Reviewer A

- p2, line 60: assumptions on control flow → assumptions on the task control flow
- p2, line 60: the segmented model is that is allows for → the segmented model is that **it** allows for
- p4, line112: I would say segmented self-suspending tasks rather than multi-segment tasks to remain consistent with the naming introduced in the system model.
- p7, line 212: to enrich the period enforcer with the following power, consider replacing power with "property" or "scheduling rule".
- p7, line 221: "it is immediately obvious that" → I would simply remove that part of the sentence since nothing is obvious with self-suspending tasks and particularly for the problem addressed in this paper.

  Response: Thank you for the suggestions. We have rephrased those parts accordingly.

Reviewer B

- First, the period enforcer analysis only applies to the case where a task defers its entire execution time but not parts of it. Tasks that suspend after executing for a while should be considered as two or more tasks with their own worst-case execution time. It is known that if tasks are broken up into multiple segments with individual relative deadlines adding up to the original deadline, the schedulability is no longer guaranteed. For example, a task set (C,T,D), with tasks (4, 5, 5) and (2, 10, 10) is schedulable, whereas breaking up the (2, 10, 10) into (1.5, 10, 7.5) and (0.5, 10, 2.5) is not schedulable.
- Section 3 should address the above clarification and the task τ_2 should be statically broken up before applying the period enforcer. It is true that transforming a task set by breaking up tasks into sub tasks with sub deadlines can lead to unschedulability on an otherwise schedulable task set.

  Response: We must distinguish two cases: the period enforcer *runtime rules*, and the *classic analysis* of the period enforcer.

  Section 3 is concerned only with the runtime rules. Specifically, it shows that the runtime rules may cause otherwise schedulable sets of multi-segment self-suspending tasks to become unschedulable. Importantly, the runtime rules do not require a conversion to single-segment deferrable tasks.

  The conversion to multi-segment deferrable tasks is relevant *only* in the context of schedulability analysis. However, section 3 does not discuss schedulability analysis.

  To elaborate a bit more, since the period enforcer *analysis* only applies to the case where a task defers its entire execution time but not parts of it, we should create the corresponding deferrable task set to carry out schedulability analysis. However, it is *not* necessary to create those deferrable task sets for *using* the period enforcer algorithm at runtime: since the computation segments of a segmented self-suspending task are independent from each other, we can simply use the period enforcer algorithm without knowing the corresponding deferrable task set. This is actually how we were able to create the examples in Section 2 and Section 3.

Actions taken: Section 2 was rewritten and expanded to clearly distinguish between the runtime rule (Section 2.2) and the classic analysis (Section 2.3). The runtime rule — Equation (1) — is stated in terms of multi-segmented deferrable tasks.

Actions taken: Section 4 was completely rewritten to more clearly explain that the problem of finding a suitable corresponding task set is open, and likely difficult.

- Section 4 should be reworded. 'precisely converting a self-suspending task system into a corresponding single-segment deferrable task set is not an easy problem' → 'converting a self-suspending tasks system into a single-segment deferrable task set introduced pessimism due to period enforcement'.

  Response: We do not fully understand this recommendation. In any case, we have completely rewritten this section, which we hope that our revision addresses this point as well.

- It is unclear how the Nelissen paper helps since it does not assume period enforcement. If each self-suspending task is broken into deferrable sub tasks (considering all the higher priority tasks as non-suspending), why is the Nelissen analysis needed? Also, It should be clarified as to why the problem of precisely computing W_k^j is useful in the first place since period enforcement would introduce pessimism anyway (as shown in Section 3).

  Response: We need the analysis from Nelissen et al. to precisely break a self-suspending task into multiple deferrable subtasks so that we can calculate the worst-case release jitter of a computation segment. Section 4 explains why it is difficult to precisely calculate the release jitter of the computation segments and clearly points out the connection to Nelissen et al.'s analysis.

  Actions taken: In the revision, we completely rewrote Section 4 to explain the gap between the segmented self-suspending task model and the corresponding deferrable sporadic task set, and to precisely state the equivalence of the conversion to the worst-case response-time analysis of individual computation segments (i.e., Nelissen et al.'s analysis).

- Section 5 should also include the scenario where period enforcement is only applicable to the normal execution segments. It seems practical to not enforce critical sections since they are usually short in most systems i.e. the back-to-back execution overhead from most critical sections in real-world scenarios should be minimal. The case 3 that should be discussed is period enforcement applied to execution segment after the critical section, without enforcing the critical section and just including its back-to-back execution penalty in the worst-case analysis.

  Response: Thank you for the suggestion; this is indeed an interesting option to explore. However, we believe this to be out of the scope of this paper.

