The readback on the deadline boot is T1 runtime 50000000 ns, deadline and period 200000000 ns; T2 100000000 / 300000000 / 300000000; T3 164000000 / 450000000 / 450000000. The ledger at arm is 947779 ppm, which is the sum of the ceilings of 50/200, 100/300, and 164/450. After the run it prints 0.

## 7.2 What the classical tests say

Reserved utilization is  $50/200 + 100/300 + 164/450 = 853/900 \approx 0.947778$ . Nominal work of 47, 95, and 156 ms is  $1617/1800 \approx 0.898333$ . This image’s FIFO probe, 843267 iterations in 500001472 ns, implies about 46.999 ms, 94.999 ms, and 155.999 ms for the frozen counts. That implication is a scale of the same mix, not a 21-sample isolated maximum of these three kernels on this boot, and not a WCET.

Under Liu and Layland’s assumptions the three-task rate-monotonic bound is about 0.780. Both the reservation and the nominal work sit above it, so the sufficient test does not pass. The classical fixed-priority response-time recurrence [12] can check each possible lowest-priority task, using the implied virtual execution costs  $C = (47, 95, 156)$  ms. With T1 lowest, its response is at least  $47 + 95 + 156 = 298 > 200$  ms. With T2 lowest, iteration yields 298 then  $345 > 300$  ms. With T3 lowest, iteration yields 298, 345, 440,  $487 > 450$  ms. Thus no assignment of distinct static priorities is schedulable *under those assumed execution costs and the standard fixed-priority model*. These inferred costs explain workload selection: the utilization is high enough to stress fixed priorities, while ideal uniprocessor EDF has schedulability slack for an implicit-deadline set with utilization about 0.898 under the classical assumptions [1]. The observed kernel outcomes are reported independently in Section 7.3.

NuttX sporadic scheduling is not that rate-monotonic system. A job can be demoted to priority 50 for part of its replenishment period, which is worse than remaining at 200, 210, or 220. The response-time recurrence [12] does not model that demotion, the limit of three replenishments, or the 1 ms tick. Audsley-style assignment analysis [13] concerns ideal fixed priorities; it does not model the measured NuttX sporadic server. The recurrence thus motivates the stress case; kernel-level comparison is provided by the paired run below.

## 7.3 Measured pair

Table 5 reports one matched pair on the traced image. Checksums match across policies.

Table 5: Measured sporadic-versus-EDF/CBS pair on the traced image.

| Policy   | Jobs T1/T2/T3 | Misses T1/T2/T3 | Checksums<br>T1/T2/T3                      | Response maxima<br>(ns)                 |
|----------|---------------|-----------------|--------------------------------------------|-----------------------------------------|
| Sporadic | 90/60/40      | 0/0/4           | 2280062158 /<br>4148732080 /<br>3182598815 | 97729920 /<br>275916032 /<br>490190336  |
| Deadline | 90/60/40      | 0/0/0           | same                                       | 103521536 /<br>203307392 /<br>353150464 |

The four sporadic misses are all T3, at job indexes 0, 4, 8, and 36. Tardiness is 40131328, 40190336, 40158784, and 4953600 ns. Job 0 started and ended at high priority 200. Jobs 4 and 8 started at low priority 50 and ended at 200. Job 36 started and ended at 50. Those priority columns show demotion on three of the

four misses. They do not identify every sporadic replenishment. The deadline policy’s longest T3 response is 353150464 ns, inside the 450 ms deadline.

The deadline boot’s trace ring is full (`used` = 6144) and records 12619 overwrites. Its summary window is empty because this scenario opens a window only around a miss, and this boot had no miss. A zero throttle count in that surviving tail is not a count of throttles. The miss table does not depend on it. The sporadic boot’s ring contains switches and no CBS charges, which is consistent with those tasks not being deadline servers. It is not a CBS measurement.

On the matched firmware, the sporadic policy’s demotion behavior coincides with four T3 deadline misses, while EDF/CBS meets every reported deadline on the same 190-job workload. Checksums match across policies, and the EDF/CBS maximum T3 response remains below its 450 ms deadline. This is direct comparative evidence for the two implemented scheduling paths under a controlled high-utilization load.

## 8 Threats to validity and implementation limits

The clock is an instruction count. Response times, the 904960 ns overrun, and the tardiness figures are quantities on that clock. Cache effects, board latency, and a hardware WCET are not in the data. QEMU’s instruction-time fidelity, relative to a Cortex-A7, was not measured.

The budget-enforcement experiment comprises two boots of one fixture. The comparison comprises one boot per policy for one task set, so it is not a statistical performance study: no independent run ensemble or confidence interval is available. The protected-task test does not include an unthrottled counterfactual. A matched run with budget enforcement disabled would help separate its contribution from EDF ordering. Earlier-image miss patterns are not samples of the traced firmware.

Sporadic scheduling demotes threads. Mapping it to rate monotonic is an illustration, not an identity. The isolation trace shows CBS throttle and replenishment for the overrunning task. The high-utilization deadline boot does not, because its ring overwrote its prefix and the report window was empty.

The tested firmware can place eligible deadline threads ahead of FIFO/RR/sporadic tasks, including system workers; interactions with kernel work queues and cross-policy lock holders were not assessed. The oracle uses a private queue. The interrupt result is one simulator epilogue. The 10 ms tick boundary case was not re-run. Scheduler overhead was not measured as a separate quantity. The traced firmware differs from earlier non-traced images, so subtracting their worst response times would confound instrumentation, release phase, and build differences; no isolated tracing-overhead estimate is claimed. Priority inheritance on mutexes held by a throttled deadline task was not exercised. Kernel worker and work-queue latency were not measured. Tickless operation, protected builds, kernel builds, and SMP were not the configurations that produced the tables. A non-preemptible lock can exceed  $Q$  before the throttle; the debt rule pays that excess later and does not make the lock interval itself respect  $Q$ .