  Prior work has explicitly suggested to use the period enforcer to control self-suspensions caused by the MPCP and the DPCP such that there is *no* scheduling penalty, which is only possible if the period enforcer is applied to critical sections. Specifically, this is assumed in Rajkumar's 1991 book [27], and also mentioned in more recent works by his group. In fact, Rajkumar writes in the original period enforcer proposal (emphasis added):

  > "The scheduling penalty of deferred execution [when using synchronization protocols] can be enormous to the extent that it can make such protocols almost useless in several cases. *The development of the period enforcer algorithm was primarily motivated by the need to reduce or eliminate this penalty* with a small, if not negligible, overhead. If tasks using these synchronization protocols use the period enforcer,

their worst-case blocking durations would be considerably smaller and the schedulable utilization correspondingly higher." [26, Section 5.1]

Our intention here is to document that this simply does not work. We do not claim to evaluate all possible ways in which the period enforcer might be used otherwise.

Actions taken: Section 5.4 has been extended to briefly comment on this third possible interpretation. We now also comment on distributed semaphore protocols (e.g., DPCP, DFLP).

Reviewer C

- This reviewer was not able to download a complete version of [20] including the figures (the .ps available online is highly corrupted). Please provide a precise link to the full version of this paper or a way to access to it.

  Response: We are not able to provide a complete version of the original paper since we are not its authors. In the preparation of the paper under review, we also relied on the same online version (without figures) to study the period enforcer algorithm. While figures are indeed missing, they can be inferred from the accompanying detailed descriptions; the rest of the paper is not corrupted.

  Actions taken: In the revision, we have expanded and improved the description of the background material in Section 2 to make our paper much more self-contained.

Reviewer D

- This paper is heavily dependent on the assumptions and analysis of the original period enforcer algorithm in [20]. Although the authors highlighted some of the assumptions and important result (e.g., Theorem 5) from [20], it may be difficult for readers to understand this paper without reading in details the analysis from [20]. The reviewer had difficulty in understanding the discussion in section 2.3 and 2.4. After the reviewer read [20], the discussion in section 2.3 and section 2.4 becomes clearer. Important details from [20] need to be added to this paper to make it more readable and self-contained.

  Actions taken: We have expanded the discussion of necessary background in Section 2. Sections 2.3 and 2.4 have been edited and clarified to make the discussion self-contained.

- There are some terms in section 2.3 and 2.4 that the authors have used assuming that readers are already familiar with those terms. For example, the terms deferrable task, task set transformation, single-segment deferrable tasks, non-self-suspending sporadic task, etc. may be formally defined before using them. For example, the set of transformed non-self-suspending tasks subject to release jitter (section 2.4) essentially means a set of deferrable tasks. Since Theorem 5 uses the term deferrable task, the relationship between transformed task set and deferrable task set may be explicitly stated for better understanding.

  Actions taken: We added Section 2.2, which reviews in detail all relevant task models and clearly defines the terminology used in the paper.

- Theorem 5 (page 5) mentions about worst-case condition. What does it mean by worst-case condition for deferrable tasks? Paper [20] explains the worst-case. The authors could explain the worst-case or restate the theorem in a different way so that reader does not need to seek for the worst-case in [20].

  Actions taken: We added an explanation of the term "worst-case conditions" immediately after Theorem 5.

- Is there any way to quantify (theoretically or using simulation) the schedulability loss/gain using period enforcer algorithm for multi-segment self-suspending tasks? The reviewer is curious to learn the impact of the negative (theoretical) results of period enforcer algorithm on randomly generated multi-segment self-suspending task sets.

  Response: We are not able to provide a meaningful empirical characterization of the loss/gain because there is neither a precise baseline to compare against, nor a practical analysis for the period enforcer.

  1. We would be curious to learn this, too, but unfortunately we are not able to convert a segmented self-suspending task set into the corresponding deferrable task set efficiently and without information loss. In Section 4, we explain the task set transformation for a limited special case. If there are multiple self-suspending tasks, there is no existing solution for exactly constructing the deferrable task set: the original proposal [26] failed to define such a conversion, and we did not find one that is both efficient and not too pessimistic.
  2. There are also no other known methods or analyses to handle self-suspensions in the general case without incurring a potentially large degree of pessimism; we are hence not sure what to use as a fair baseline.

  Actions taken: Section 4 has been rewritten to more clearly explain the difficulty of analyzing self-suspensions (both with or without the period enforcer).

- Based on the notes of this paper, it is not clear whether one should consider period enforcer algorithm or not for self-suspending tasks.

  Response: There is currently no known schedulability analysis for the period enforcer (if any task has more than one computation segment). Until future work finds such an analysis, the period enforcer should not be considered as a solution for handling self-suspensions.

  Actions taken: We have rephrased the last paragraph of the conclusion to make this point more clearly.

- Why the dynamic self-suspension model is presented and compared with segmented model in section 2.1? It is not clear why period enforcer algorithm applies only to the segmented model. Is there any reference or argument to support this claim in line 61-62?

  Actions taken: We added Section 2.2.4 to explain this limitation.