Admission of  $D = P$  reservations under the cap does not prove that every such set meets its deadlines. The experiments show two specific sets on one virtual clock.

## 9 Conclusion

This work adds an EDF/CBS scheduling path to uniprocessor Apache NuttX, combining absolute-deadline dispatch with bounded execution reservations, deferred-throttle handling, and explicit repayment of overrun debt. The integrated experiments show the mechanism in action: 75 protected jobs completed within their deadlines while a competing server was repeatedly throttled and replenished. On a matched high-utilization workload, EDF/CBS completed all 190 jobs within their deadlines, compared with four misses under NuttX sporadic scheduling. The implementation, transition checks, and archived deterministic virtual-time experiments establish a reproducible foundation for further development and evaluation of deadline-based scheduling in small real-time kernels.

## Funding and Competing Interests

**Funding.** This research was conducted independently by the author.

**Competing interests.** The author is the founder of ExtenSilica. ExtenSilica has no commercial interest in the research presented in this paper, and the findings are intended for public dissemination. The author declares no competing interests related to this work.

## References

- [1] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. <https://doi.org/10.1145/321738.321743>
- [2] L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” in *Proc. 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998, pp. 4–13. <https://doi.org/10.1109/REAL.1998.739726>
- [3] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, “Deadline scheduling in the Linux kernel,” *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016. <https://doi.org/10.1002/spe.2335>
- [4] The kernel development community, “Deadline task scheduling,” *Linux kernel documentation*, v6.1, retrieved 2026-10-11. <https://docs.kernel.org/6.1/scheduler/sched-deadline.html>
- [5] The Apache Software Foundation, “Task scheduling interfaces,” *Apache NuttX documentation*, retrieved 2026-10-11. [https://nuttx.apache.org/docs/latest/reference/user/02\\_task\\_scheduling.html](https://nuttx.apache.org/docs/latest/reference/user/02_task_scheduling.html)
- [6] Apache NuttX, issue 9712, “Help wanted for testing and improving real time performance,” GitHub, retrieved 2026-10-11. <https://github.com/apache/nuttX/issues/9712>
- [7] RTEMS Project, “Constant Bandwidth Server Scheduler API,” *RTEMS Classic API Guide*, version 7.5ece2a4, 12 June 2026. [https://docs.rtems.org/docs/main/c-user/constant\\_bandwidth\\_server.html](https://docs.rtems.org/docs/main/c-user/constant_bandwidth_server.html)
- [8] Zephyr Project, “Scheduling,” retrieved 2026-10-11. <https://docs.zephyrproject.org/latest/kernel/services/scheduling/index.html>
- [9] A. Paschoaletto, P. B. Sousa, L. M. Pinho, and T. Carvalho, “Task-Based Constant Bandwidth Server in the Zephyr Operating System,” in *Proc. 30th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2026)*, Open Access Series in Informatics (OASiCs), 16 pages, publication date 3 June 2026. <https://doi.org/10.4230/OASiCs.AEiC.2026.6>
- [10] L. Abeni, G. Lipari, and J. Lelli, “Constant bandwidth server revisited,” *ACM SIGBED Review*, vol. 11, no. 4, pp. 19–24, 22 Jan. 2015. <https://doi.org/10.1145/2724942.2724945>
- [11] B. Sprunt, L. Sha, and J. Lehoczky, “Aperiodic task scheduling for hard-real-time systems,” *Real-Time Systems*, vol. 1, pp. 27–60, 1989. <https://doi.org/10.1007/BF02341920>
- [12] M. Joseph and P. Pandya, “Finding response times in a real-time system,” *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986. <https://doi.org/10.1093/comjnl/29.5.390>
- [13] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, “Applying new scheduling theory to static priority pre-emptive scheduling,” *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993. <https://doi.org/10.1049/sej.1993.0034>

## Appendix: Artifact Identity

The research artifacts are publicly archived as version **v0.1.0-research** at Zenodo: DOI: 10.5281/zenodo.23332003. The related source repository and research release are available at GitHub release v0.1.0-research. The exported source snapshot can be restored against the pinned NuttX and Apps commits. Reproducibility targets the exported sources, configuration, and virtual-time experiment; byte-identical binary generation depends on the original build environment.

The results in Sections 6 and 7 are the traced image only. The hash values here are abbreviated for readability; the full SHA-256 values are preserved in the accompanying **ARTIFACT\_MANIFEST.md**.

| Piece                    | Identity            |
|--------------------------|---------------------|
| NuttX base               | f0240da863ed0fe1... |
| Apps base                | 1f8cac8b87a92540... |
| Traced ELF               | f5205bc3cd5f4b93... |
| Raw BIOS                 | 4b469a682d3cf034... |
| Configuration            | f4d3bef80c8a0a4f... |
| CBS source in that image | 23826ea038208530... |

The configuration sets **CONFIG\_SCHED\_DEADLINE\_TRACE**. Relative to the CBS file 12b3c4922dda277e..., the difference in the booted CBS file is that trace record: each slot stores **overrun\_ns**, and the report lists throttle, replenishment, and pending events in the requested window. Payment and admission are the semantics already in the earlier hash. The historical patches alone do not reconstruct this image, but the complete exported source snapshot, including the final trace-only CBS source and isolation benchmark, is now included in the artifact archive under **source\_export/**. The complete file hashes, original firmware and five boot logs accompany it. This supports source-level reconstruction relative to the pinned bases, but not an identical binary build. The Zenodo DOI above identifies the public artifact record for this research release.